POLITECNICO DI MILANO
Master of Science in Computer Science and Engineering
Dipartimento di Elettronica, Informazione e Bioingegneria

A Markov chain model for dependability evaluation and fault prediction on UPS systems

Supervisor: Prof. Francesco Amigoni
Co-supervisor: Prof. Letizia Tanca

M.Sc. Thesis by:
Antonio Gianola, 877235

Academic Year 2017-2018

Ai miei amici, ai miei genitori e a mia sorella.

Abstract

Nowadays, electricity is one of the key assets in most of the activities. Just think of the importance of this power to companies, hospitals, or data warehouses where a blackout of a few minutes could cause injuries, businesses disruptions, or data loss. One of the solutions most used by companies to deal with outages involves the use of Uninterruptible Power Supply (UPS), electrical systems able to provide energy to a load when the input source fails. It is redundant to specify how essential these devices are and how much dependability is crucial to the system. Up to now, this last property is mainly guaranteed by specific and sophisticated design techniques, and by log files generated automatically to keep track of how the system behaves. Usually, the files are manually managed by the maintenance service, that has the purpose of locating and replacing the damaged components. Since this approach is mostly manual and has several limitations in terms of prediction capabilities, the need came to derive automatic techniques for analysing the generated data and creating estimates on how the system will behave in the future. Without any knowledge of the domain and of the system, by using Artificial Intelligence techniques. In this thesis we build a model that tries to capture the events leading to the failures and to predict the happening of malfunctions, allowing an operator to repair the system before it fails. We focus our attention on Markov chain models, and through the thesis, we discuss the operations performed to build the models in our application. In the end, we develop an easy-to-use software program able to compute and display the models.

Sommario

L'energia elettrica è una delle risorse chiave per la maggior parte delle attività. Basta pensare alla sua importanza per le aziende, gli ospedali o i data warehouse dove un blackout di qualche minuto può causare infortuni, danni economici o perdita di dati. Una delle soluzioni più usate per contrastare le interruzioni di energia elettrica consiste nell'utilizzo dei gruppi di continuità (UPS), sistemi elettrici capaci di fornire elettricità al carico quando la sorgente non funziona. È superfluo specificare come questi dispositivi siano essenziali e come l'affidabilità sia un aspetto fondamentale di questi sistemi. Ad oggi quest'ultima proprietà è garantita da specifiche e sofisticate tecniche di progettazione e da file di registro generati automaticamente per tenere traccia del comportamento del sistema. I file solitamente vengono controllati manualmente dal servizio di manutenzione che ha il compito di sostituire eventuali componenti danneggiati. Dato che queste tecniche manuali hanno diverse limitazioni in termini di previsione dei malfunzionamenti futuri, è sorto il bisogno di creare un metodo automatico capace di analizzare i dati generati e fornire stime di come il sistema si comporterà in futuro. Senza nessuna conoscenza di dominio ed utilizzando tecniche di Intelligenza Artificiale, in questa tesi costruiremo un modello del sistema capace di catturare le dipendenze fra i guasti e prevedere l'arrivo di malfunzionamenti, permettendo quindi ad un operatore di riparare il sistema prima che si guasti. Concentreremo la nostra attenzione sulle catene di Markov e, durante la tesi, descriveremo le operazioni compiute per creare il modello nel caso della nostra applicazione a sistemi UPS. Infine, svilupperemo un programma software capace di generare e visualizzare i modelli.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our world is more and more dependent on electricity: anyone who offers us a service or produces goods needs it. Just think of the importance of this power to companies, hospitals, or data centers, where a blackout of a few minutes could cause injuries or fatalities to people, severe businesses disruptions, or data loss.

For these reasons, companies have implemented several solutions to guarantee theirs services in face of electric power shortages. One of the most used solutions involves Uninterruptible Power Supply (UPS), electrical systems able to provide energy to a load when the input source fails. These systems can increase their size easily from units designed to protect a single computer to large units powering data centers. The largest UPS is a 46-Megawatt Battery Electric Storage System, in Fairbanks, Alaska, built by ABB in 2003, and it provides power to the city during outages [4].

It is redundant to specify how these devices are essential and how the dependability is a crucial aspect for them. Up to now, this property is guaranteed by specific and sophisticated design techniques and by log files generated automatically to keep track of how the system behaves in order to repair it as soon as possible in case of failure. There are several limitations of this approach solutions; the design phase can be performed only by skilful people, without any chance to take into account the actual behaviour of the system during its operation and therefore without considering the hidden variables that affect the rising of failures. By using the log files, we must necessarily wait for a failure of the system before performing any reparation. Moreover, a manual analysis of these files could require a long time affecting the repairing time. Therefore, the need to create an automatic method able to analyse the data and provide insightful information on the working of the system.

In this thesis, we focus our attention on the log files provided by a company and produced by a complex system composed of different UPSs, with the goal of creating a behavioural model able to relate with each other the events occurring within the devices. Our objective is to generate a series of models based on Artificial Intelligence techniques, to predict the happening of malfunctions, and allow operators to preventively act on the system before it fails. In particular, we will focus on Markov chain models, statistical models capable of capturing dependencies between events occurring during time. In the thesis, we will not use any knowledge of the system, therefore all the dependencies will be extracted automatically from the data.

During the thesis, we encounter the restriction of having a limited amount of data; an initial analysis is devoted to understand how these data are generated from the UPS system and written on the log file. This problem will, in fact, limit the prediction capability of the created models.

We will associate to each other the events of a UPS system by developing a method able to extract the dependencies between them. Our goal is to create a model that includes the relations between the events occurred in a system. For each association between events, we will include some attributes able to quantify their strength. We will especially pay attention to the post-processing phase, fundamental in our context to add the maintenance actions needed to repair a UPS in case of failure and to display the transitions with graphical techniques. In the end, we will create a software program able to ask the user for the parameters and to generate a model reported as a graph in order to improve the immediacy and clarity of fruition by an operator of the company that provided the data. Through this thesis, we will demonstrate the utilisation and the effectiveness of our approach in dependability evaluation and fault prediction for UPS systems.

The literature offers interesting papers related to our work, spanning from the use of Bayesian network for dependability analysis on circuit breaker [5] to the use of a Markov chain to model the correlations between faults in a complex system [6]. Different papers use predictions techniques based on domain knowledge to elaborate a model of a UPS system [7–9]. However differently from these works, we do not have any prior knowledge of the system, but our analysis is based only on the events produced and the associations among them.

The thesis is structured as follows:

- Chapter 2 (**Preliminary notation and state of the art**): in this chapter we present the state of the art and the notation used to define the Markov chain and Hidden Markov model.
- Chapter 3 (**Problem setting**): in this chapter we define the context of the thesis, its main goals, the data provided, and the tools used.
- Chapter 4 (**Preprocessing and data analysis**): here we show the process followed to analyse the data provided by the company.
- Chapter 5 (**Markov chain model**): in this chapter we present the steps required to build a Markov chain, including we also include the postprocessing phase.
- Chapter 6 (**Implementation details and examples of use**): we describe the architecture developed and we compare how the model behaves by changing its parameters.
- Chapter 7 (**Conclusions and future work**): we briefly summarize the thesis and discuss the future steps to improve our work.

Chapter 2

Preliminary notations and state of the art

In this chapter, we review the state of the art of the most important techniques relevant to this thesis. In Section 2.1, we show the different approaches to analyse the System Dependability by comparing the Measurement-Based methods and the Model-Based methods. In Section 2.2, we discuss the different types of Model-Based methods by comparing the Combinational approach and the State-Space approach. In Section 2.3, we present the Markov models by focusing our attention on Markov Chain and Hidden Markov Models. In Section 2.4, we review the most important work related to my thesis.

2.1 System dependability

For many physical systems, one of the most important properties is the *dependability*. The dependability of a system is its ability to deliver a service that can be trusted. A service is defined as the *set of outputs* perceived by the user and a *system failure* is a deviation from the correct service and it is usually assumed that failures are caused by random events. The most critical dependability attributes are *reliability*, *availability*, *safety* and *maintainability*.

A system does not always fail in the same way, according to [10] there are three main classes of faults: *design faults*, *physical faults* and *interaction faults*. When a *design faults*, most of the time, it is generated due to designers or developers who can forget unforeseen situations of the system. A *physical* *faults* depends on the components used to build the system. The typical way to remove these faults consists of replacing the component with a new one. An *interaction faults* regards the exchange of messages among the system components. Most of these faults can be avoided by providing a deepened analysis during the design phase.

Some important tools which designers and developers of a dependable system must count on, according to [10], are: *fault prevention*, *fault tolerance*, *fault removal* and *fault forecasting*: these aspects must be considered to improve the dependability of a system. For instance, the *fault removal* concerns how to reduce the number or severity of faults and is related to monitoring and maintenance strategies. In order to minimise the maintenance cost, two policies are possible: *proactive maintenance* tries to prevent the component or system failure. While in the *reactive maintenance* the failure of the system triggers an action. The *fault tolerance* is the property of delivering the correct service in the presence of faults and it is typically reached by duplicating some components in the system, according to [11], the redundancy can be performed on different levels: *hardware*, *software*, *time* or *information*, depending on the system a specific level may be better than another. Fault prevention and forecasting are archived by using *fault identification* and *detections strategies.*

According to [12], there are two main methods to evaluate and analyse the dependability of a system:

- The *Measurement-Based* method requires to observe and to measure the behaviour of the components of a system in its operational environment; This method gives the most credible results, but it may be unfeasible or too expensive. In some cases, a copy of the system is tested to measure these parameters and, it could cause the ending of the working life of the system with money wasting.

- The *Model-Based* method is preferable in those cases where is impossible to replicate a copy of the original system. This method involves the construction of a model by defining and abstracting the main elements and properties of the system. The resulting model should be accurate enough to give a proper evaluation of the dependability.

2.2 Model-based methods

Dependability evaluation research comes up with a variety of Model-based methods. While each method relies on a specific level of abstraction and/or system characteristics, all the Model-based methods share some prior domain knowledge. We distinguish two different main paradigms of Model-Based methods: *Combinational models* and *State-Space models*.

Combinational models represent the system by using logical connections. These methods are also known as qualitative model-based representations [13]. In these models, the prior domain knowledge is composed of a fundamental understanding of the process using experience and evidential information. The model is developed thanks to the understanding of the physics of the process with the aim to capture knowledge in a formal methodology. The structure of a system is expressed by using logical interconnections of components. The notation usually is quite intuitive and easy to manipulate. Many combinational model techniques have been applied to dependability evaluation and fault detection problems: Fault Trees, formally described by [14], Reliability Block Diagrams [15], and many other models.

Combinational model techniques have also been applied to UPS systems using the Fault Tree Analysis to identify all the potential causes leading the system failure [7]. That work proposes a technique to estimate the failure rates, the mean time between failures, and the reliability of five UPS topologies. In that work, to develop the model it is required a previous knowledge of the system in terms of main sub-components and interconnections. Moreover, they don't use any specific data to compute the overall failure probability but they consider the literature probability of an event causing a failure. Also the Reliability Block Diagram method to model UPS system availability and reliability is proposed [8, 9], however, this work does not consider the causes of a failure and it is difficult to perform a fault identification analysis.

Note that It is not always possible to use a combinational technique since sometimes there is not enough prior domain knowledge to model the system behaviour. Moreover, the modelling power if this technique might be inefficient because the combinational techniques assume the statistical independence of the events. In real systems the events are not independent, in these cases, a State-Space model is more advisable.

State-space models represent the behaviour of the system in terms of reachable states and probabilistic dependence between states. We assume that there is a hidden state of the system that evolves with time, possibly as a function of the inputs, and generates the observations. The goal is to exploit the observation to infer the hidden state up to the current time. This category of models is also known as History-based methods [16]. State-Space models provide a general framework for analysing deterministic and stochastic dynamical systems that are measured or observed through a stochastic process. The prior knowledge needed, in this case, is only composed of a large set of historical process data. This knowledge is transferred to the model after a feature extraction process that has the aim to extract all the useful information from the historical data. The state space analysis may be computationally expensive because of the number of states of the model and the number of features used to elaborate it. Models as Markov chains [17], and Petri nets [18], and Bayesian networks [19] belong to this category. In all these methods, a state can be considered as a specific configuration of the system that holds for an interval of time. An event can change the system configuration and consequently the state of the model. Given a set of random variables that represent the system configuration at a specific time, the modeller has two main choices: either specifying the joint probability distribution or asserting a suitable set of independence assumptions, that can be learned from data. If we define every variable as independent of the others, the joint distribution can be easily obtained by multiplying the probabilistic parameters. However, assuming the complete independence of the variables is unrealistic for most systems. By considering a complete dependence of the variables, we create an opposite problem, in which it is computationally impossible to analyse and create the state space. The best solution consists of providing a reasonable set of information concerning the dependence and independence of the variables.

There are many published papers about dependability and fault analysis by using State-Space models. For instance [20] has studied how to develop a Bayesian network to support the fault analysis and the reliability estimation of a power system nodes architecture. The result is a versatile tool to automate the construction of models driven by probabilistic distribution of variables. A use of State-Space model for failure analysis is presented in [18], where the authors try to overcome the Combinational models by developing a Petri network, that can be constructed to represent the cause-effect relationship between events. Fault detection and dependability analysis have also been applied to intermittent faults of electronic system [21], where the many factors that may cause intermittent faults are modelled by using a Markov model. The data are taken from US military and electronic industries. The faults are divided into three categories: drift, intermittent, and burst fault and the goal of the model is to determine the true alarms and false alarms. The produced model is a three-state Markov model able to reduce the number of false alarms and increase the fault detection rate with respect to the traditional two-state model.

The approaches to dependability evaluation and fault detection described in the literature cannot be applied to our problem. We must exclude Combinational models because we have not enough domain knowledge to elaborate a model. Moreover, we cannot use directly the State-space modelling techniques because of the structure of the data, in which it is not clear which are the dependent variables and the independent variables. An interesting paper [5] proposes a Bayesian network framework for supporting the operations of circuit breakers. The aim of that paper is to introduce the use of AI tools easy to be integrated and easy to use and the input data are taken from the literature and regard the reliability of circuit breakers. We can identify some analogies between that work and this thesis, especially in the goals of the models developed. In this thesis, we use Markov chains to create an innovative model for operations and support in power-critical applications by modelling the event correlations.

2.3 Markov model

In probability theory, a Markov model is a stochastic model used to model randomly changing systems. There are four common Markov models that are used in different situations, depending on whether every state is observable or not, and on the presence of autonomous or controlled circumstances [3].

	Fully Observable	**Partially Observable**
Autonomous	Markov chain	Hidden Markov model
Controlled	Markov Decision process	Partially Observable Markov Decision process

Table 2.1: This table shows the typical Markov models used depending on the situation [3].

- **Markov chain:** it is the simplest Markov model. It represents the state of a system through continuous or discrete time. All the states of the model are fully observable by the observer. We describe the Markov chain in Section 2.3.1.
- **Hidden Markov model:** it is a Markov chain for which the states are partially observable. An observation is related to the state of the system, but it is typically insufficient to precisely determine the state [2]. We describe the Hidden Markov model in Section 2.3.2.
- **Markov Decision process:** it is a discrete time stochastic control process. It provides the mathematical framework to model decision making situations where the outcome is partially random and partially decided by an agent. At each time step, the decision maker may choose an action (a) based on the available set of actions (A) and the current state x. The model responds by randomly changing the state into s' and giving reward $R(s, s')$ [22].
- **Partially Observable Markov Decision process:** it is a generalisation of a Markov Decision process in which an agent cannot directly observe the system. An agent chooses the best action depending on the maximisation of the expected reward over a possible infinite horizon. [23]

In our problem, it is impossible to identify an agent that takes actions and affect the evolution of the system. So, we focus our analysis on Markov chain model and Hidden Markov model.

2.3.1 Markov chain model

A stochastic process is a family of random variables, namely $\{X_n\}$ where $n = 1, 2, 3....$ The value x_n of the random variable X_n at the time n is called the *state* of the random variable at that time instant. The set of all possible values that the random variable X_n can assume is called *state space*, namely S.

When the dependence of x_{n+1} is entirely captured by the dependence on the last sample x_n we say that the stochastic process is a *Markov chain*. More precisely, we have:

$$P(X_{n+1} = x_{n+1} | X_1 = x_1, ..., X_n = x_n) = P(X_{n+1} = x_{n+1} | X_n = x_n) \quad (2.1)$$

$where: \quad x_{n+1}, x_n, ..., x_1 \in S$
Which is called *Markov Property.* We provide a definition of this property.

Definition 1 (Markov Property [24]:) *A sequence of random variables* $X_1, X_2, ..., X_n, X_{n+1}$ *forms a Markov chain, if the probability that the system is in state* x_{n+1}*, given the sequence of past states it has gone through, is exclusively determined by the state of the system at time* n*.*

We can think of the Markov chain as a *generative model*, consisting of a number of states linked by transitions. Each time a state is visited, the model outputs the symbol associated with that state.

The process is also characterised by a *transition probability* defined over every combination of states. Let

$$P(X_{n+1} = j|X_n = i) = p_{i,j} \tag{2.2}$$

denote the transition probability from the state i to the state j. If the transition probability between two states is fixed and does not change with time, the Markov chain is said to be *homogeneous.*

Definition 2 (Homogeneous Markov chain [17]) *An homogeneous Markov chain on a finite or countable set* S *is a family of random variables* $X_0, X_1, ..., X_n$ *such that:*

$$P(X_{n+1} = j|X_n = i) = p_{i,j} \tag{2.3}$$

The distribution of the Markov chain is uniquely determined by the initial distribution and the transition probabilities

$$\phi(i) = P(X_0 = i) \qquad \textit{Initial distribution} \tag{2.4a}$$
$$P_{i,j} = P(X_t = j|X_{t-1} = i) \qquad \textit{Transition probability} \tag{2.4b}$$

Definition 3 (Initial distribution [17]) *For each* $i \in S$ *let* $\phi(i)$ *be the probability of the system to be in state* i *at time* $n = 0$ *where we assume that:*

$$\phi(i) \in [0, 1] \tag{2.5a}$$
$$\sum_{i \in S} \phi(i) = 1 \tag{2.5b}$$

In order to be valid, all the transition probabilities of a Markov chain must satisfy the following properties:

$$\sum_{j \in S} p_{i,j} = 1 \quad \forall i \in S \tag{2.6a}$$

$$p_{i,j} \geq 0 \quad \forall i, j \in S \tag{2.6b}$$

In case of a system with a finite number of states, the transition can be compactly represented by using the *transition matrix* P. The transition matrix of a system composed by K states is:

$$P = \begin{bmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,K} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,K} \\ \vdots & \vdots & \ddots & \vdots \\ p_{K,1} & p_{K,2} & \cdots & p_{K,K} \end{bmatrix} \tag{2.7}$$

A condition on P is that each row must add to unity.

The definition of transition probability given in Equation 2.3 may be generalised to cases where the transition from i to j take place in a fixed number of steps. Let m be the number of steps and denote $p^m_{i,j}$ the *m-step transition probability.*

$$P(X_{m+n} = j | X_n = i) = p^m{}_{i,j} \tag{2.8}$$

$p^m{}_{i,j}$ may be seen as the sum over all intermediate states, k, through which the system passes in its transition from i to j.

$$p^{m+1}{}_{i,j} = \sum_k p^m{}_{i,k} p_{k,j} \tag{2.9a}$$

$$p^1{}_{i,k} = p_{i,k} \tag{2.9b}$$

A directed graph gives the dynamics of a Markov chain with a state space S and transition matrix P with nodes representing the individual states and the edge labelled by the probability of possible transition. In Figure 2.1, a simple example of a Markov chain is shown using a directed graph. According to the figure, a bull week is followed by another bull week the 90% of the time, a bear week 7.5% of the time and a stagnant week by another 2.5%. Labelling the state $\{1 - \textit{bull}, 2 - \textit{bear}, 3 - \textit{stagnant}\}$ the transition matrix is:

$$P = \begin{bmatrix} 0.9 & 0.075 & 0.025 \\ 0.15 & 0.8 & 0.05 \\ 0.25 & 0.25 & 0.5 \end{bmatrix} \tag{2.10}$$

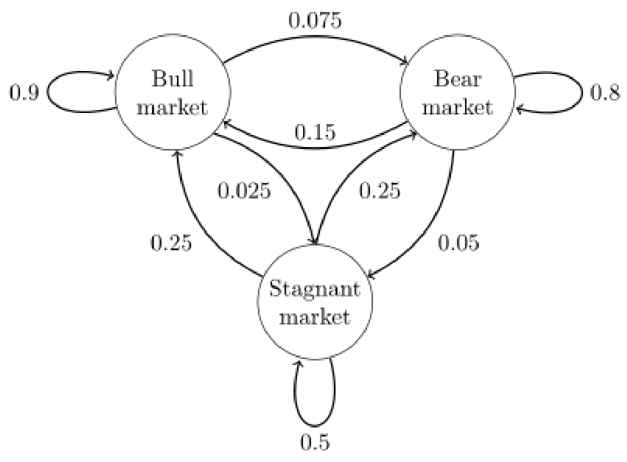


Figure 2.1: An example of Markov chain with three states, taken from [1].

Now, we consider the problem of determining the probability of a sequence of states. Given a sequence of states $s_0, s_1, ..., s_n \in S$ where n is the length of the sequence, and $p_{i,j}$ the probability reach the state s_j given the state s_i. The probability of the entire sequence is computed as:

$$P(s_0, s_1, ..., s_n \in S) = \prod_{i=1}^{n} p_{i\text{-}1,i} \tag{2.11}$$

This allows the model to be applied to sequences of arbitrary length. By considering the example introduced by the Figure 2.1, we can compute the probability of the sequence $(Bull, Bull, Bear, Stagnant)$, respectively (s_1, s_1, s_2, s_3) by using 2.11.

$$P(s_1, s_1, s_2, s_3) = p_{1,1} * p_{1,2} * p_{2,3} \tag{2.12a}$$
$$= 0.9 * 0.9 * 0.075 * 0.05 = 0.003 \tag{2.12b}$$

The most known examples of Markov chains are:

- Random walk: it is a stochastic random process, that describes a path that consists of a succession of random steps on some space such as the integers. An elementary example can start from 0 and move +1 or -1 with equal probability. The move depends only on the current position because of the Markov property. The transition probability to reach the next smaller or larger integer are both 0.5.

- The weather example: given the weather at the present moment, a Markov chain can predict the weather in the next days. The evolution of the weather can be modelled as a stochastic process.

2.3.1.1 Learning

When the transition probabilities are unknown we can learn them automatically from the historical data of the system. The method implemented in this thesis tries to extract a set of pairs from a set of sequences of states. A pair represents a transition of the final model and it can be written as (s_i, s_j). To find $p_{i,j}$ we have to count the number of times where from the state i we observe a transition to j, and divide it from the total number of pairs that have i as starting point. Let us give a formal definition of the concepts explained above:

$$s_0, s_1, ..., s_n \in S \tag{2.13a}$$
$$Q = (q_0, q_1, q_2, ..., q_T) \quad | \quad q_i \in S \tag{2.13b}$$

Where S is the set of states and Q is the sequence of states observed in a single execution of the system. By observing the system K times, we can obtain a set of different sequences with different length $T_0, T_1, ..., T_K$. All these sequences are collected in D, the set of all the data we have.

$$Q_k = (q_0^k, q_1^k, q_2^k, ..., q_{T_k}^k) \tag{2.14a}$$
$$D = \{Q_0, Q_1, Q_2, ..., Q_K\} \tag{2.14b}$$

The next step consists of exploding the sequences we have in a set of pairs. Every pair (s_i, s_j) represents an association between s_i and s_j coming from the sequences we have. All the pairs generated are inserted in the set F.

$$C_k = \{({q^k}_t, {q^k}_{t+1}) | \forall t \in 0 : T_k \wedge \forall k \in 0 : K\} \tag{2.15a}$$
$$= \{(s_i, s_j) | \forall t \in 0 : T_k \wedge \forall k \in 0 : K, {q^k}_t = s_i \wedge {q^k}_{t+1} = s_j\} \tag{2.15b}$$
$$F = C_0 \cup C_1 \cup ... \cup C_k \tag{2.15c}$$

In the end, the probability to reach the state j by being in the state i can be computed as follows:

$$p_{s_i,s_j} = p_{i,j} = \frac{\|\{(s_i, s_j) | (s_i, s_j) \in F\}\|}{\|\{(s_i, x) | (s_i, x) \in C \wedge x \in S\}\|} \tag{2.16}$$

Where in the numerator we are counting the number of couples having both s_i and s_j and in the denominator we are counting the number of couples that have s_i as first member.

2.3.2 Hidden Markov model

Hidden Markov model (HMM) is a statistical model in which the system modelled is assumed to be a Markov process with unobserved states. An HMM consists of two stochastic processes: a visible process of observable symbols and an invisible process of hidden states. In simpler Markov chain the states provide a single output. They are directly visible to the observer and the transition probabilities are the only parameters. According to [25] and [26], an HMM is defined by the following parameters:

- n**:** number of states of the model. Although the states are hidden, it is practical to determine the number of different states. For many practical applications there is a physical significance of a state. The set of all possible states is denoted by

  $$S = \{S_1, S_2, ..., S_n\}$$

  The state at time t is denoted by S_t.

- m**:** number of different observation symbols. In HMM the different observations are no more equal to the states of the model. An observation corresponds to the physical output of the system being modelled. We denote a symbol as v_i and the all set of symbols as

  $$V = \{v_1, v_2, ..., v_m\}$$

- **Transition probability:** the probability of reaching a state by starting from a defined state is computed as in the classical Markov chain, Equation (2.3). The probability of the hidden variable x_t depends only on the value of the variable x_{t-1} because of the *Markov property.* The transition probability distribution $A = \{a_{ij}\}$ is a stochastic matrix and defines the connection between the states of the model.

- **Observation probability:** the probability distribution in each state $B = \{b_j(k)\}$ where $b_j(k)$ is the probability that the symbol v_k is the output of the state S_j.

- **Initial state distribution:** as in typical Markov chain, it is defined by $\pi = \pi_i 1 < i < n$, where π_i is the probability that the model is in state S_i at $t = 0$.

The following notation is often used in the literature by several author [26]:

$$\lambda = (A, B, \pi) \tag{2.17}$$

The observation probability is defined by the following formula and must satisfy the following attributes:

$$b_j(k) = P(v_k = o_t | s_t = j) \tag{2.18a}$$

$$b_j(k) \geq 0, \quad 1 \leq j \leq n, \qquad 1 \leq k \leq m \tag{2.18b}$$

$$\sum_{k=1}^{m} b_j(k) = 1, \quad 1 \leq j \leq n, \quad 1 \leq k \leq m \tag{2.18c}$$

Where v_k denotes the k^{th} symbol in V and o_t the current observation.

Figure 2.2 shows the generic architecture of an HMM. Each oval shape represents a state of the system we are considering. The random variable $y(t)$ represents an observation at time t and the arrows denote conditional dependencies.

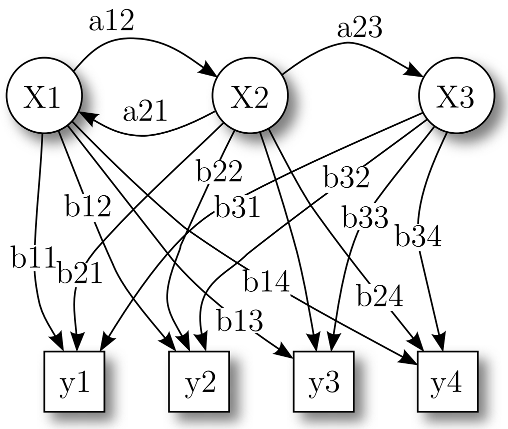


Figure 2.2: The figure shows the parameters of an HMM. X_i is a state, y_i is a possible observation, a_{ij} is the transition probability, and b_{ij} is the observation probability [2].

2.3.2.1 Learning

The learning is the operation of adjusting the HMM parameters to represent a sequence of observations in the best way. Depending on the application there are several optimisation criteria for learning. We introduce the

Forward-Backward variables and the *Baum-Welch algorithm* by following the steps provided by [26]. There is no way to solve analytically the problem of maximising the parameters of the model given a specific sequence of observations. The algorithm finds a local maximum for $P(O/\lambda)$ where O is an observation sequence and λ is the HMM model.

Definition 4 (Forward variable) *$\alpha_t(i)$, called forward variable, is the probability of the partial observation sequence $o_1, o_2, .., o_t$ (until time t) and state s_i at time t.*

$$\alpha_1(i) = \phi_i b_i(o_1) \tag{2.19a}$$

$$\alpha_{t+1} = \sum_{i=1}^{N} \alpha_t(j) a_{ij} b_j(o_{t+1}) \tag{2.19b}$$

$$P(O|\lambda) = \sum_{i=1}^{N} \alpha_T(i) \tag{2.19c}$$

Definition 5 (Backward variable) *The backward variable $\beta_t(i)$ is the probability of the partial observation sequence given the current state i.*

$$\beta_t(i) = \sum_{j=1}^{N} \beta_{t+1}(j) a_{ij} b_j(o_{t+1}) \tag{2.20a}$$

We define the a posteriori probability $\gamma_t(i)$ as the probability of being in state i given the observed sequence O.

$$\gamma_t(i) = P(s_t = i|O, \lambda) \tag{2.21a}$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{P(O|\lambda)} \tag{2.21b}$$

$$\gamma_t(i) = \frac{\alpha_t(i)\beta_t(i)}{\sum_{i=1}^{N} \alpha_t(i)\beta_t(i)} \tag{2.21c}$$

Secondly, we define $\xi_t(i, j)$, the probability to be in state i at time t and state j at time $t + 1$, given the model and the observation sequence.

$$\xi_t(i, j) = p\{s_t = i, s_{t+1} = j|O, \lambda\} \tag{2.22}$$

By using the *forward-backward* variables we can rewrite $\xi_t(i, j)$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})}{P(O|\lambda)} \tag{2.23a}$$

$$\xi_t(i, j) = \frac{\alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})}{\sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_t(i) a_{ij} \beta_{t+1}(j) b_j(o_{t+1})} \tag{2.23b}$$

In fact the previously defined $\gamma_t(i)$ can be related to $\xi_t(i,j)$

$$\gamma_t(i) = \sum_{j=1}^{N} \xi_t(i,j) \tag{2.24}$$

We can now introduce the *re-estimation formulas*, that update the HMM parameters of the model:

$$\bar{\phi}_i = \gamma_t(i) \tag{2.25a}$$

$$\bar{a}_{ij} = \frac{\sum_{t=1}^{T} -1\xi_t(i,j)}{\sum_{t=1}^{T} -1\gamma_t(i)} \tag{2.25b}$$

$$\bar{b}_j(k) = \frac{\sum_{t=1,o_t=v_k}^{T} \gamma_t(j)}{\sum_{t=1}^{T} \gamma_t(i)} \tag{2.25c}$$

Based on the above procedure, we use iteratively $\bar{\lambda}$ to repeat the reestimation until some limiting point is reached. At each iteration, in the model obtained, the probability of observing a specific sequence O is bigger than the probability of the previous iteration.

2.4 Related work

Despite being a quite simple model with respect to other models, Markov chains are used in several interesting applications as a statistical model for real-world systems like speech recognition [27] and hand gesture recognition [28]. The algorithm proposed by Google and known as PageRank is based on a Markov process [1]. Furthermore, Markov chains are used in fault detection, diagnosis and analysis, like in [29] that studies a diagnostic system to detect incipient faults of a three-tank system. The Markov chain model is also used to estimate the failure state probability of permanent magnet AC machines [30], focusing on the prognosis of faults in the presence of limited data. The results describe a new method to compute Markov model parameters using different distributions and an algorithm to determine the next most probable fault.

By using the consensus of a group of agents, [31] develops a fault diagnosis procedure based on Markov chain. In this work a set of agents share information about observations and the most likely parameters of the general system: when the convergence to the consensus is archived, the implementation of the fundamentals of fault analysis can start. A Markov model is also used to model the correlation between faults in a complex electrical infrastructure [6]: the input data are composed of the events of the system. Instead of using every record contained on the input files, the authors found some relevant situations of important devices. Differently from most of the approaches used in the fault analysis, they don't build a nominal model of the system but the model defines the behaviour and the relations of the anomalous events.

Chapter 3

Problem setting

In this chapter, we describe and analyse the central aspects of the problem we have worked on. In Section 3.1, we illustrate the application domain of the system. In Section 3.2, we provide the main definitions used in this thesis. In Section 3.3, we describe the main aspects of the initial data. In Section 3.4, we define the requirements and the goal of the project. In Section 3.5, we formalise the problem setting. In Section 3.6, we define the libraries used to develop the application.

3.1 Application domain

This work is focused on systems composed by different Uninterruptible Power Supply (UPS). The main goal of those systems is to provide uninterrupted, reliable and high-quality electric power for vital loads. In facts, they protect sensitive loads against power outages, overvoltage, or undervoltage conditions. Applications of UPS systems include medical facilities, server farms and computer systems, industrial processing, and telecommunications [32].

A UPS is a complex system that contains many sub-components interconnected. A single UPS can be connected, in parallel or in series, to other UPS, with the purpose of creating a redundancy of the system. In case of a fault on a single UPS, the other part of the system can continue to work. During operation, a UPS can issue messages or alarms that are stored in the log file of the system.

Up to now, given an alarm, a set of possible inspection and maintenance actions is defined in the service manual of the system, but there is no information on which one to prioritise first. However, it is likely that a failure mode is more common than others. By considering the log file and the service manual provided by the company that collaborated to this thesis, we characterise the probability of a failure mode with respect to another. A user will be able to prioritise different inspection and maintenance actions, so the result could be an overall improvement of the system time to repair.

3.2 Main definitions

These are the main definitions of the elements used in this document.

- **System:** It is composed of a set of UPS that are connected together.
- **Event log file:** The file that contains all the data generated by the system. The company that follow this thesis provided this file to us.
- **Device:** It is a single UPS. A device can generate events that are stored in the event log file.
- **Sub-component:** It is composed by the set of the internal components of a UPS. Each sub-component is connected to others and each sub-component has an ID code in order to identify it in case of fails.
- **Event:** It is a record in the event log file generated by the device. It can be a message or an alarm.
- **Message:** It is a non-critical fault and it is not associated with a maintenance strategy.
- **Alarm:** It is a hard fault. If an alarm occurs the device is blocked. The customer service is needed to repair it.
- **Customer service:** It is a separate operating unit of the company with the task of repairing the UPS systems of the customers.
- **Maintenance strategy:** It is a set of maintenance actions performed by the customer service in order to repair a device after an alarm. Usually after an alarm there are a lot of sub-components that can be broken. For each alarm, the maintenance strategy is a list of sub-components to change or check.

- **Maintenance action:** It is a single action that the customer service can take in order to repair a UPS in case of fails.

- **Service log file:** The file that contains all the maintenance actions performed on the system by the service. Every maintenance action is associated with an alarm. The company did not provide this file to us, so we hypothesise its existence in Section 5.6.2.

3.3 Data available

Now, we show the structure of both the *event log file* and the *UPScale*, which were provided by the company that collaborated to this thesis. The event log file is a table in which columns represent the device that generate an event and rows represent the occurrences of those events. An example of the event log file is reported in Table 3.1, A, B, C, D, E, F are the events that occur during the execution of the system. For instance, the event A is generated by the device 1, the events B and C are generated by the devices 1 and 5 and they occur at the same time, namely 2.

	Devices			
Time	01	02	05	06
1	A			
2	B		C	
3		D		
4		E		F

Table 3.1: This table shows how the event log file is structured.

When two events occur in the same row but with different columns it means that they are happening at the same time but from different devices. The time column represents the succession of the events without giving us information about the amount of time between these events. By considering the example introduced before, the events A and B may happen approximately at the same time, in an hour, or in different days. In the event log file, every event is written as a plain string that contains a set of attributes. Table 3.2 shows how the event A of the example could appear.

01: 09.05.17 14:18:38 c=1202 s=miL- MAINS RECT. FAULT

Table 3.2: This table shows how an UPS reports an event.

We can already identify a set of information related to each event:

- **Device Number:** reported at the beginning of each string that is the same number of the column where the event appears.
- **Date:** represents the instant of time at which an event is written in the log file. That different with respect to the instant of time at which the event is generated, we describe in Section 5.2.3 how resolve this potential problem.
- **Event information:** represents a description of the event that has been generated. In the event log file, an event is described by both a code and a text description. The code is usually number between 1000 and 9999, in some cases it is composed by a letter followed by 3 numbers, for instance $A101$. The text description is composed of few words, for instance *MAINS RCT CTRL ERROR*.

The second file provided by the company and used in our analysis is the UP-Scale. When a failure occurs, this file contains all the information regarding that failure mode. The customer service uses UPScale as the reference guide to maintain and repair a UPS. The main structure of this file is reported in Table 3.3 where we show the events with code 1201 and 1203.

Code	Display and Newset Test	Mesg.	Alr.	UPS	Upscale
1201	MAINS RCT CTRL ERROR		X	Yes	A software control alarm, Action: Replacement of the control board XXXXX is probably required.
1203	MAINS RECTIFIER OK	X		Yes	Rectifier Input mains is available and within tolerances.

Table 3.3: This table shows the information contained the UPScale.

The first two columns are used to identify an event by including both a code and a text description. Subsequently, there are Boolean information about the event type (message or alarm), an event cannot belong to both the categories. In the end, there is a natural language description on what to do to repair the UPS in case of fail. As we see from the data, when an alarm occurs, there is an action or a set of actions to take in order to repair the device. On the other hand, when a message occurs it is reported the reason of that message without any maintenance strategy. The event type is a crucial information for our model because it is the only way we have to separate the non-crucial fails and critical faults.

3.4 Goals and requirements

As already illustrated, UPS systems are fundamental to many business companies, in order to avoid that if an unexpected power disruption causes injuries, fatalities, data loss, or serious business disruption. Usually, in the contracts between the UPS providers and clients, there may be fines when the UPS system violates its correct behaviour. In order to know the failure mode that could be reached by a UPS, a model capable of predicting its future evolution is needed. Thanks to this prediction it is possible to improve the quality and the dependability of the UPS system in terms of time to repair and costs from both provider and consumer sides.

A model that relates messages, alarms and maintenance strategies could be used by the provider to understand on which sub-component the fails occurs. Furthermore, given the probability of every change of state, it is possible to inform the use of different maintenance strategies, by dynamically providing the most probable sub-component to change. The model can also be used to evaluate the maximum utility of a replace versus a repair strategy. A model capable of predicting the behaviour of the faults could also be used by the sales support in order to demonstrate how that UPS system can perform well with respect to competitors. Thus, an application able to compute and display the models is required.

Obviously, since the application will be used by different sectors inside a company (technicians, engineers, customer service, maintenance service), an easy to use user interface is needed to set the parameters. Moreover, the resulting model should be shown to the user easily. Another fundamental requirement of this kind of application regards the integration with existing processes and the embedding of domain knowledge and company-specific information.

3.5 Formalising the problem setting

Before creating a model, we must understand how to pre-process the data provided, in order to be efficiently read by a computer. Moreover, we must analyse their distribution over different factors. We conduct this analysis by considering the day, the device, and the order of the events in the event log file. A complete description of the work done on the data is reported in Chapter 4. After this analysis, the model we decided to build relates the information provided in the event log file and in the UPScale file. Starting from the event log file, we create a series of chains that associate a message with the following until an alarm occurs. Based on the UPScale file we also make an association between an alarm and a set of maintenance strategies.

3.5.1 Model selection

We choose to develop a model of the system by considering Markov chains and hidden Markov models. We have already highlighted their points of strength in Section 2.3 but, in our problem, the data provided by the company are not enough to compute a real probability of the events. Furthermore, the *service log file* is missing and we cannot give a real probability on the association of a maintenance action with an alarm. We will assume in Section 5.6.2 how the service log file could be used.

In general, while it is obvious that a complex model describes better a system, a simple model is easier to understand, build, and maintain. In these cases the best solution consists of picking up the easiest model that is "good enough" to describe the system. We decide to study and implement a Markov chain model and consider that the states are fully observable. The main reason is the possibility to understand better the behaviour of the Markov chain model with respect to the hidden Markov model. We describe at the end of the Chapter 5 the hidden Markov models may be used to create a model with better prediction capability.

3.5.2 message/alarm association

To be as general as possible, the model can be built by taking into account four parameters. The first is the device number: we can choose to create the model by considering only a subset of devices. This allows us to consider the specific set of states coming from that specific set of devices, this parameter is useful in case we want to model only a portion of the entire system. The second parameter that can be considered for building the model is the interval of time of interest. This parameter selects a subset of events, from the entire set, from which we will build the model. This parameter allows us to extract and analyse the behaviour of the system during a particular time interval. In fact, the first two parameters are used to obtain a subset of events from the event log file to train the model.

The following two parameters of the model are used to define how the events are associated with each other. The third parameter that is used to build the chains of the model is the delta time (ΔT) between the first event of the chain and the last. In our analysis we build the chains composed of a series of ordered messages ending with an alarm, in Section 5.2 we discuss this choice. By considering Figure 3.1, the messages m_1 and m_2 are not inside the time window ΔT. The sequence generated from this event log file with the given ΔT contains three messages and one alarm: $Q = m_3, m_4, m_5, a_1$.

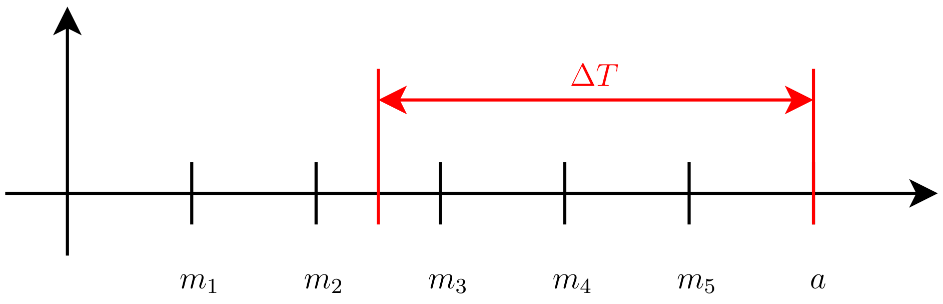


Figure 3.1: The figure shows how the ΔT works in the event log file.

To define the sequences, the fourth parameter that we consider is the *chain length (N)*, defined as the number of messages inside the time window. This parameter is needed because it may happen that inside a ΔT we can have too many messages and it is reasonable to assume that not all these messages are associated with that alarm. So, we define N that is the maximum number of the messages to be considered in a sequence before an alarm. As shown in Figure 3.2 with $N = 3$ it is possible to say that m_1 and m_2 are not associated with a because there already are at least N messages between them and a. The result of these operations creates a sequence of events that

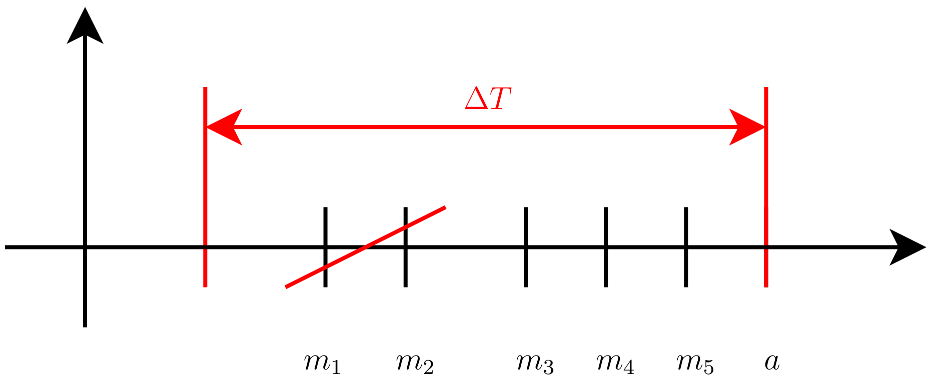


Figure 3.2: The figure shows how the chain length parameter (N) works when a sequence is too long.

ends with an alarm. The formal definition of a sequence is reported in the formulas below:

$$Q = (q_1, q_2, ..., q_N, a) \tag{3.1a}$$
$$where \quad a \in A \wedge q_{1...N} \in M \tag{3.1b}$$
$$T(a) - T(q_1) \leq \Delta T \tag{3.1c}$$

A is the set of alarms, M is the set of messages, and $T(k)$ represent the instant of time of the event k. A detailed description of how these parameters are used is shown in Chapter 5.

3.5.3 Alarm/maintenance actions association

Based on the UPScale file we can add more information to each sequence. An alarm a can be associated with a set of different maintenance actions f_1, f_2. It means that for each alarm we have a set of possible actions to do on the UPS in order to restore it. These associations are extracted from the UPScale and appended at the end of the chains defined in the previous Section. Let us introduce the example in Figure 3.3 where we have a_1 associated with f_1 and f_2 and a_2 associated with f_2 and f_3.

$$\begin{tikzcd}
 & a_1 \arrow[dl] \arrow[dr] & & a_2 \arrow[dl] \arrow[dr] & \\
f_1 & & f_2 & & f_3
\end{tikzcd}$$

Figure 3.3: The figure shows how the associations between alarms a_1, a_2 and maintenance actions f_1, f_2, f_3 look like.

In order to be as accurate as possible, there are two ways to verify these associations. In the first case, every association should be checked by technical experts, the result will be composed of a static result of which are the real associations. In the second case, the associations could be extracted automatically from historical data, the result will be dynamic and capable of capturing the hidden dependencies between alarms and maintenance actions unknown by the technical experts.

The last element we introduce in the chains is an initial state *(init)* representing a condition in which we have not received messages or alarms yet. After this analysis, the series of complete chains are composed by four different elements. The first is the *init* state, after that it continues with the sequence discovered in the previous process $Q = (m_1, m_2, m_3, a_1)$. At the end of the chain, we find the maintenance strategy (f_1, f_2, f_3) to repair the UPS that generates that alarm. In general, the probability to reach an alarm state can be higher or lower than that of reaching another alarm state. This probability depends on the data contained in the event log file. The example below shows how a chain is represented in an informal graph and with formal equations:

$$\begin{tikzcd}
\text{init} \arrow[r] & \mathrm{q}_1 \arrow[r] & \mathrm{q}_2 \arrow[r] & \mathrm{q}_\mathrm{N} \arrow[r] & \text{a} \arrow[r] & \text{f}
\end{tikzcd}$$

Figure 3.4: By using the graph representation, the figure shows an example of a chain.

$$R = (init, Q, f_j) \tag{3.2a}$$

$$R = (init, q_1, q_2, q_3, ..., q_N, a, f_j) \tag{3.2b}$$

$$a \in A \quad \land \tag{3.2c}$$
$$q_1, q_2, ..., q_N \in M \quad \land \tag{3.2d}$$
$$T(a) - T(q_1) \leq \Delta T \land \tag{3.2e}$$
$$f \in F \tag{3.2f}$$

In Equations 3.2 we formally define a chain. F is the set of maintenance actions. Obviously starting from the initial state, the chains can share some messages, alarms, or maintenance actions. We can identify a set of messages that can occur in the first place, others that occur in the second and so on. Instead of representing the model by using separate chains we can use a graph. For instance, we can express the chains below by using the chart in Figure 3.5:

$$R_1 = (init, m_1, m_3, m_4, a_1, f_1) \tag{3.3a}$$
$$R_2 = (init, m_2, m_3, m_5, a_1, f_2) \tag{3.3b}$$
$$R_3 = (init, m_1, m_2, m_4, a_2, f_1) \tag{3.3c}$$

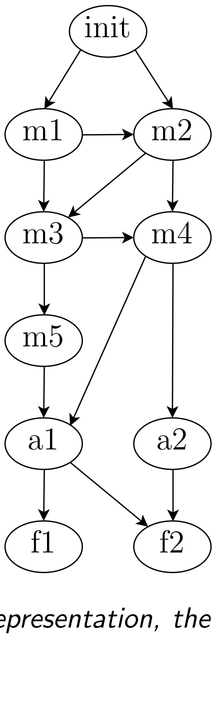


Figure 3.5: By using the graph representation, the figure shows an example of three chains whose have shared states.

3.6 Libraries and software used

We have started the work by using Microsoft Excel to extract the attributes and to extract, from the file provided, all the useful information. We developed our application using Python because of the presence of a large number of standard libraries. Furthermore, Python meets our integration and portability requirements. The application relies on different libraries, the first is called "Pandas" [33] and it offers data structures and operations for manipulating numerical tables and time series efficiently. In our case, Pandas is used to import the files and to handle all the data we have. To show the results we used "Networkx" [34] and "Graphviz" [35], two libraries for studying and creating graphs and networks. The application is easy to use thanks to "Tkinter" [36], the standard Graphical User Interface of Python.

Chapter 4

Preprocessing and data analysis

This chapter illustrates the preliminary steps performed on the provided files in order to evaluate and understand the data we have. In Section 4.1 we extract all the information provided by the files. In Section 4.2 we analyse the information extracted in order to find patterns and distributions of the data. In Section 4.3 we introduce the *dot notation* to address events and attributes. In this work, the preprocessing and data analysis part is essential because we have a small dataset so we must understand thoroughly the information contained in the input files.

4.1 Preprocessing of the data

The first file provided by the company, as mentioned in Section 3.3, is composed of plain strings and it cannot be used to build the models. In the preprocessing phase, we make this file understandable from a computer by extracting as much information as we can and by inserting the information in a new *.xls* file. In this table, we can divide the columns in three logical groups:

- **Id:** it describes the number of the device that generates the event.
- **Date and hour:** they describe the instant of time in which the event occurred.
- **C, S, and error:** they describe the type of event by providing a code and a text description. The S attribute is unusable because, for

each event, it always contains the same value ($miL-$) so it will not be considered in this analysis.

Beyond these fields, in the resulting table that is the input data to build the models, we add the *temp* attribute that describes the order of the events. In the event log file there can be two or more events happening at the same time for two or more different devices. The *temp* attribute aims at keeping track of that by providing the same number when two events are on the same row of the input file. Considering, for instance, Table 3.1, introduced in the previous chapter, the events B and C have the same number in the temp attribute.

For some lines of the input file, the strings are bold and coloured in red or green. To keep track also of this information we add two further attributes:

- **Colour:** it can be B when the input string is coloured in black, G when green, and R when red.
- **Bold:** this attribute can assume the values B when the string is written in bold or N when it is not.

The next step of the preprocessing consists of converting the dates and hours to a standard format in order to make the computation of the time intervals easy. We have also to provide some considerations about the C (code) attribute because different codes may represent the same event in the system. In these specific situations we use the same event code. There are also some codes containing letters instead of only numbers, and we convert them to a number. By considering the Table 3.2 as input file, the result of the preprocessing phase is reported in the Table 4.1.

temp	id	timestamp	date	hour	c	error	colour	bold
1	1	51518	09/05/ 2017	14:18:38	1202	MAINS RECT. FAULT	B	N

Table 4.1: This table shows the information extracted from the event log file of Table 3.2.

From the second file, called UPScale and defined in Section 3.3, we manually extract information about the event type and the maintenance actions needed to repair a device. The result of this extraction consists of two separate files. The first makes an association between event code and its type. This association is merged in the event log file in order to have a unique source of data. Starting from the Table 3.3 of the Section 3.3, the resulting association between event code and type is shown in the Table 4.2.

code	Message	Alarm
1201	0	1
1202	0	1
1203	1	0

Table 4.2: This table shows the relation between event code and event type.

The second file relates codes and the maintenance actions by manually extracting the useful information of the Upscale attribute, defined in Section 3.3. Obviously, for each event code we can have more than one maintenance action. For example, according to the file, the alarm 1202 can be resolved by replacing the Main control board XXX (Table 4.3).

code	UPScale
1202	ReplaceXXXX

Table 4.3: This table shows the relation between event code and maintenance actions.

4.2 Data analysis

After extracting all the data from the sources, in this section, we analyse them in terms of number of records and distribution over devices, dates, and time intervals. In Section 4.2.1 we analyse the number of events over days and devices. In Section 4.2.2 we compare the events of the event log file and the events of the UPScale file. In Section 4.2.3 we analyse the pattern of the event log file. In Section 4.2.4 we analyse the time between events.

4.2.1 Number of records in the event log file

The event log file is composed of 390 events recorded by four different UPSs. We have 297 messages and 93 alarms that respectively correspond to 77% and

23% of the entire dataset. The number of events for each device is reported in the Table 4.4. As we can see they are well distributed and each device has between 95 and 99 events.

Device	Number of events
1	99
2	95
5	97
6	99

Table 4.4: This table shows the number of events for each device.

The events are also equally distributed in terms of messages and alarms for each device. As we can see from Table 4.5, we have at least 70 messages and 18 alarms for each device.

Device	1	2	5	6	sum
Messages	71	70	79	77	297
Alarms	28	25	18	22	93

Table 4.5: This table shows the number of events by type for each device.

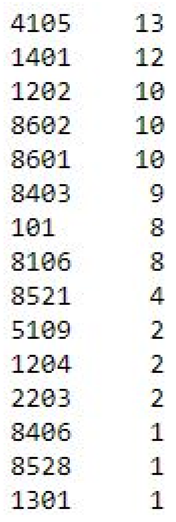
Another analysis we can do on the dataset regards the distribution of events among days. In this case, we have three different days and some differences in terms of number of events for each day. As shown by the Table 4.6, we have the 89% of the events on the 10/05, the 9% on the 11/05, and the 2% on the 9/05.

Date	9/05	10/05	11/05
Messages	3	274	20
Alarms	4	76	13

Table 4.6: This table shows the number of events by type for each day.

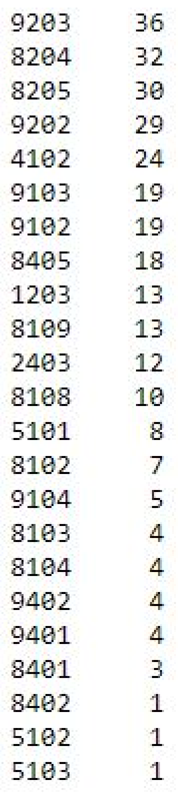
4.2.2 Distribution events log file vs. UPScale

Now, we analyse the distribution of events in terms of differences between the event log file and the UPScale file. By inspecting the UPScale file, a device



4105	13
1401	12
1202	10
8602	10
8601	10
8403	9
101	8
8106	8
8521	4
5109	2
1204	2
2203	2
8406	1
8528	1
1301	1

Figure 4.1: This figure shows the number of times in which every alarm appears in the event log file.



9203	36
8204	32
8205	30
9202	29
4102	24
9103	19
9102	19
8405	18
1203	13
8109	13
2403	12
8108	10
5101	8
8102	7
9104	5
8103	4
8104	4
9402	4
9401	4
8401	3
8402	1
5102	1
5103	1

Figure 4.2: This figure shows the number of times in which every message appears in the event log file.

can arise 218 different events. In this set of different events, we have found 155 alarms and 63 messages. In the other and, the event log file is composed of 38 different events, 23 of them are messages, and 15 alarms. Figure 4.1 shows the distribution of alarms and Figure 4.2 represents the distribution of messages. Some events, reported in the UPScale file, are not contained in the event log file. It is trivial to exclude, in the construction of the final model, those events because it is impossible to associate them with occurrence of the event log file. We realize that in real systems may occur transitions that are not specified in the event log file but we focus our attention only on the subset of events contained in the event log file.

Moreover, the length of the used dataset is not enough to capture all the transitions and to correctly define how is likely that a transition occurs. We model only the transition contained in the data. This means that, the process to generate a model is precise and the results obtained are correct but they are based on the data of that particular system without the possibility to extend the generated model on other systems which could be possible if more data were available.

In fact, we assume the statistical significance of the data provided, we know that the data are not enough but the model can be easily extended on other systems by taking into account a huge dataset of events. By using the big data to build the model, we can capture all the events, define all the transition and assign a true probability to each transition.

4.2.3 Event log file detailed analysis

We have few information from the company about how the system generates messages and alarms. So, a detailed analysis of the event log file is required to understand how the system behaves. Looking at the event log file we can discover some recurrent patterns of the events. For instance, considering the messages 8205, 9202, and 9203 that appear respectively 30, 29, and 36 times, we found the following correlations:

- The message 8205 is followed by the message 9202 in 8 cases.
- The message 8205 is followed by the message 9203 in 16 cases.
- The message 8205 is followed by other messages in 6 cases.

So, we can start to build some straightforward transitions between events and assign some probabilities to these transitions. Figure 4.3 shows the results.

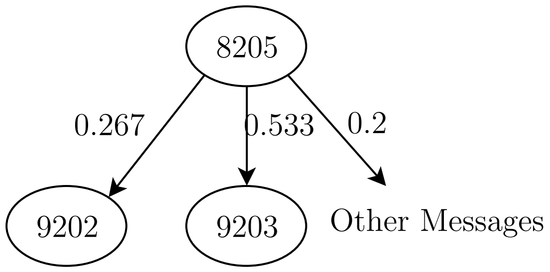


Figure 4.3: The figure shows a simple graph representing the correlations between the events 8205, 9202, *and* 9203.

Some considerations take into account the behaviour of the system when an alarm occurs. We can assert that an alarm may be related to the set of events occurred before it, but in particular the alarm is associated with the last message before it. The assumption is reasonable because the mean of a message is a changing of the state of the system, this alteration of the initial condition may cause an alarm in the future, and the last message is the one that causes the alarm. To support this assertion, we can analyse how the system behaves considering the alarm 4105. From the data we have, considering an interval of 240 seconds between the event 4105 and the messages before, we found the following correlations:

- The alarm 4105 is preceded by the message 4102 in 4 cases, by the message 8405 in 1 case.
- By changing the time interval the occurrences (and the corresponding frequencies) could be different. In the next section we describe the distribution of the time between events (TBE) in order to find the best value.

The above considerations could be represented by the graph in Figure 4.4 where, by giving an alarm a, we calculate the probability to have the message m immediately before. This probability can be expressed as $P(m/a)$. In the image we have the alarm 4105 and the messages 8405 and 4102.

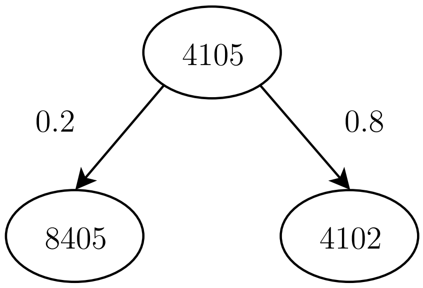


Figure 4.4: The figure shows the correlation between an alarm (4105) and the messages immediately before (8405,4102).

By considering the notation introduced before, the probabilities are the following:

$$P(8405/4105) = 0.2 \tag{4.1a}$$

$$P(4102/4105) = 0.8 \tag{4.1b}$$

The graph introduced in Figure 4.4 creates associations from an alarm that happens at time t with a set of messages happening immediately before. A good improvement is to follow the time flow of the event log file and create an association between a message m and the set of alarms that can be reached after. In Figure 4.5 we calculate the probability to reach an alarm a by having the message m. The probability can be expressed as $P(a/m)$. In fact, giving the message m, we calculate the probability of having an alarm immediately afterwards.

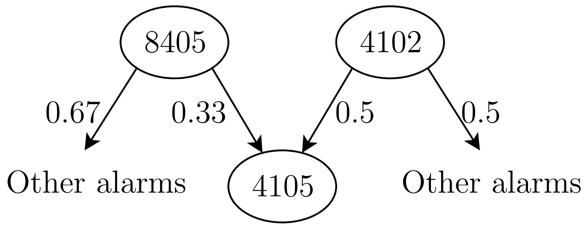


*Figure 4.5: The figure shows the relations between a set of messages (*8405,4102*) that are associated with the same alarm (*4105*).*

Considering the notation introduced before, the probabilities are the following.

$$P(4105/8405) = 0.33 \tag{4.2a}$$

$$P(4105/412) = 0.5 \tag{4.2b}$$

In Chapter 5 we exploit this correlation between messages and alarms to create a model able to express all the transitions of a system before reaching an alarm.

4.2.4 Time between events analysis

As introduced before, events are associated to each other. We have now to analyse if two subsequent events are always associated or there exists some cases in which they are not related. We base our analysis on the Time Between Events (TBE) that is the difference between the time of an event and the time of an event immediately before. By considering the ΔT and N introduced in Section 3.5.2, we find the following correlation:

$$\Delta T \approx E[TBE] \times N \tag{4.3}$$

The ΔT of a chain should be approximately the mean TBE of the system times the chain length.

For instance, considering the message 4102 happening on the 10/05 at 14 : 52 : 38, the following event is the message 9203 happening on the 11/05 at 11 : 16 : 10 that is 21 hours later. It is intuitive to say that in this case the two events are not associated each other. On the other hand, we can consider the event 8601 happening the 10/05 at 10 : 24 : 49 and the following event that is the message 2403 happening 1 second later. It is reasonable to think that these two events could be associated each other. In this analysis we discover relationships among time, messages, and alarms.

The preliminary step considers the overall system, namely all the devices together. This analysis studies the TBE between an event and its successor. By considering the mean TBE, the standard deviation, the minimum, and the maximum, we have a general point of view of the distribution of the events over time. Based on the day attribute, Table 4.7 analyses the TBE. This separation is needed because the TBE between events happening on two different days is quite high and can falsify the results. All the following results are expressed in seconds. As we can see from Table 4.7, the minimum

Day	Mean	Std	Min	Max
9/05	262	442	0	1096
10/05	46	325	0	5045
11/05	1	3	0	19

Table 4.7: This table shows the mean, the standard deviation, the minimum, and the maximum of the TBE for each day.

is always 0, this means that for each day we have at least two events that occurs at the same time. We can see also that the mean time between events is quite high the first day (262 seconds) and very low the third day (1 seconds). The standard deviation is quite high and it is at least two times the mean; probably, the reason of this value is that a series of events are generated, in almost the same time, when a fault occurs. After that, the system continues some time without any event and, when a new fault occurs, a new series of events are generated and written in the event log file. When an event occurs, the following one occurs within a delta time of 2 minutes (120 seconds) in the 95% of the cases.

We can perform a more specialized analysis by considering the behaviour of a single device instead of all the system. As shown by Table 4.8, the minimum and the maximum are the same as before but the mean time between two events on the second day, from 43 seconds, goes to more than 100 seconds. We can say that between two events of the same device there are often events from different devices. To build the model, based on the data we have,

Device	Date	Mean	Std	Min	Max
1	9/05	262	442	0	1096
	10/05	176	640	0	5047
	11/05	-	-	-	-
2	9/05	-	-	-	-
	10/05	143	617	0	5048
	11/05	3	5	0	20
5	9/05	-	-	-	-
	10/05	139	591	0	5045
	11/05	3	8	0	32
6	9/05	-	-	-	-
	10/05	113	533	0	5046
	11/05	-	-	-	-

Table 4.8: This table shows the mean, the standard deviation, the minimum and the maximum of the TBE for each device, for each day.

a TBE between 150 and 300 seems to be appropriate and we will choose the ΔT by considering this. In the Chapter 6 we will discuss the validity of this decision.

In the end, we can evaluate the TBE in case we have a lot of data, these data have statistical value, and, by using the TBE, we can perform data analysis and data cleaning. First of all, in presence of big data, the standard deviation should become lower and a better estimation of the TBE could be archived. Since in the hypothesis of big data we have more than one pair of the same events, we can compute a TBE for each pair and evaluate its correctness.

For instance, there can be a pair of events that usually occur with a mean time of 3 minutes. If the dataset contains one sequence of these two events with a TBE of a couple of seconds, we can remove this relation because is wrong. In our case, we are modelling the failures of a system, and we are trying to analyse the pattern of the messages in order to predict an alarm, so all the random or wrong currencies should be removed. As analysed in this chapter, we do not have enough data to compute a TBE of every pair of events because most of the times every pair is reported one time in our dataset. Thus every TBE has its own significance and it is impossible to distinguish the fake TBE with respect to the real one. In chapter 7 we analyse the future work

4.3 Notation used to refer events and attributes

We introduce now the *dot notation* that will be used in the following part of the document to refer the events and their attributes. We provide also a set of definitions of what the variables contain.

- S: It is the final dataset used to create the model, it contains the data read from the event log file and modified by the preprocessing, defined in Section 4.1. We identify two attributes of the dataset:
  - S.devices: It contains all the ids of the devices we have inside the dataset.
  - S.dates: It contains all the different dates we have inside the dataset.
- e: It is an event contained in the final dataset (S). An event has the list of attributes reported below:
  - e.timestamp: It contains the timestamp of the event.
  - e.date: It contains the specific date of the event by using the format gg/mm/yyyy.
  - e.time: It contains the specific time of the event by using the format hh:mm:ss.
  - e.code: It contains the numerical identifier of the event.
  - e.id: It contains the numerical identifier of the device that generates that event.
  - e.message: It contains a Boolean flag, 1 if the event is a message, 0 otherwise.
  - e.alarm: It contains a Boolean flag, 1 if the event is an alarm, 0 otherwise.

Chapter 5

Markov chain model

In this chapter we describe the steps to build the Markov chain model that represents the system. In Section 5.1 we define how to select the data in order to have a specific view of the system. In Section 5.2 we define the methodologies for building the Markov chains. In Section 5.3 we describe the different representations of the states of the model. In Section 5.4 we resolve the problem of identifying a chain inside the final model. In Section 5.5 we define the Transition matrix how to label the transitions. In Section 5.6 we describe the postprocessing operations performed on the models created.

5.1 Data selection

In this section, we describe the meaning and the behaviour of the parameters used to select a subset of events from the entire dataset. S contains all the information and, as explained in Section 4.3, every event has a set of attributes that are indexed by using the *dot notation*. For instance, the *date* attribute of the event e_1 is written as $e_1.date$. Moreover, we have defined a set of general attributes of the dataset S, for instance, by writing $S.dates$ we receive the different dates contained in the dataset.

We identify two parameters that help the generation of the model. The first considers a subset of the entire set of devices contained in the dataset. By using this parameter, the model focuses the attention only on a particular subset of devices. All the events coming from a device that is not contained in that subset will be excluded from subsequent operations. The formula below reports the subset of devices:

$$DEVICE \subseteq S.devices \tag{5.1}$$

The second parameter is the time interval within the events are considered. The model considers only the events coming from that specific time interval. This parameter is practical when we want to model the behaviour of the system in a specific instant of time. In the case of very exceptional behaviours, we can exclude these events from the general model and construct a specific view considering time interval in which they occur.

$$DAYS \subseteq S.days \tag{5.2}$$

Starting from the dataset S, the subset E contains only the selected elements by considering the two parameters before:

$$E = \{x | x \in S \wedge x.date \in DAYS \wedge x.device \in DEVICES\} \tag{5.3}$$

The set E is further divided into two different sets by considering the event type. The result of this operation is expressed by the equations below:

$$M = \{e_1, e_2, ..., e_k,\} = \{e | e \in E \wedge e.message = 1\} \tag{5.4a}$$
$$A = \{e_1, e_2, ..., e_h,\} = \{e | e \in E \wedge e.alarm = 1\} \tag{5.4b}$$

From now on only the two sets M and A are considered as sources because they contain the data we have selected to build the model. M includes all the messages selected from E and A includes all the alarms selected from E.

5.2 Building the chains

After defining the methodologies for selecting the data, we provide a detailed description of how the events are associated to each other. We have already found the presence of correlations between an event and its successors in Section 4.2.3. Naturally, we cannot build a single chain starting from the first event in the dataset and finishing with the last event, because it represents only the succession of the events without eliciting any useful information. So, we divide the chains based on the alarms. Our choice is reasonable because we assume that after an alarm the system stops its work until the customer service repairs the device. Thus, the alarms provide a sort of stop points for the system and we can assume that an alarm depends on the events occurring before it. When an alarm occurs a maintenance action on the system is required, so we build the chains by associating a sequence of messages with an alarm. We consider every alarm independent to other alarms, however this constraint allows us to predict better the maintenance action required to repair the system.

Let us use an example by considering Table 5.1, we consider the alarms C and E independents and, we associate each alarm with the sequence of messages occurred before. By exploiting this independence when the system reports in the event log file the chain that ends with C, before reaching the alarm C we are able to perform the maintenance actions required to repair the system from the alarm C. This operation cannot be performed if we build the chains by considering more than one alarm because when the system reports in the event log file a sequence of messages we cannot anything about the alarm in the end because they are both related to that chain of messages.

We can, informally, define a chain in two ways. Looking backwards, a chain is the sequence of messages that occur before an alarm. Looking forward, a chain is a sequence of messages ending with an alarm. We provide now a simple example of how the chains are computed. Given Table 5.1 representing a portion of the dataset, we have two different alarms and three different messages. The results consist of two different chains associating an alarm with the messages before it.

Code	Message	Alarm
A	1	0
B	1	0
C	0	1
D	1	0
E	0	1

Table 5.1: This table shows a portion of the dataset used to build the chains.

We report the events by using a letter instead of all the attributes to make the example easier to understand:

$$C_1 = (A, B, C) \tag{5.5a}$$
$$C_2 = (A, B, D, E) \tag{5.5b}$$

By using this method, the chains have an increasing length based on where the alarm is in the event log file. The formula below expresses the formal definition of a chain. We use the *dot notation* defined in the previous chapter:

$$
\begin{aligned}
& C_k = (m_1, m_2, ..., m_{T_k}, a_k) && \text{(5.6a)}\\
\textit{where} \quad & a_k \in A \quad \wedge \quad \forall t : [1 : T_k - 1] \implies m_t \in M \quad \wedge && \text{(5.6b)}\\
& m_{T_k}.\textit{timestamp} \leq a_k.\textit{timestamp} \quad \wedge && \text{(5.6c)}\\
& \forall t : [1 : T_k - 1] \implies m_t.\textit{timestamp} \leq m_{t+1}.\textit{timestamp} && \text{(5.6d)}
\end{aligned}
$$

In this formula we declare that the last element is an alarm, all the elements before it are messages, the messages inside the chain have a timestamp that is smaller or equal to the timestamps of the alarm and of the following messages in the chain.

We discover two problems of building chains with this method. The first is the time interval that occurs between the first message of the chain and the alarm reported at the end. If the dataset reports events spanning different days, by using this method an alarm may be related to a message occurred much time before. The second problem regards the number of the elements of a chain. It may be possible that a chain contains many messages before the alarm because we do not have any constraint on the chain length.

As described in Section 3.5.2, to deal with these problems we introduce two parameters: the first, ΔT, is defined as the maximum time between the first message and the alarm of a chain (by using the Equation 5.7 the first message is m_1 and the alarm is a_k).

The second parameter is the chain length (N). It is intuitive that, considering an alarm, only few messages are strictly associated with that alarm. When, inside ΔT, we still have many messages, this parameter maintains in a chain only the last N messages. We provide now a new, formal definition of a chain:

$$
\begin{aligned}
& C_k = (m_1, m_2, ..., m_{T_k}, a_k) && \text{(5.7a)}\\
\textit{where} \quad & a_k \in A \quad \wedge \quad \forall t : [1 : T_k - 1] \implies m_t \in M \quad \wedge && \text{(5.7b)}\\
& m_{T_k}.\textit{timestamp} \leq a_k.\textit{timestamp} \quad \wedge && \text{(5.7c)}\\
& \forall t : [1 : T_k - 1] \implies m_t.\textit{timestamp} \leq m_{t+1}.\textit{timestamp} && \text{(5.7d)}\\
& T_k < N && \text{(5.7e)}\\
& a_k.\textit{timestamp} - m_1.\textit{timestamp} \leq \Delta T && \text{(5.7f)}
\end{aligned}
$$

Now, by using two examples, we show the behaviour of the two parameters.

5.2.1 Example considering ΔT

Starting with the example introduced in Table 5.1, we add to each event a timestamp in seconds shown in Table 5.2.

Timestamp (s.)	Code	Message	Alarm
105	A	1	0
1000	B	1	0
1002	C	0	1
1004	D	1	0
1006	E	0	1

Table 5.2: This table shows a portion of the dataset including the timestamp.

We also define a ΔT equal to 4 minutes (240 seconds). The resulting chains automatically exclude the message A because it takes place outside the ΔT interval. The resulting chains are:

$$C_1 = (B, C) \tag{5.8a}$$

$$C_2 = (B, D, E) \tag{5.8b}$$

5.2.2 Example considering N

We now consider a dataset composed of many events occurred almost at the same time. The dataset is reported in Table 5.3.

Code	Message	Alarm
A	1	0
B	1	0
C	1	0
D	1	0
E	1	0
F	1	0
G	0	1

Table 5.3: This table shows a portion of the dataset where we have many messages.

We also define the chain length equals to 3, namely $N = 3$. The results automatically exclude the messages A, B, C from the association with alarm G. The generated chain is:

$$C_1 = (D, E, F, G) \tag{5.9a}$$

5.2.3 Dealing with elements coming at the same time

In the event log file it is frequent to have events that appear exactly at the same time. It is a design decision of the system logger. The event logger has the goal to write the events in the event log file, and it usually has a writing granularity of one second. When more than one event arises from the same device, that device stores the set of events in its internal memory and it waits for the event logger that saves the set of events on the event log file. The event logger writes the events without maintaining the order in which they were generated.

We now provide an example by considering the creation of four events.

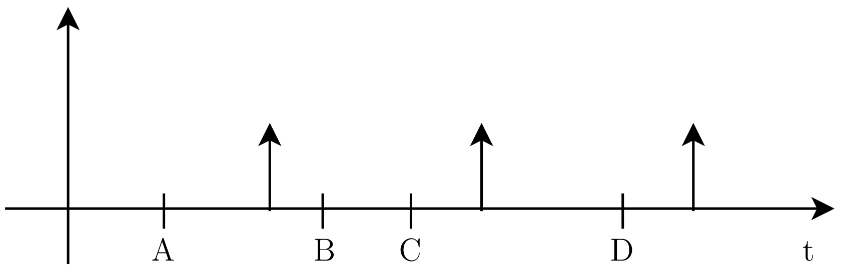


Figure 5.1: The figure shows the writing sequence done by the event logger of the system.

Figure 5.1 shows the creation of events and the writing window of the event logger.

The events B and C are generated inside the same window, so they have the same timestamp on the event log file. There is no way to control which of the two events will be written before on the event log file. The sequences can appear in the event log file in two different ways:

$$(A, B, C, D) \tag{5.10a}$$
$$(A, C, B, D) \tag{5.10b}$$

We identify two different cases that affect the creation of chains. The first case regards the presence of two or more messages at the same time. The second case regards the presence of both alarms and messages generated at the same time. Depending on the case we use different strategies.

When W messages are coming at the same time we evaluate two strategies: the first, the most complete, creates $W!$ chains by expliciting every possible sequence of messages. This solution is practically infeasible because, among the generated set of chains, only one represents the (unknown) correct order of the messages. Moreover, by considering the final model, this solution creates a series of loops that is unpractical. The second solution creates W chains that consider only one message at a time. This solution combines efficiency and completeness because, with respect to the first solution, the loops in the final model are removed. The resulting chains are W even though we are still considering all the messages contained in the event log file. Obviously, among the W chains only one represents what happened to the system correctly but, by using our data, we cannot assert which is the real chain that represents the system. In the idea of working on real big data, that have statistical value, the wrong chains are not a problem because they will result on the tails of the distribution.

Let us introduce an example concerning the presence of three messages at the same time. We consider a dataset reported in Table 5.4.

Timestamp (s)	Code	Message	Alarm
10000	A	1	0
10005	B	1	0
10005	C	1	0
10005	D	1	0
10008	E	1	0
10010	F	0	1

Table 5.4: This table shows a portion of an event log file where three messages are registered at the same time.

We cannot assume the order of the events B, C, D because they have the same timestamp, so we build a chain considering only a single event:

$$(A, B, E, F) \tag{5.11a}$$
$$(A, C, E, F) \tag{5.11b}$$
$$(A, D, E, F) \tag{5.11c}$$

When both messages and alarms come at the same time, we consider the alarm as occurred as last and we consider all the messages as occurred before that alarm. The difference with respect to the strategy reported before comes out because an alarm causes a system block and it is reasonable to assume that after an alarm there cannot be any message because the system is blocked. We introduce an example concerning this case, by considering the dataset in Table 5.5.

Timestamp (s)	Code	Message	Alarm
10000	A	1	0
10003	B	1	0
10005	C	0	1
10005	D	1	0

Table 5.5: This table shows a portion of an event log file where one message and one alarm are coming at the same time.

As shown, the message D comes at the same time of the alarm C. The chains generated consider all the messages that arrive before or at the same time of C, the result is:

$$(A, B, D, C) \tag{5.12a}$$

5.2.4 Initial state

After creating a chain, we need to find a way to represent the operation of the system or the devices before the first message occurs. We add the *initial state* (*init*) at the beginning of each chain. The meaning of the initial state is to represent a condition for the system in which any message or alarm has been generated yet.

5.3 State representations

After having defined the methodologies for building the chains, we analyse how to represent the events of the system in terms of states of the Markov chain model. In the examples introduced the previous section, we used letters (A, B, C) instead of the real events information, now we discuss what should be used to replace those letters. Every event represents a particular condition of the system; therefore, we can use each event of the system as a state of the Markov chains model. For identifying the states, the best solution consists in using the *code* attribute, because, for each event type, it is a unique numerical id easy to manipulate. We have found two ways to represent a state of the Markov chains model: the first includes only the *code* of an event, the second also uses the device number from which the event is coming. Below we report a detailed description of both the representations.

5.3.1 State representation: code

The first state representation includes, for each state, only the event code. It is the minimal set of information needed to identify each state of the system. In Figure 5.2, we show how the final model represents the state 1203.

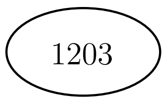


Figure 5.2: The figure shows the event with code 1203 *represented by the code state representation.*

By using this method to represent the states, we leave out the device code that generates the event. This representation provides a general overview of the system without entering into the details on how each device behaves because it considers each message as coming from the system instead from the single device. Let us provide an example of how the chains are built by considering a small dataset composed of three messages and two alarms. The dataset is reported in Table 5.6.

Timestamp (s.)	id	Code	Message	Alarm
128713	1	8401	1	0
128714	2	8401	1	0
128715	2	8102	1	0
128716	1	8403	0	1
128717	1	8204	1	0
128718	2	2203	0	1

Table 5.6: This table shows a portion of the dataset including the timestamp, the device id, and the event code.

Without considering the *init* state, the ΔT, and N, the resulting chains are:

$$C_1 = (8401, 8401, 8102, 8403) \tag{5.13a}$$
$$C_2 = (8401, 8401, 8102, 8204, 2203) \tag{5.13b}$$

As we can see there are two messages with the same code (8401) and coming from different devices (1 and 2). In the final chains they are represented as the same state because in this case the model excludes the information about the device.

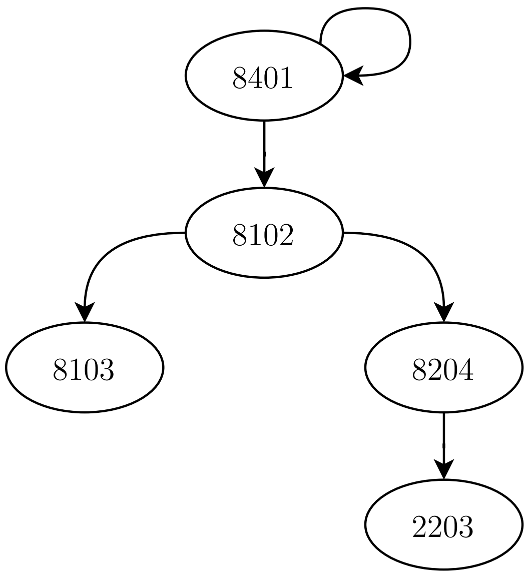


Figure 5.3: The figure shows the graph produced by considering the chains in Equation 5.13.

Let us introduce the graph representation used to understand quickly how the events of the model are related. By considering the chains reported in Equation 5.13, each node of the graph is a state and it is reported by using its state representation. An ark between the events A and B of the graph represents that a chain contains the event A and the successor is the event B. The graph in Figure 5.3 represents how the chains reported in Equation 5.13 are represented.

By considering the figure, the 8401 is associated with an event with the same representation, so we have a loop on the first state. By following the chains, we have the message 8401 associated with message 8102. After the state 8102 the chains follow different paths, thus from this there are two arks that reaches two different states.

5.3.2 State representation: id/code

The second state representation includes, for each state, both the event code and the id of the device that generates the event. In Figure 5.4 we show how the state 1203 coming from the device with id 1 and the state 1203 coming from the device number 2 are represented in the final model.

1/1203
2/1203

Figure 5.4: This figure shows the event 1203 *coming from the devices with id* 1 *and* 2 *appears in the* $id/code$ *representation.*

By using this method, for each state, we provide more information with respect to what introduced in the previous section. The model considers every event of the event log file as written by a specific device instead of the system. By using this representation, we model the behaviour of the single device and the correlations between different devices. So, the system is no longer considered as a *black-box* because the model looks at the single device. Let us give an example of how this method represents the states of the system. By using Table 5.6 introduced in the previous section, the new chains are the following:

$$C_1 = (1/8401, 2/8401, 2/8102, 1/8403) \tag{5.14a}$$

$$C_2 = (1/8401, 2/8401, 2/8102, 1/8204, 2/2203) \tag{5.14b}$$

The message 8401, coming from the devices number 1 and 2, is now represented as two different states of the model. By using the graph representation of the chains, introduced in Section 5.3.2, Figure 5.5 shows how the chains are represented.

By comparing the graph in Figure 5.3 and the one in Figure 5.5, in the second figure, the loop on the state 8401 is removed by using two different states, namely 1/8401 and 2/8401.

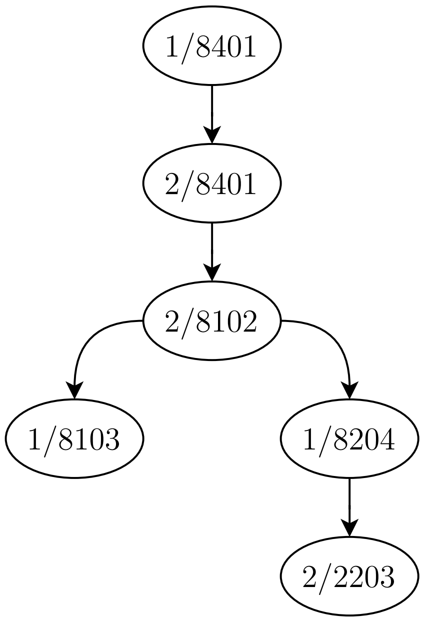


Figure 5.5: This figure shows a graph representing the chains of Equation 5.14.

5.4 Introducing levels

The state representations introduced before provide both a general view of the system and a view of the behaviour of a specific device but, by using these representations, we are not able to have a complete and clear analysis of the sequences of events that happen inside the system. In fact, until now, when we are in a specific state of the graph, we are not able to identify the chain in which this specific state belongs. For instance, by considering the graph in Figure 5.3, when we have the event 8401, we cannot say from which chain (reported in the Equation 5.13) the event arrives. In addition to that, we must model only chains that are reported in the initial dataset without creating, on the graph representation, some paths that do not exist. This problem comes out because of the *Markov property* of the states of the model. We provide an example to understand better the problem by considering the dataset of Table 5.7.

Note that the last four events occur reasonably after the first four. The resulting chains, by considering the ΔT, the *code* representation and including

Timestamp (s)	Code	Message	Alarm
128713	A	1	0
128714	B	1	0
128715	C	1	0
128716	D	0	1
134584	E	1	0
134585	C	1	0
134586	A	1	0
134587	F	0	1

Table 5.7: This table shows a portion of the dataset with 6 messages and 2 alarms.

the *init* state, are reported below.

$$C_1 = (init, A, B, C, D) \tag{5.15a}$$
$$C_2 = (init, E, C, A, F) \tag{5.15b}$$

The resulting graph is shown by Figure 5.6.

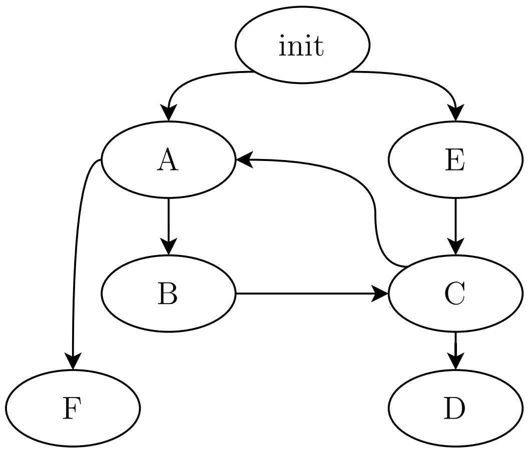


Figure 5.6: This figure shows the resulting graph obtained by considering the chains defined in Equation 5.15.

We notice the presence of two new chains that comes out from the model.

C_3 and C_4, reported below, should not be reachable paths of the model.

$$C_3 = (init, E, C, D) \tag{5.16a}$$
$$C_4 = (init, A, F) \tag{5.16b}$$

We introduce the *level* state representation that includes, for each state, some information about the position of the message inside the chain. In the next section we define what the level is and how the new representation influences the model.

5.4.1 State representation: level**id/code

We develop now a new state representation that also includes the level of the message in the chain when we are building the model. The *level* is described as the depth of the message inside the chain. In addition to the *code* and the *id* used in the previous representation we add the level to each state. Two examples are shown in Figure 5.7 where the event $1**2/1203$ means that at the position 1 of a chain we have an event produced by the device number 2 and with code 1203.

1**2/1203
3**1/1204

*Figure 5.7: This figure shows two states represented by using the Level**id/code representation.* $1**2/1203$ *means that the message with code* 1203 *from Device* 2 *is in the first position of its chain.*

We can consider this representation independent from the representations introduced before. The level representation can be used independently with the code representation or with the id/code representation. In this state representation, the alarms are still reported without introducing the level attribute. This attribute does not provide any useful information for alarms because an alarm is a hard failure that requires some maintenance actions to be resolved independently of its depth in a chain. By considering the chain length (N) described in Section 5.2, we can relate this parameter to the maximum level that a message can have. The level is smaller or equal to N because in a chain we have at most N messages.

The representation introduced before provides different advantages but also some disadvantage. First, as already explained, thanks to this method we remove some of the wrong associations that are not reported in the event log file. Second, the model created by using this method allows us to visualise better the sequences of the messages that end with an alarm. On the other hand, the model will contain more states that report the same code and the same device. This representation is translated into a bigger Transition matrix that can be difficult to analyse. By using the level state representation, it is problematic to understand, given a specific message, which are the messages that follow it, because they are spread out in the model at different levels.

5.5 Transitions

After defining how to represents the states of a model, we provide a clear explanation of how to define and to compute the transitions between states. A transition from the state m_1 to the state m_2 asserts that from the state m_1 the system may reach the state m_2 in the immediate future. As already explained in the previous chapter, from a generic message, we have a set of possible messages that can be reached in one step, thus we have to associate a label to each transition in order define how is likely that the transition can take place. Before calculating the transitions, we have to convert every chain in a set of pairs.

5.5.1 Preprocessing of the chains

Our dataset is now composed of a series of k chains defined by Equation 5.7. In order to efficiently calculate the probabilities of the Transition matrix, we divide the chains in pairs. Every pair is composed of two elements: the first is a message and the second one is the message or the alarm that occur immediately after it in a specific chain. For instance, let us consider the three chains described below:

$$C_1 = (init, A, B, C) \tag{5.17a}$$
$$C_2 = (init, B, D, E) \tag{5.17b}$$
$$C_2 = (init, B, D, C) \tag{5.17c}$$

The pairs generated are the following:

$$\{(init, A); (A, B); (B, C); (init, B); (B, D); (D, E); (init, B); (B, D); (D, C)\} \tag{5.18}$$

By considering the example, we can draw the graph of Figure 5.8 that contains all the information provided by the pairs. Every state is reported once and every pairs represents a transition between the first element and the second.

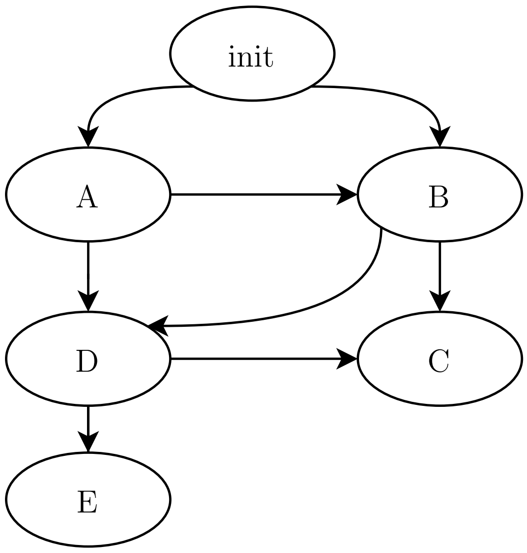


Figure 5.8: The figure shows the pairs reported in Equation 5.18 organized in a graph.

This operation is independent of the representation used to distinguish the states. Every pair is a physical edge of the final graph. Looking at the example introduced before, we can see that the pair (B, D) is reported two times. It means that starting from the message B, it is more likely to get the message D instead of the alarm C. Starting from the pairs we can analyse the size of the transition matrix.

5.5.2 Analysis of the transition matrix

In this section we analyse the size of the transition matrix in order to understand if the data we have allows us to compute a correct probability of the transitions. This analysis can also show the size of the problem we are solving.

The transition matrix of a Markov chain model is a square matrix that has a dimension depending on the number of the states of the model. That number also depends on the representation we choose; therefore, we divide the analysis based on the representation of the states. In addition, for each representation, the branching factor is important because, according to the states and the branching factor, we can estimate the number of edges.

5.5.2.1 Branching factor

We analyse first the branching factor of the graph. The scope of this analysis consists in determining the number of states to which an event is related. Considering the Table 5.8, we found out some interesting results. The maximum is always related to the *init* state because the graph is widely spread out at the first level. The minimum is always 1, this means that, in the final graph, there is always a transition with probability equal to one. The mean decreases with respect to the increasing level of detail of the representation. By considering the code representation, we have a branching factor greater than 7, by considering the level**code/id representation we have a bracing factor below 3.

Since we have more states, in the last representation, it is evident that the branching factor decreases. We also decided to analyse the number of states for each model representation to have a look at the size of the matrix.

Representation	Mean	Std.dev	Minimum	Maximum
Code	7.5	5.6	1	19
id/Code	5.3	5	1	26
Level**Code	3.1	2.6	1	15
Level**id/Code	2.9	2.6	1	26

Table 5.8: This table shows the branching factor of the model with respect to the state representation.

5.5.2.2 Number of states

To calculate the number of states of the final models, we decided to consider different state representations and different model parameters. As shown by Table 5.9 the chain length (N) influences significantly the number of states of the model. The most compact uses 35 states and the most detailed model uses 270 states. In both cases, a computer has no difficulties to process a transition matrix with this size so that we can continue our analysis. Up to now, we discovered that the transition matrix could be computed by a computer in a reasonable time, in Chapter 6 we discuss which model is detailed enough for adequately represent the system.

Repr.	N=5 ΔT=1000	N=10 ΔT=1000	N=5 ΔT=2000
Code	35	36	36
id/Code	88	101	88
Level**id/Code	167	270	170

Table 5.9: This table shows the number of the states with respect to different states representation and model parameters.

5.5.3 Labelling the transitions

As explained by the previous section, every pair represents a transition from the first element of the pair to the second element. Some pairs can be repeated because the first and the second element are adjacent in different chains. On the other hand, some different pairs have the same first element and different second element. We need now to associate a label to each transition in order to represent its importance.

We use two attributes for each transition: the first is the probability that this transition happens, the second is the mean time and standard deviation between the occurrence of the first and of the second element of the pair. We describe now in detail both the attributes.

5.5.3.1 Probability

The probability of each transition is calculated as follows: we already know that we are in the state A, the probability of reaching the state B is the typical conditional probability $P(B/A) = p_{B,A}$. This probability is calculated by dividing the number of pairs that contains A as the first element and B as the second element, by the number of pairs that have A as first element. That probability is written in column A at row B of the transition matrix. Let us consider, for instance, the chains and transition reported in Equation

5.18. The graph generated by the pairs is reported in Figure 5.8. We can compute the probability of every transition:

$$p_{D,B} = \frac{\|(B,D)\|}{\|(B,x)\|} = \frac{2}{3} = 0.67 \tag{5.19a}$$

$$p_{C,B} = \frac{\|(C,B)\|}{\|(B,x)\|} = \frac{1}{3} = 0.33 \tag{5.19b}$$

$$\dots \tag{5.19c}$$

Finally, the B-row of the transition matrix is:

-	init	A	B	C	D	E
B	0	0	0	0.33	0.67	0

Table 5.10: The table shows a row of the transition matrix.

To be corrected, the sum of the elements on each row of the transition matrix should be equal to 1.

Some transition could appear with a very low probability. Some considerations about how to use the probability to evaluate the correctness of a transition is reported in Section 5.6.3.

5.5.3.2 Mean time and standard deviation

Since the Markov chain model provides a way to observe the sequence inside the chains, we include information about the mean time and the corresponding standard deviation between two consecutive states. The mean time provides information about the number of seconds before the event B occurs given the event A. The standard deviation provides information about how much the information returned by the mean time is reliable.

When the model is used to understand the behaviour of a system, this information can be used to predict the occurrence of events dynamically. For instance, we suppose two transition: the first from A to B, with a probability of 90%, a mean time of 5 seconds and standard deviation almost zero, and the second from A to C with a probability of 10%, a mean time of 1 minute and a standard deviation quite high. When we are in the state A and after 5 seconds the event B does not appear in the event log, we can be more confident to reach C and less confident to reach B.

We provide another example regarding the process of removing, from a model, the corrupted or inaccurated transitions. The goal of this operation is to remove the random associations and maintain in a model only the reals one. In the hypothesis to work with big data, the standard deviation for every transition should be low. We can introduce the coefficient of variation (CV) [37], expressed by the formula:

$$CV = \frac{\sigma}{\mu} \tag{5.20}$$

Where σ is the standard deviation and μ is the mean of the interval of time between two subsequent events.

By calculating CV on each transition, it is possible to clear the model from those that does not have a CV highly enough. In fact, by using big data, the transitions that do not respect the CV must be considered as random events or as events that appear without a relationship with the events before.

5.6 Postprocessing

We provide and analyse now a complete example of a simple model generated from the data. The parameters used to build this model are:

- **Considered Date:** 9/05/2017, **Considered Device:** 1.
- **ΔT:** 1500 seconds, **N:** 5.
- **State Representation:** id/Code.

As is shown in Figure 5.9, there are three chains, the first two share the messages but end with different alarms, the third one is entirely separated from the previous. From the *init* state we have two possibilities, the first is to reach 1/1023 with a probability of 0.67. The second is to reach 1/4102 with a probability of 0.33. The mean time variates from 26 seconds (between 1/9103 and 1/1201) to 1096 seconds (between 1/4102 and 1/9103). Between *init* and the first level of messages the time is always zero. It is impossible to know the time between the starting of the system and the occurrence of the message because in the event log file this information is not reported. As shown by Figure 5.9, the *std* is always zero because we have either one occurrence of each couple or more than one occurrence of the same couple with the same TBE.

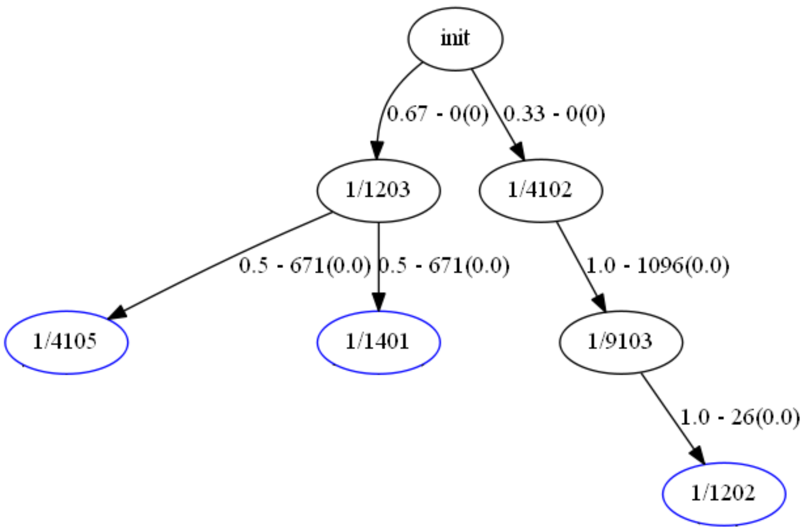


Figure 5.9: The figure shows the model without considering any postprocessing.

5.6.1 Adding maintenance actions

The dataset containing the maintenance actions that can be performed to address each alarm looks like Table 5.11.

code	UPScale
1202	checkVin
8528	ReplaceNW20140
8528	ReplaceNW20171
8521	checkParalConn
...	...

Table 5.11: This table shows the relation between event code and maintenance actions.

It can be seen as an association between alarm code and the corresponding maintenance actions. An alarm has one or more maintenance actions and the maintenance actions can be shared among alarms. For instance, *ReplaceNW20010* is reported 9 times for 9 different alarms and the alarm 8521 can be resolved with 5 different maintenance actions. We can represent these associations by using a simple chain relating every alarm with the set of maintenance actions. Figure 5.10 represents the generated graph of alarm 8521.

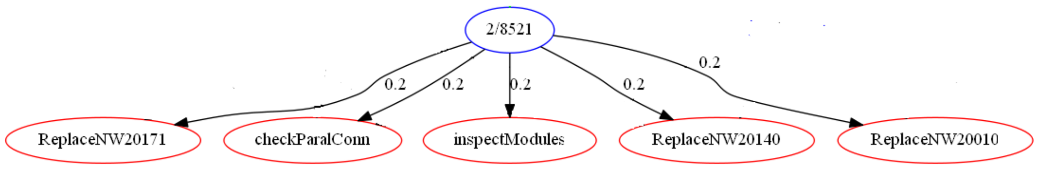


Figure 5.10: The figure shows the associations between alarm 8521 *and its maintenance actions.*

In this case a transition represents the selection of maintenance actions needed for repairing a device. A transition can be stronger or weaker depending on the number of times an action is needed for successfully resolve that specific alarm; thus we add a probability that represents such strength. As shown in the example, the probability is equal on each transition for the same alarm, because the dataset we have contains the following information: given the alarm K, the possible maintenance actions are A, B, C. The dataset we need should contain the following information: given the alarm K we repaired the system using A in n cases, B in m cases, and so on. Section 5.6.2 hypothesises the presence of that file and extends the model.

5.6.2 Adding service log file

A good improvement of the model regards the inclusion of what we call *service log file*. This file should be compiled by the customer service every time it is called for a system reparation. It should include, at least, the alarm code found and the maintenance actions performed to repair the system. Moreover, this file could include the order of the actions, the time spent to repair the UPS and the cost of each action. Table 5.12 represents two entry of how the service log file could be.

Id-intervention	Id	Code	Order	Id-reparation	Action	Time (s)	Cost(€)
1	2	1023	1	2	checkVin	150	0
1	2	1023	2	2	replaceXXX	500	350

Table 5.12: This table shows two records of the service log file.

- Id-intervention is an incremental number that represents intervention identification.
- Id and Code are used to univocally identify the device and the alarm code that triggered the intervention.
- Order is used to store the numerical order of the actions taken on the system.
- Id-reparation and action contain the maintenance action taken and the number of the device where the customer service performs the action.
- Time and Cost are used to save how expensive is an action with respect to the client or the provider.

For each intervention, these attributes are quite easy to collect and save on a single file and they could improve a lot the expressiveness of the model. We now provide some examples of the usage of this model.

First, instead of using the static manual that provides only a list of the possible faults on the subcomponents, the model can be considered as a dynamic service manual which relates alarms to maintenance actions with some probability. The customer service can check the most probable action to take in order to repair a device. The actions can be reported by considering the repairing order and the customer service can blindly follow it. In case more actions have the same probability, one could be chosen by looking at the time or cost attribute and minimize the intervention cost.

Second, the model created can be used to evaluate the real points of weakness of the system by querying which replacing action is associated to the most extensive set of alarms, this allows an evaluation of the general behaviour of the system.

5.6.3 Transition colouring

By adding more than one device to the model it is easy to have too many transitions drawn on the final model and the visualization is no more clear. We have already explained in the previous sections how the transitions between states could be filtered by using the probability and the mean time between events. By removing, for instance, the transitions with a probability below 0.15 we may remove strong dependencies of the real system that are not fully captured by our data. By considering the results of this chapter, we are able to remove transitions only in the hypothesis of the statistical value of the data. Since we work on a dataset without this hypothesis we can only cluster the transitions by their importance by using different colours. A transition is marked in green if its probability is considerably high, when the probability is not so high but the transition has a good coefficient of variation (CV), it is in red.

On the models introduced in the following chapter we draw the transitions by using the following rules:

- When the probability of that transition is more than 0.3, the transition is in green.
- When the probability is below 0.3 but the ratio between the standard deviation and the mean of the time between the two states is below 0.3, the transition is drawn in red.
- The transition is in black on the other cases.

Chapter 6

Implementation details and examples of use

In this chapter, we summarise the architecture of the application developed and we show some relevant results from which we can derive insightful conclusions. First, in Section 6.1 we describe the main elements of the architecture of the final application. In Section 6.2 we provide some interesting examples.

6.1 Architecture

Now, we provide a detailed explanation of the architecture of the application we have developed. The application has to satisfy two requirements: to be used by people who do not know programming languages and to be extended easily in case something more specific is needed. To meet these requirements, we implement the model view controller pattern (MVC), because it allows fast deployment and easy extensibility of the parameters [38] and a Graphical User Interface (GUI) because it allows the application to be easy to use. We now provide more details on each part of the application by describing the Model, the View, and The Controller. At the end of this section we present some considerations about the transition matrix and its implementation.

6.1.1 Model

The model contains all the parameters used by the application. These parameters are both initialised by reading the input files and by the input of the user. The model also contains all the methods needed to access those

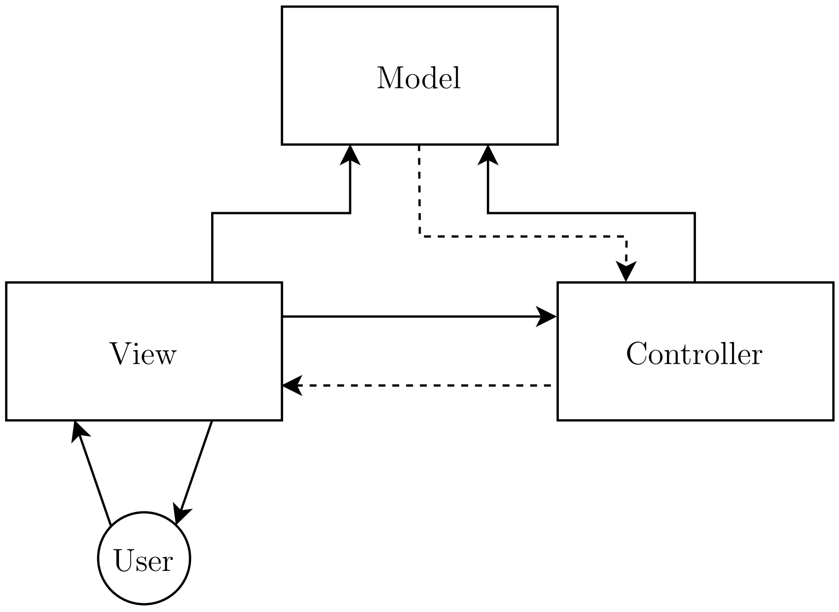


Figure 6.1: The figure shows how the main components of the application exchange messages.

parameters. A list of the attributes with a brief explanation is reported below:

- ***devices[](:Integer)*** This attribute contains the list of all the devices read from the event log file. In our case, the file has four different devices: $1, 2, 5, 6$.
- ***dates[](:String)*** This attribute contains the list of all the dates from which the events come. In our case, the events are written on the event log file in three different days: $9/05/2017, 10/05/2017, 11/05/2017$.
- ***n(:Integer)*** This attribute contains the maximum chain length of the model. The user sets this parameter during the execution of the application.
- ***deltatime[](:Integer)*** This attribute contains the ΔT described in Section 5.2.1. The user sets this parameter before building the Markov chains model.
- ***resetDeviceNumber[](:Boolean)*** This flag is used to select the state representation. When it is $True$, the application does not consider the

id during the construction of the chains. When it is *False*, during the construction of the chain, the application also includes the *id* of the device that generates that event. A detailed description is provided in Section 5.3.

- ***makeChain[](:Boolean)*** This flag is used to enable the *Level* state representation. When is *True*, to represent the states, the application includes the *Level***. A detailed description is provided in Section 5.4.

- ***selectedDevices[](:Integer)*** This array contains the list of devices to consider during the construction of the model. The events coming from different devices will be not considered. A detailed description is provided in Section 5.1.

- ***selectedDays[](:String)*** This array contains the list of days to consider during the construction of the model. The events coming from different days will be not considered. A detailed description is provided in Section 5.1.

6.1.2 View

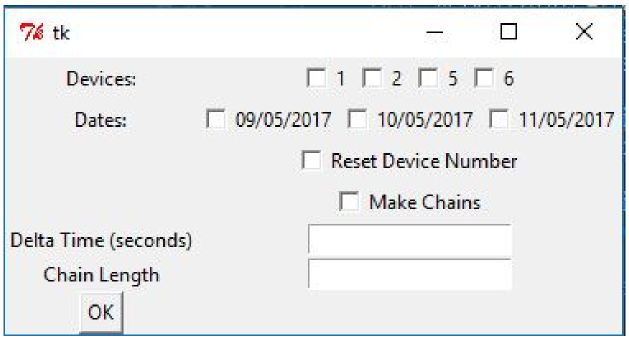
The view is the part of the application that has the goal of receiving the parameters by the user and displaying the results. By using the Tkinter library, the view creates a simple graphical user interface. In our application, we need to develop two different windows, the first allows the user to set all the parameters of the model and the second shows the final model.

Once the application starts, the first window displayed is reported in Figure 6.2. By setting the values for all the attributes and pressing the *submit* button, the application pops out a new window with the Markov chains model displayed. The application allows the user to change the parameters and to create a new model without restarting.

We use two classes:

- ***Checkbar:*** implements a generic checkbox input widely used for asking parameters to the user.

- ***Gui:*** implements all the logic of the graphical user interface.



Devices: ☐ 1 ☐ 2 ☐ 5 ☐ 6

Dates: ☐ 09/05/2017 ☐ 10/05/2017 ☐ 11/05/2017

☐ Reset Device Number

☐ Make Chains

Delta Time (seconds)

Chain Length

OK

Figure 6.2: The figure shows the first window of the application.

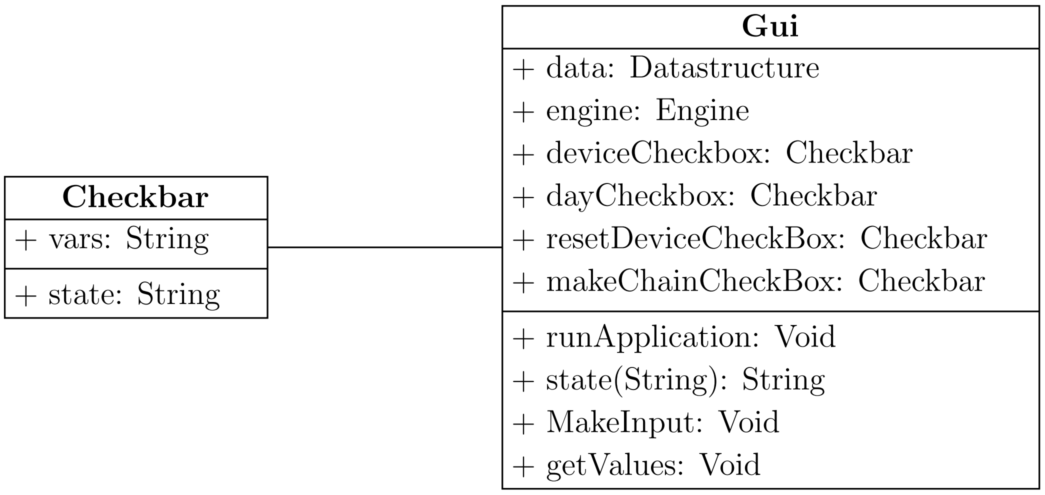


Figure 6.3: The figure shows the class diagram of the view.

6.1.3 Controller

The controller is the most important class of the application because it implements the real logic to compute the chains. This class contains the datasets and all the needed methods to work on it. The controller receives as input the parameters of the model and, by using the datasets, it computes the Markov chains model of the system. As output, it saves on the hard disk an image containing the graph composed of all the states, the transitions and the probabilities. The application also writes on the hard disk a text file containing the parameters used to build the model and a *.csv* file containing the final transition matrix of the model.

6.1.3.1 Summary of the methods

We provide now the list of the methods of the controller. Table 6.1 shows the name of every method and the section where it is described in this thesis.

Preprocessing	
addTimeStamp	Section 4.1
joinMessagesAndValues	Section 4.1
writeDatastructure	Section 4.1
Model Generation: Computing States	
selectDataFromDays	Section 5.1
removeDev	Section 5.1
sequenceMiningByTime	Section 5.2
cutMessages	Section 5.2
CreateInitState	Section 5.2.4
Model Generation: Labeling Transitions	
computeProbTargetGivenFeature	Section 5.5.3.1
addMeanTimeAndStdDev	Section 5.5.3.2
PostProcessing	
createMaintenance	Section 5.6.1
DefineColor	Section 5.6.3
createGraph	
drawGraph	

Table 6.1: This table contains a summary of the primary methods of the controller.

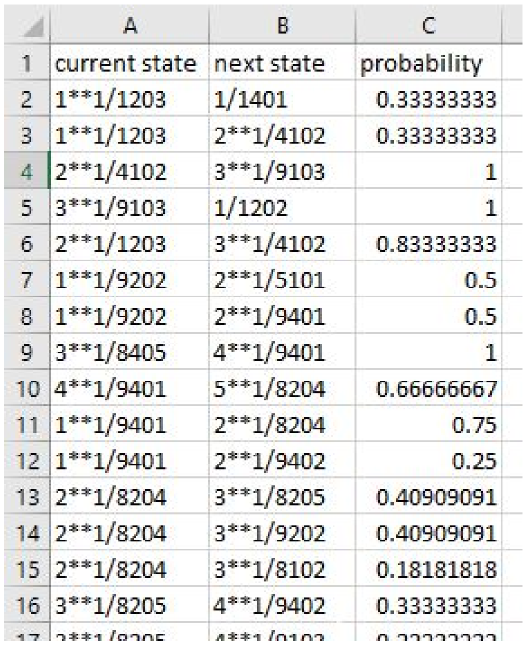
The preprocessing methods read the data from the file and perform all the initial manipulations needed. The first phase of the model generation is in charge of creating the chains by selecting the data, associating them according to some parameters, and creating the chains and the pairs. The second part of the model generation is used to label the transitions by computing the probability, the mean time and the standard deviation. In the postprocessing part there are two methods used to add the maintenance actions and colouring the transitions. In the end, in the support methods we have the functions to draw the graph.

The primary method of the controller is called *makeModel* and it is not reported in the table above. This method is called by the view and it contains all the steps performed during the construction of the models, in facts this method calls in the right order most of the methods reported in Table 6.1. In Appendix B we provide a clear explanation of this method.

6.1.4 Transition matrix

We provide some details regarding the implementation of the transition matrix. As explained in Section 5.5.2, every state is related averagely to 5 other states. Having a complete matrix reporting the transition will be inefficient in terms of memory. In fact, considering N states, we save an $N \times N$ matrix, but the application uses only $5 \times N$ cells.

We have decided to implement the transition matrix by using a table with three columns: the first includes the state of the system at time t namely *current state*, the second column contains the state of the system at time $t+1$, namely *next state*. The third column contains the probability of the transition between the first element and the second. By using this datastructure we memorise only the needed information. Figure 6.4 represents a portion of the transition matrix of a model.



	A	B	C
1	current state	next state	probability
2	1**1/1203	1/1401	0.33333333
3	1**1/1203	2**1/4102	0.33333333
4	2**1/4102	3**1/9103	1
5	3**1/9103	1/1202	1
6	2**1/1203	3**1/4102	0.83333333
7	1**1/9202	2**1/5101	0.5
8	1**1/9202	2**1/9401	0.5
9	3**1/8405	4**1/9401	1
10	4**1/9401	5**1/8204	0.66666667
11	1**1/9401	2**1/8204	0.75
12	1**1/9401	2**1/9402	0.25
13	2**1/8204	3**1/8205	0.40909091
14	2**1/8204	3**1/9202	0.40909091
15	2**1/8204	3**1/8102	0.18181818
16	3**1/8205	4**1/9402	0.33333333
17	3**1/8205	4**1/9103	0.33333333

Figure 6.4: The figure shows a portion of a real transition matrix.

6.2 Examples of use

For each section, in this analysis, we compare two models generated by taking into account different parameters. In Section 6.2.1 we compare the *code* state representation with respect to the *id/code*. In Section 6.2.1 we show the differences between *id/code* state representation with respect to the *level*. In Section 6.2.3 we consider two different chain lengths. In Section 6.2.4 we compare two models generated by considering different ΔT.

6.2.1 *code* vs. *id/code* state representation

As we discussed in Section 5.3, the model could be built by using different state representations. In this example, we highlight the differences between the *code* and *id/code* representations. Figure A.1 in Appendix A shows the behaviour of the system by using the *code* representation, and the Figure A.2 in Appendix A shows how the messages are related among the devices by using the *id/code* representation. Both the models are generated by using the following parameters:

1. **Date:** 10/05/2017
2. **Device:** 2,5
3. ΔT**:** 800
4. **Chain Length:** 5

As we have already discussed, the *code* representation is convenient to understand the behaviour of the system, but it does not show which of how the devices exchange the messages. The *id/code* representation includes in each state also the device number, and it allows to understand how the devices influence each other.

The first model includes 20 messages and 8 alarms, the second model has 27 messages and 12 alarms. Since we build the models by considering two devices, there are different messages with the same code but coming from different devices, in the first model they are merged in the same state, while in the second model these messages are divided into different states. Figure 6.5 and Figure 6.6 show this behaviour of the models.

We can analyse the behaviour of the *code* representation by considering Figure 6.5. The portion of the model shows a cycle on the messages 8109,

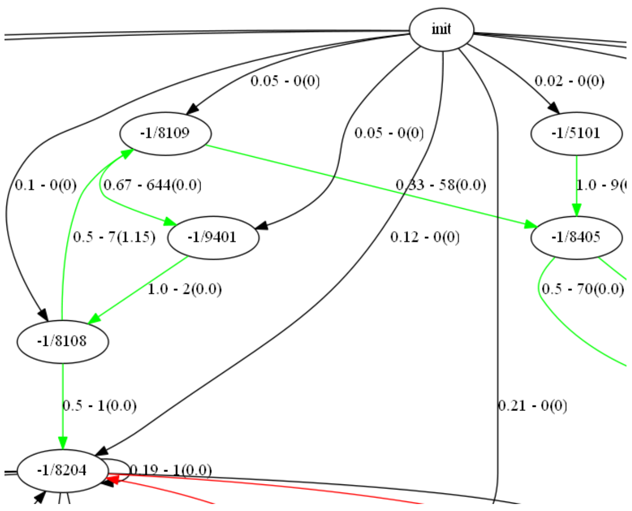


Figure 6.5: The figure shows a portion of the model reported in Figure A.1 of Appendix A.

9401, 8108. When the system reaches one of these states, it is likely to stay inside the loop. In the model, there is also a loop on the message 8204, therefore, we can conclude that it is probable to see many times in a row the message 8204.

By using Figure 6.6, we can introduce second model. In this case, the loop consists of five different states: 2/8108, 5/8109, 2/9401, 5/8108, 2/8109, and it is possible to understand better the flow of messages. By considering the green arcs, there are two likely sequences of messages: the first is composed of 2/8108, 5/8109, 2/9401, 5/8108, the second is composed of 2/8108, 2/8109, 2/9401, 5/8108. Both these sequences share devices 2 and 5. Thus we can conclude that these devices are strongly connected.

We highlight another difference between the *code* and *id/code* representation.

In case of the *code* representation, reported in Figure 6.7, the chain starts with 5101, it continues with the message 8405, and it is divided into two different chains. In the end both the chains are merged at the message 9103.

In the second case, reported in Figure 6.8, we understand that all the sequences described before is generated from the same device, in our example the number 5. As a result, we understand that this sequence of messages and the final alarm involve only on a specific device. As we have already understood, the difference between *code* and *id/code* is that the second provides

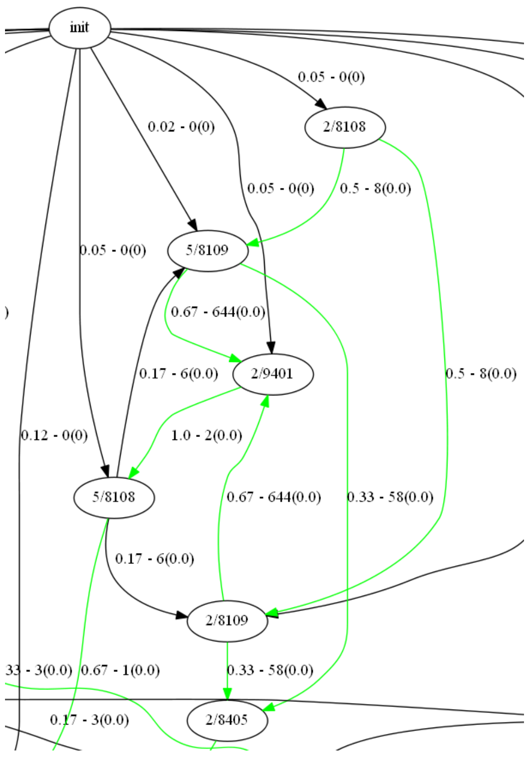


Figure 6.6: The figure shows a portion of the model reported in Figure A.2 of Appendix A.

more information.

6.2.2 *id/code* vs. *level* state representation

In this example, we highlight the differences between models created by using the *id/code* representation and by using the *level* representation. We also provide an example of how this model could be used in a real case. In both cases, we consider the following parameters:

1. **Date:** 11/05/2017

2. **Device:** 2

3. **ΔT:** 1000

4. **Chain Length:** 4

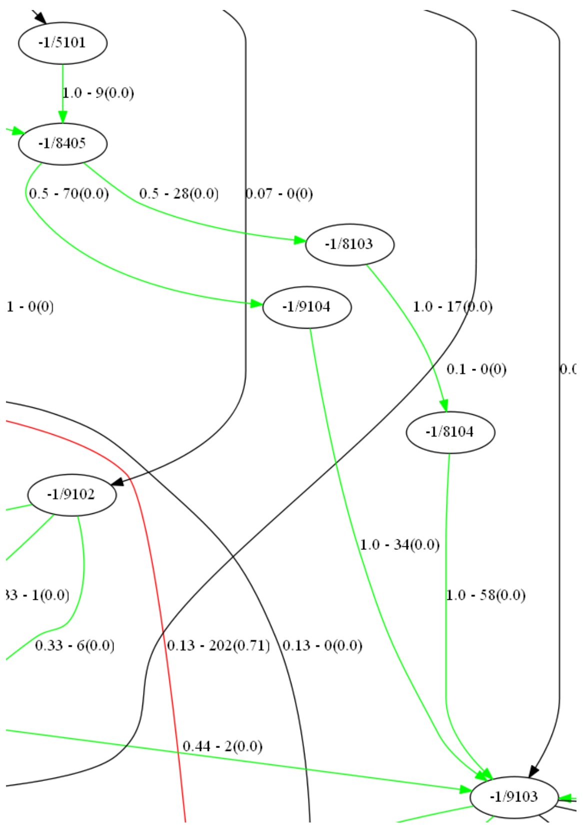


Figure 6.7: The figure shows a portion of the model reported in Figure A.1 of Appendix A.

Figure 6.9 shows the model built by using the *id/code* representation. Starting from the *init* state, the first message is always 8204, which is also the last message before every alarm. Indeed, every chain of this model starts and ends with the message 8204, and due to this representation, there are loops among states. We identify two different loops: the first pass through the state 8204, it reaches the state 8205, and after it returns to the state 8204. The second loop starts from the state 8204, then it goes to the state 9203, and it reaches the state 8204 in the end.

The alarm 101 is the most probable with a probability of 24% and a mean time of 4 seconds. However, the transition is reported in black because the threshold is 30%. Following the probability, the second alarm that can be raised is the 8521 with a probability of 12% and within 10 seconds, and the standard deviation of 0.58, therefore the transition is reported in red.

Figure 6.10 shows the model generated by using the same parameters and

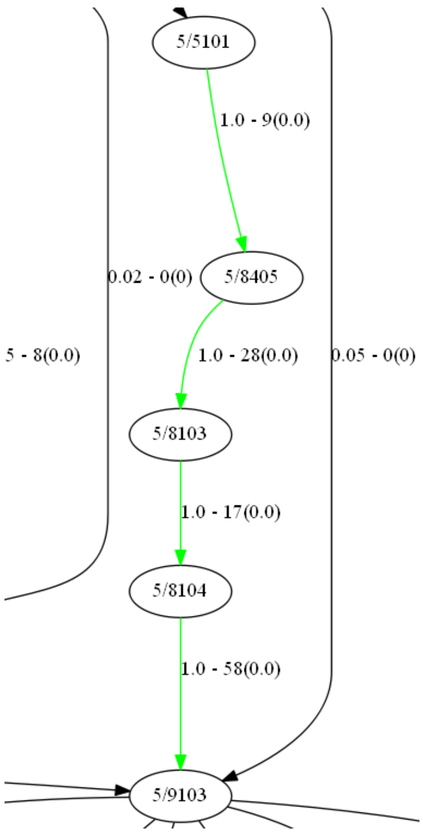


Figure 6.8: The figure shows a portion of the model reported in the Figure A.2 of the Appendix A.

the *level* state representation. The loops are removed by dividing the state 2/8204 into two different levels: the first and the third.

Resuming the considerations of Section 5.2.3, here we can also deal with the problem of events that occur at the same time. In Figure 6.10, there are the states $2 * *2/8205$ and $2 * *2/9203$ that are both coming precisely 20 seconds after the state $1 * *2/8204$. There is no way to determine which among the states of the second *level* occurred first. For this reason, the algorithm divides the chains after the state $1 * *2/8204$, and it associates to each transition the same probability. As explained, only one between the two patterns is correct, but the only way to determine it is to consider much more events.

With respect to the first model, in this graph, the probability of reaching directly the alarm 101 by being in the state $1**2/8204$ is lower because this probability is divided between two different transitions, namely ($1**2/8204$,

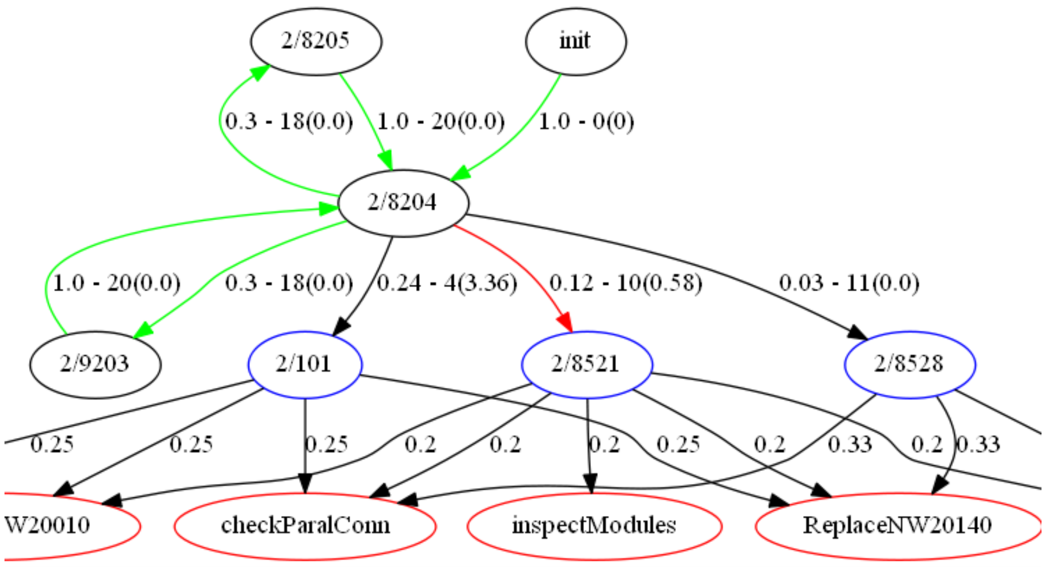


Figure 6.9: The figure shows a portion of the model reported in Figure A.3 of Appendix A.

2/101) and $(3 * *2/8204, 2/101)$.

By using the parameters defined in Section 5.6.3, the transitions have different colours. The green means a strong probability and the red a good distribution over time, when a transition belongs to both the categories, the application shows the transition in green. The black is used when a transition is not considered important enough.

Let us assume to have turned on the system and, after some minutes, the event log file shows the messages 8204, 9203, and 8204. By considering every possible path, the most probable alarms are 101 or 8521. Those alarms have *check the parallel connection* and *replace of the component* $NW20140$ as mutual maintenance actions so we can perform these actions to prevent the system failure.

6.2.3 Comparing different chain length (N)

In this example, we inspect the differences between a model with $N = 10$ and a model having $N = 4$. The parameters used to build the models are:

1. **Date:** 11/05/2017
2. **Device:** 2,5

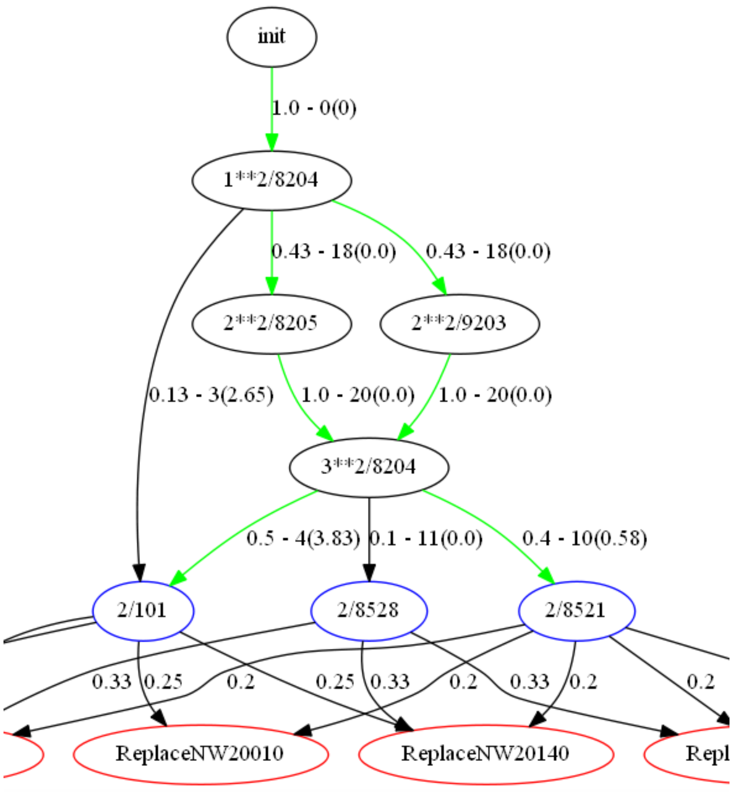


Figure 6.10: The figure shows a portion of the model reported in Figure A.4 of Appendix A.

3. ΔT**:** 1000

4. **State Representation:** $Level **id/code$

In Figure 6.11 the chain length is equal to 4. In this figure, we provide a zoom of the generated model. Figure A.5 in Appendix A contains the complete view of the model generated by the same parameters. In this model, we have a first level composed of 3 states, a second level composed of 5 states, a third level composed of 2 states and a fourth level composed of 3 states.

By considering a chain length equal to 4, we have a bottleneck on the third level composed by the states $3 **5/9202$ and $3 **5/9203$, because all the chains share the same two states at that level. A model built in this way is quite impractical because we cannot use it easily, from a state before the third level, to derive which are the states after that level. For instance, when the system is in the state $2 **5/9202$, we can reach both the states of the

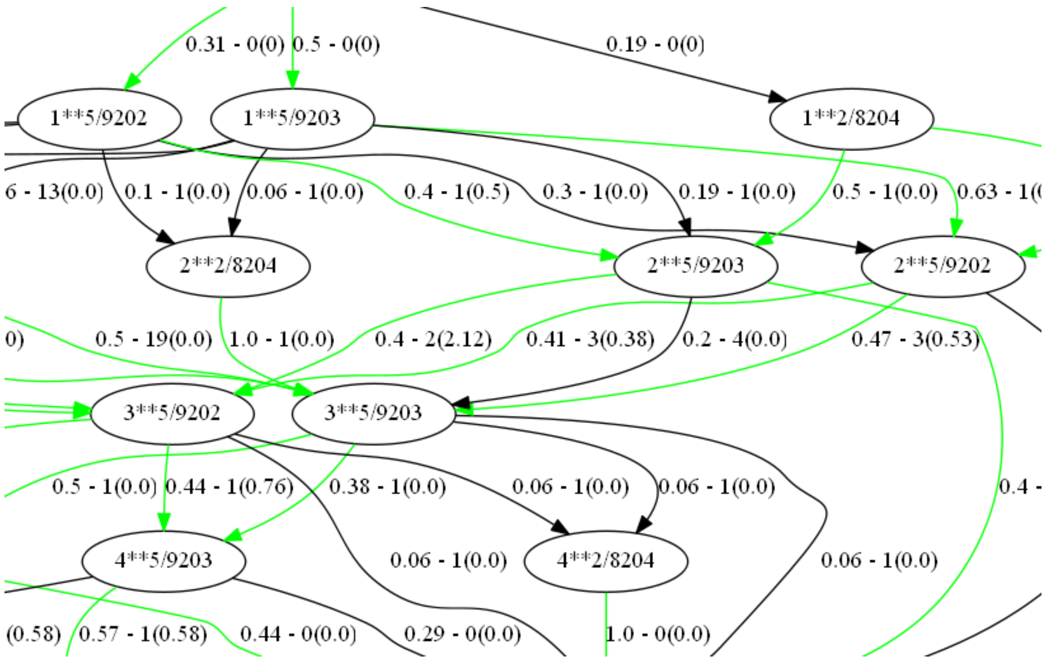


Figure 6.11: The figure shows a portion of the model reported in Figure A.5 of Appendix A.

third level with a probability higher than 40%, and, from this level, we can reach all the states at the fourth level.

To remove the presence of bottlenecks, we need to set our parameters in order to have almost the same number of states on each level. By extending the chain length up to 10, we provide a zoom of the resulting model in Figure 6.12. Apparently, by using this chain length, we resolve the problem. The model starts with a large set of possible states in the first three levels and then it converges in two different chains.

By using this model, it is easier to infer which chain we are following. For instance, given the sequence $1**5/9203$, $2**2/9203$ we are sure to belong on the chain to the right of Figure 6.12 because there are no transitions between the two chains.

When we set a small chain length, we may remove some associations among events, and by setting a high chain length, we may include relations that are not true. Therefore, we need to choose the chain length parameter carefully by considering how the system behaving.

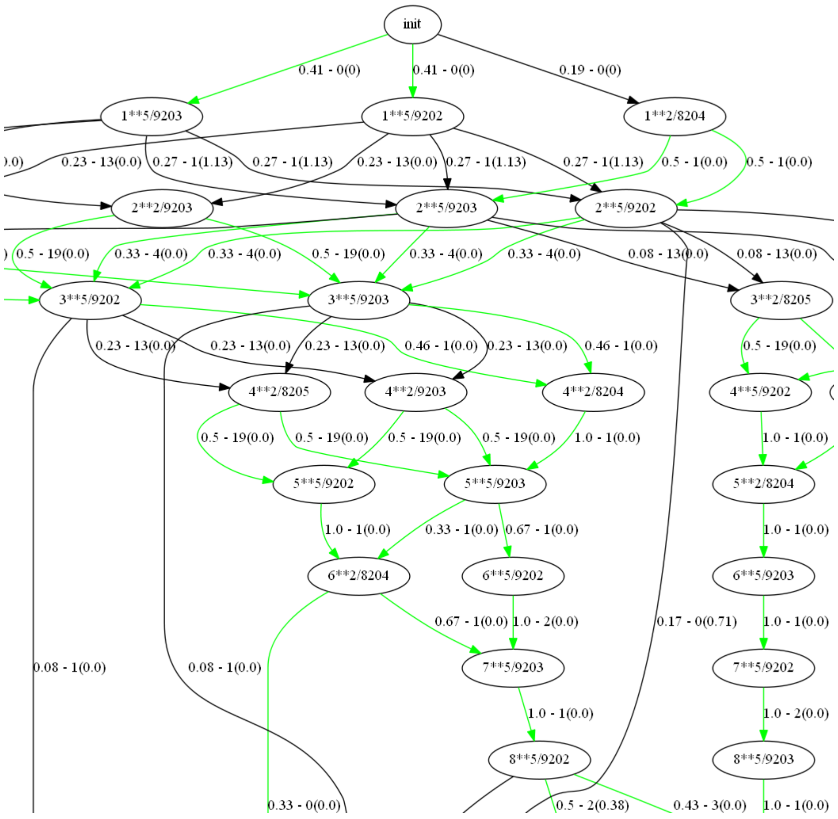


Figure 6.12: The figure shows a portion of the model reported in Figure A.6 of Appendix A.

6.2.4 Comparing different ΔT

We now evaluate how the ΔT influences the model. We have defined the TBE in Section 4.2.4, and we have hypothesised that a value between 150 and 300 seconds could be appropriate. Now, we build two models, the first by using a TBE equal to 30 seconds and the second by using a TBE equal to 500 seconds. The models share the following parameters:

1. **Date:** 10/05/2017
2. **Device:** 2
3. **N:** 5
4. **State Representation:** $Level **id/code$

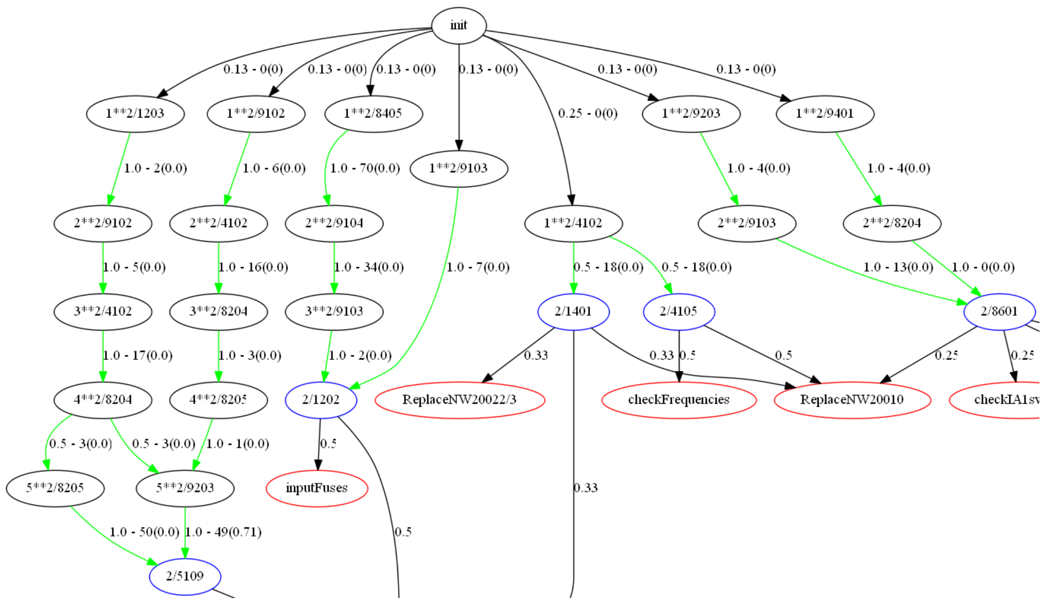


Figure 6.13: The figure shows a portion of the model reported in Figure A.7 of Appendix A.

In Figure 6.13, the ΔT is 150 seconds (TBE = 30 and Chains length = 5). We provide a zoom, the complete model is reported in Figure A.7 in Appendix A.

In this model we notice the presence of chains that have one or two messages before the alarms. For instance, given the state $1**2/9103$, we reach certainly the alarm 2/1202 with a chain composed of only one state. Two reasons cause that behaviour: The first concerns the fact that ΔT is not large enough to capture all the dependencies of the events, the second regards the possibility of a lack of data. Moreover, there is only one transition with a standard deviation different from zero: it means that we have an occurrence for each pair that is not enough to generate a real model of the system.

We compare the model before with the model that uses a ΔT equal to 2500 seconds (TBE = 500 and Chains length = 5). Figure 6.14 shows a portion of the model, the complete results are reported in the Figure A.8 of Appendix A.

By looking at Figure 6.14, with respect to the previous model, we find a new alarm, namely 2/8602, and it is associated with two messages, respectively $5**2/8204$ and $2**2/9103$, with a probability of 50% and more than 200 seconds. There are also different transitions with a probability of 100%

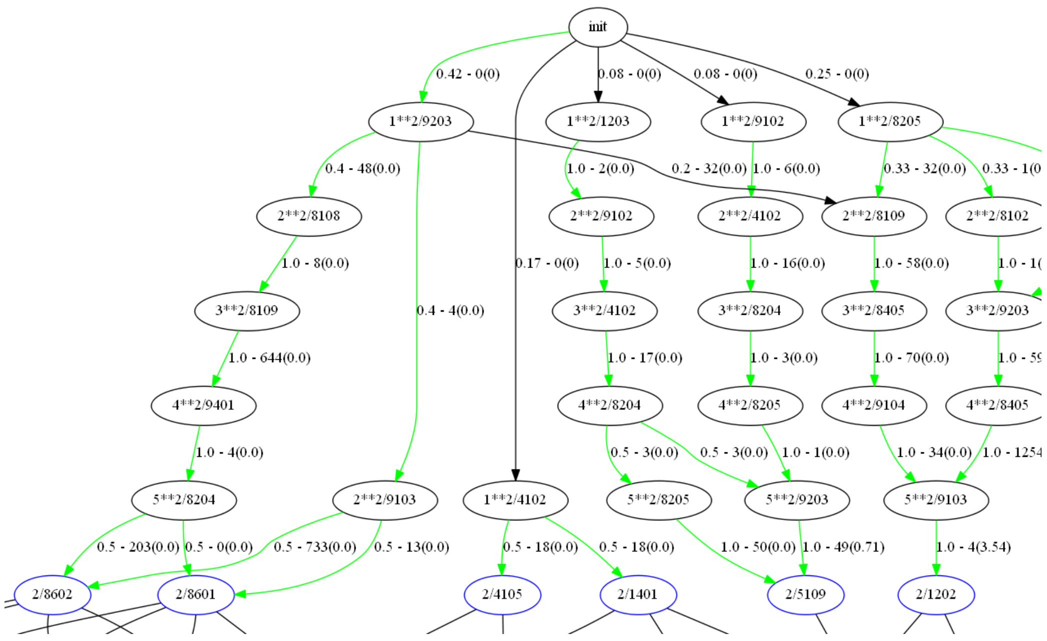


Figure 6.14: The figure shows a portion of the model reported in Figure A.8 of Appendix A.

and a TBE greater than 500 seconds. For instance, the transition between the message $3**2/8109$ and the message $4**2/9401$ occurs in 644 seconds, also the transition between the event $2**2/9103$ and the alarm 2/8601 occurs in 733 seconds. A good improvement of the model consists of analysing all the transitions with an high mean time. When the probability is high and the standard deviation low, a transition captures a real association of the data.

In the second model, the chains that end with the alarms 2/4103 and 2/1401 have only one message, and the time interval between the *init* state and the alarm is quite low (18 seconds) with a probability of 17%. All this chain looks a bit particular, we may assert that is probably an imprecision of the data. All the other chains have at least N messages, so we conclude that ΔT is too high and the only constraint that influences the model is the chain length.

In the end, by considering the model in Figure 6.14, we can hypnotise how to use this model to predict how the system behaves. Let us assume that we turn on the system and, after a while, the event log file contains the message 2/9203. By looking at the model, the 80% of the times, it ends up with the alarms 2/8602 and 2/8601, and they have the same maintenance actions. Before knowing the real alarm, the customer service is able to perform some check like *checkIA1switch*. In this particular case, it is useless to blindly replace critical components because we have no evidence of saying which specific sub-component does not work. Let us suppose that, after the first message, the event log file also contains $2 * *9103$. By using the model, the system should reach either the alarm 2/8601 or the alarm 2/8602 respectively after 13 and 733 seconds. In this case, the customer service should wait a couple of minutes before repair the system, if any alarm arrives, it can suppose that the next alarm will be the 2/8602.

Chapter 7

Conclusions and future work

In this chapter, we present the main conclusions of our work in Section 7.1. In Section 7.2 we describe its limitations. Finally in Section 7.3 we describe how to use this work for future analysis.

7.1 Conclusions

In this work, we have shown how the Markov chain model could be applied for the dependability evaluation and analysis in the context of UPS systems. Mainly, we studied the relations among the events derived from a UPS system. Unlikely most of the previous models used to analyse the behaviour of these systems, our model is based only on the log data without any domain knowledge of how the devices are connected.

Since we did not have enough data, we spent much effort to analyse the input files in terms of distribution of the data over devices, dates, and time intervals. In Chapter 3, we have presented the preprocessing techniques used to read and understand the input files. In this phase, we understood the limitations of the information provided and highlighted the main assumptions on which the work is based (Section 4.2.1 and Section 4.2.2). We have then demonstrated an association with messages and alarms based on the *time between events*.

After this analysis, we have started to work with Markov chains in order to precisely represent the system in terms of states and transitions. First, we have chosen to build chains composed of a list of messages ending with an alarm. The chains rely on two parameters: the maximum time interval

ΔT between the first message and the alarm and the maximum chains length (N), as discussed in Section 5.2. We have developed some state representations able to characterise the behaviour of the system, in particular we have introduced the *code* representation (Section 5.3.1) and the *id/code* representation. Since the first two methods show some problems (described in Section 5.3.2), we have decided to build also the *Level**id/code* representation that can keep track of the history of the events occurred during the execution.

We have decided to add to each transition the probability of occurrence, the mean and the standard deviation of the time of occurrence in order to represent the frequency at which the two events occur. For understanding the feasibility of the computation, we have evaluated the size of the transition matrix produced (Section 5.5.2). Since, up to now, the models do not include all the information provided by the company, we have introduced associations between alarms and maintenance actions. These associations provide a description of which actions could be performed on the system to repair a device after a failure is identified (Section 5.6.1). In this phase, we have discovered a lack of data that could only be resolved by using the *service log file* hypothesised in Section 5.6.2. We have also introduced the transition colouring based on the probability and the coefficient of variation that provides a way to visually understand the importance of a transition (Section 5.6.3).

Considering the business requirements of the company that provided the data, we have developed a software program that uses a simple graphical user interface to acquire the parameters from the user and that builds the models and displays them graphically. The application can be extended easily by adding domain knowledge of the system or company-specific information to build richer models. As output, the application creates three files: the first contains an image of the model, the second contains the parameters used to build the model, and the third contains the computed transition matrix. In this phase we have handled the problem of efficiently memorising the transition matrix, the solution is reported in Section 6.1.4.

In the end, we have analysed the results obtained by creating several different models and understanding their characteristics in terms of strengths and weaknesses.

7.2 Limitations

By limitations we mean those characteristics of design or methodology that impacted or influenced the interpretation of the findings from research [39]. In this section, we describe the main limitations encountered both during the work and for the future use of the models on UPS systems.

One of the main limitations of our study is the numerosity and the reliability of the event log file. As described in Chapter 3, the number of events used to build models is not enough to provide results that can be practically used by the company. Moreover, we had no way to verify the correctness of the information contained in the files. We considered them as ground truth and, with the availability of a large amount of data, the algorithm can compute a real model of the system. This limitation also affects the possibility to evaluate the results obtained. Up to now, we were only able to perform visual checks on the created models without the possibility to give any accuracy measure. We tried to supply to this lack of information by inserting in the models some domain knowledge but, since the information required was some strategic business data, we were unable to get the data required. After providing the results to the company that followed our thesis we received positive feedbacks. The company is also interested to continue the work on these models.

By looking at the results obtained, we could also discover some limitations of these types of models applied to UPS systems. As we have understood from Section 6.2, given a message, the model can predict future alarms with a time window of at most few minutes. For instance, by considering the model in Figure A.6, the prediction time is almost 30 seconds, or, by considering the model in Figure A.8, that time is around two minutes. We believe that the prediction of the future alarms may come too late to allow the maintenance service to repair the system before it fails.

7.3 Future work

Many ideas arisen during the work can be analysed even further. First of all, up to now, by analysing the results obtained, we can confirm some assumptions made during the thesis. The core of the work consists in finding an association between messages and alarms, and in our work, it is performed by considering the ΔT and the chain length (N) statically provided by the user. It is difficult to find a perfect set of parameters able to guarantee the presence of all the relevant associations among messages without creating an overhead of associations. In order to improve the effectiveness of a model, we must choose dynamically the parameters to build it. In more detail, for capturing all the correlations, the algorithm should extend or reduce automatically N or ΔT based on the events of the event log file at that specific time.

Another interesting work, based on this thesis, could be performed by extending the current model and including a real service log file in order to capture the correlations among alarms and maintenance strategies. In this case, the Markov chain model might be converted into a hidden Markov model. By considering the HMM defined in Section 2.3.2, every message and alarm will be the states of the model, and every maintenance actions will be the observation. The chains will be composed in the same way as in this thesis but, for each state, there will also be a transition to the maintenance actions. By using this model, we can potentially extend the prediction time of the model by associating the messages with the maintenance actions. Let us suppose to have the HMM and to observe, in the event log file, the message A. By querying the model, we can receive a list of the most likely maintenance actions and a list of possible future messages. Already after the first message, we will be able to check the maintenance actions needed and, in case the probability of a specific maintenance action is very high compared to the others, we can already repair the system. In case almost all the maintenance actions have the same probability, we can wait for a message B of the system, join the maintenance actions of both the messages and extract the most probable set of actions to repair the system and, in fact, improve the dependability of the system.

Bibliography

[1] Wikipedia contributors, “Markov chain — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 28-October-2018].

[2] Wikipedia contributors, “Hidden Markov model — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 14-November-2018].

[3] Wikipedia contributors, “Markov model — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 14-November-2018].

[4] Wikipedia contributors, “Uninterruptible power supply — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 23-November-2018].

[5] C. Alberto, E. S. K., P. Gabriele, R. Enrico, and A. Francesco, “A bayesian network framework for operations of circuit breakers,” Proceedings of the IEEE International Conference on Industrial Technology., 2019.

[6] A. Pozzi and L. Costantini, “Fault analysis of a complex electrical distribution system with Bayesian networks and Markov chains,” 2018. Thesis, Politecnico di Milano.

[7] M. K. Rahmat and M. N. Sani, “Fault tree analysis in UPS reliability estimation,” in *Proceedings of the 4th International Conference on Engineering Technology and Technopreneuship (ICE2T)*, pp. 240–245, IEEE, 2014.

[8] M. A. Biraol, S. Momoh, and I. G. Saidu, “Failure events analysis of uninterruptible power supply (UPS) in nigeria,” *International Journal of Engineering Science Invention*, vol. 1, pp. 26–32, 2012.

[9] H. Guo and H. Liao, “Methods of reliability demonstration testing and their relationships,” *IEEE Transactions on Reliability*, vol. 61, no. 1, pp. 231–237, 2012.

[10] A. Avizienis, J.-C. Laprie, B. Randell, *et al.*, *Fundamental concepts of dependability.* University of Newcastle upon Tyne, Computing Science, 2001.

[11] B. W. Johnson, “An introduction to the design and analysis of fault-tolerant systems, fault-tolerant computer system design,” 1996. Prentice-Hall, Inc.

[12] P. Luigi and C. R. Daniele, *Modeling and analysis of dependable systems: a probabilistic graphical model perspective.* World Scientific, 2015.

[13] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and Yin, “A review of process fault detection and diagnosis: Part II: Qualitative models and search strategies,” *Computers & chemical engineering*, vol. 27, no. 3, pp. 313–326, 2003.

[14] W. G. Schneeweiss, “Advanced fault tree modeling,” *Journal of Universal Computer Science*, vol. 5, no. 10, pp. 633–643, 1999.

[15] S. Distefano and A. Puliafito, “Dynamic reliability block diagrams: Overview of a methodology,” in *Proceedings of the European Safety and Reliability Conference (ESREL)*, vol. 7, pp. 1059–1068, 2007.

[16] V. Venkatasubramanian, R. Rengaswamy, S. N. Kavuri, and K. Yin, “A review of process fault detection and diagnosis: Part III: Process history based methods,” *Computers & chemical engineering*, vol. 27, no. 3, pp. 327–346, 2003.

[17] A. Tolver, “An introduction to Markov chains,” 2016. Department of Mathematical Sciences, University of Copenhagen.

[18] T. Liu, S. Chiou, *et al.*, “The application of Petri nets to failure analysis,” *Reliability Engineering & System Safety*, vol. 57, no. 2, pp. 129–142, 1997.

[19] N. Thomas Dyhre and J. Finn Verner, *Bayesian Network and Decision Graph.* Springer, 2001.

[20] A. Ciobanu, F. Munteanu, and C. Nemes, "Bayesian networks utilization for reliability evaluation of power systems," in *Proceedings of the International Conference and Exposition on Electrical and Power Engineering (EPE)*, pp. 837–841, IEEE, 2016.

[21] J. Liu and X. Zhang, "Detection method of intermittent faults in electronic systems based on Markov model," in *Proceedings of the Fourth International Symposium on Computational Intelligence and Design (ISCID)*, vol. 1, pp. 216–219, IEEE, 2011.

[22] Wikipedia contributors, "Markov decision process — Wikipedia, the free encyclopedia," 2018. [Online; accessed 14-November-2018].

[23] Wikipedia contributors, "Partially observable Markov decision process — Wikipedia, the free encyclopedia," 2018. [Online; accessed 14-November-2018].

[24] S. S. Haykin, S. S. Haykin, S. S. Haykin, and S. S. Haykin, *Neural networks and learning machines*, vol. 3. 2009. Pearson Upper Saddle River.

[25] L. R. Rabiner, "A tutorial on Hidden markov models and selected applications in speech recognition," *Proceedings of the IEEE*, vol. 77, no. 2, pp. 257–286, 1989.

[26] G. L. Kouemou, "History and theoretical basics of hidden markov models," in *Hidden Markov Models* (P. Dymarski, ed.), ch. 1, Rijeka: IntechOpen, 2011.

[27] H.-K. Yun, A. Smith, and H. Silverman, "Speech recognition HMM training on reconfigurable parallel processor," in *Proceedings of The 5th Annual IEEE Symposium on Field-Programmable Custom Computing Machines*, pp. 242–243, IEEE, 1997.

[28] B.-W. Min, H.-S. Yoon, J. Soh, Y.-M. Yang, and T. Ejima, "Hand gesture recognition using hidden markov models," in *Proceedings of the International Conference on Systems, Man, and Cybernetics, Computational Cybernetics and Simulation.*, vol. 5, pp. 4232–4235, IEEE, 1997.

[29] G. Garajayewa and M. Hofreiter, “Markov chains for Fault Diagnosis,” in *Proceedings of the Seminar in Instruments and Control, Ostrava.*, Department of Mathematical Sciences, University of Copenhagen., 2001.

[30] H. S. S. Zaidi, W. G. Zanardelli, S. Aviyente, and E. G. Strangas, “Prognosis of electrical faults in permanent magnet AC machines using the Hidden Markov model,” in *Proceedings of the IEEE Industrial Electronics Society (IECON)*, pp. 2634–2640, IEEE, 2010.

[31] D. P. Jovanović and P. K. Pollett, “Fault diagnosis using consensus of Markov chains,” in *Proceedings of the 3rd International Workshop on Dependable Control of Discrete Systems (DCDS)*, pp. 65–71, IEEE, 2011.

[32] A. Nasiri, S. B. Bekiarov, and A. Emadi, *Uninterruptible power supplies and active filters.* CRC press, 2004. CRC press.

[33] Wikipedia contributors, “Pandas (software) — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 6-November-2018].

[34] Wikipedia contributors, “Networkx — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 6-November-2018].

[35] Wikipedia contributors, “Graphviz — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 6-November-2018].

[36] Wikipedia contributors, “Tkinter — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 6-November-2018].

[37] Wikipedia contributors, “Coefficient of variation — Wikipedia, the free encyclopedia,” 2018. [Online; accessed 10-November-2018].

[38] G. E. Krasner, S. T. Pope, *et al.*, “A description of the model-view-controller user interface paradigm in the smalltalk-80 system,” *Journal of object oriented programming*, vol. 1, no. 3, pp. 26–49, 1988.

[39] J. H. Price and J. Murnan, *Research limitations and the necessity of reporting them.* 2004. Taylor & Francis Group.

[40] F. Jelinek and R. L. Mercer, “Interpolated estimation of Markov source parameters from sparse data,” in *Proceedings of the Workshop on Pattern Recognition in Practice* (E. S. Gelsema and L. N. Kanal, eds.), pp. 381–397, Amsterdam: North Holland, 1980.

[41] M. Servitja Robert, “A first study on Hidden Markov models and one application in speech recognition,” 2016. Linkoping University, Department of Mathematics, Mathematical Statistics.

[42] X. D. Huang, Y. Ariki, and M. A. Jack, *Hidden Markov models for speech recognition.* 1990. Edinburgh university press Edinburgh.

[43] L. R. Rabiner, J. G. Wilpon, and B.-H. Juang, “A segmental k-means training procedure for connected word recognition,” *AT&T technical journal*, vol. 65, no. 3, pp. 21–31, 1986. Wiley Online Library.

Appendix A

List of the discovered models

We provide here a list of the models generated by our application. Since we reported the results with a zoom, we provide now the full images. For each section we include the parameters used and the model generated.

A.1 Model 1

1. **Date :**10/05/2017
2. **Device:** 2,5
3. ΔT**:** 800
4. N**:** 5
5. **Representation:** *code*

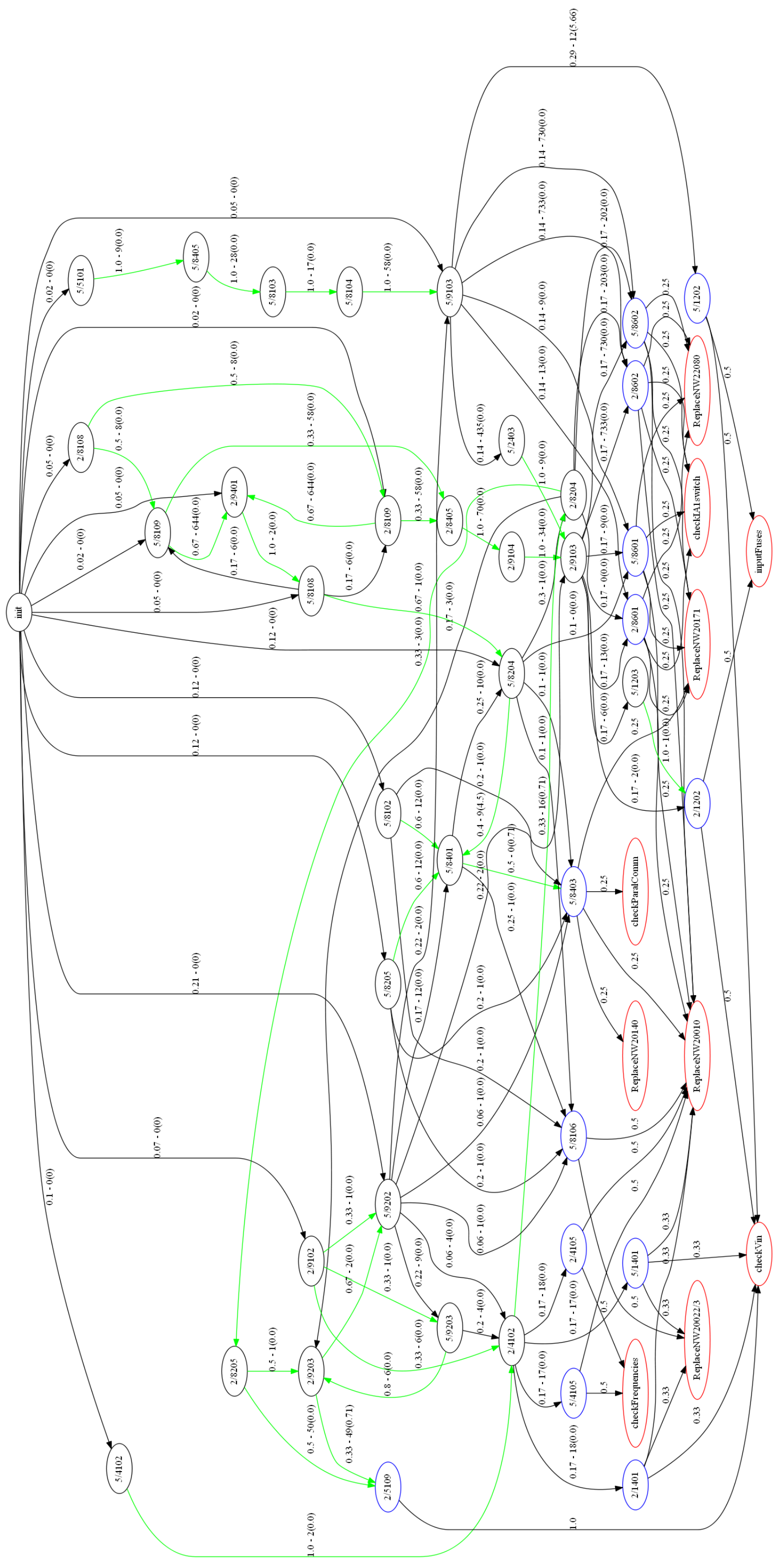


Figure A.1: Model 1.

A.2 Model 2

1. **Date :**10/05/2017
2. **Device:** 2,5
3. ΔT**:** 800
4. N**:** 5
5. **Representation:** $id/code$

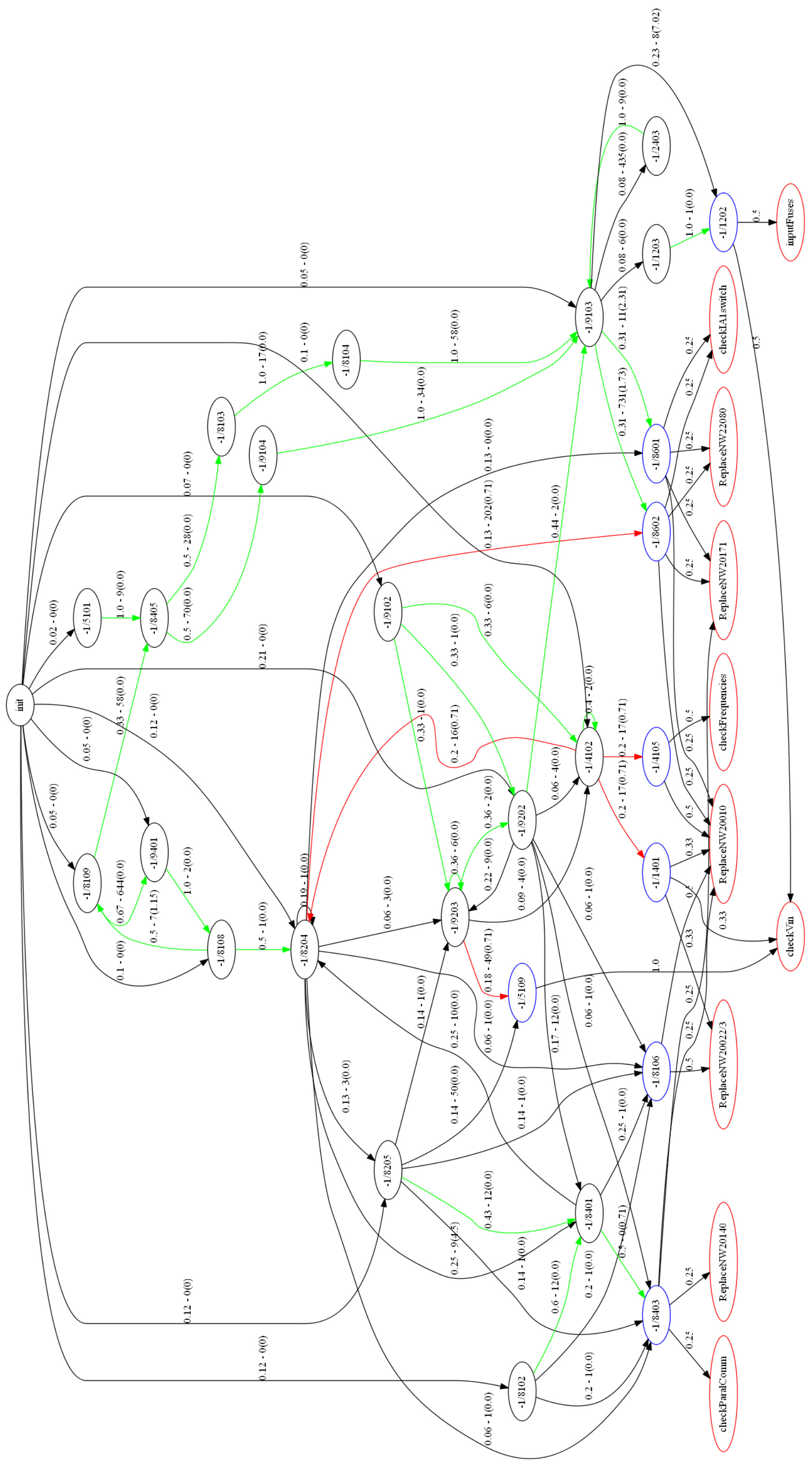


Figure A.2: Model 2.

A.3 Model 3

1. **Date :**11/05/2017
2. **Device:** 2
3. ΔT**:** 1000
4. N**:** 4
5. **Representation:** id/code

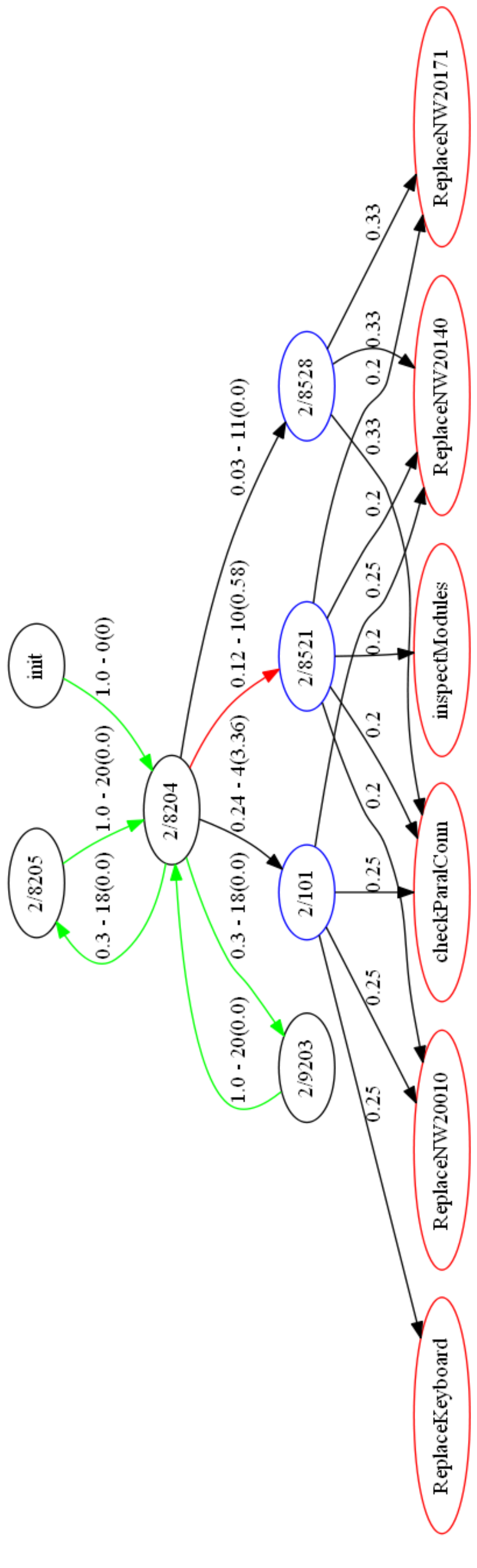


Figure A.3: Model 3.

A.4 Model 4

1. **Date :**11/05/2017
2. **Device:** 2
3. **ΔT:** 1000
4. **N:** 4
5. **Representation:** level**id/code

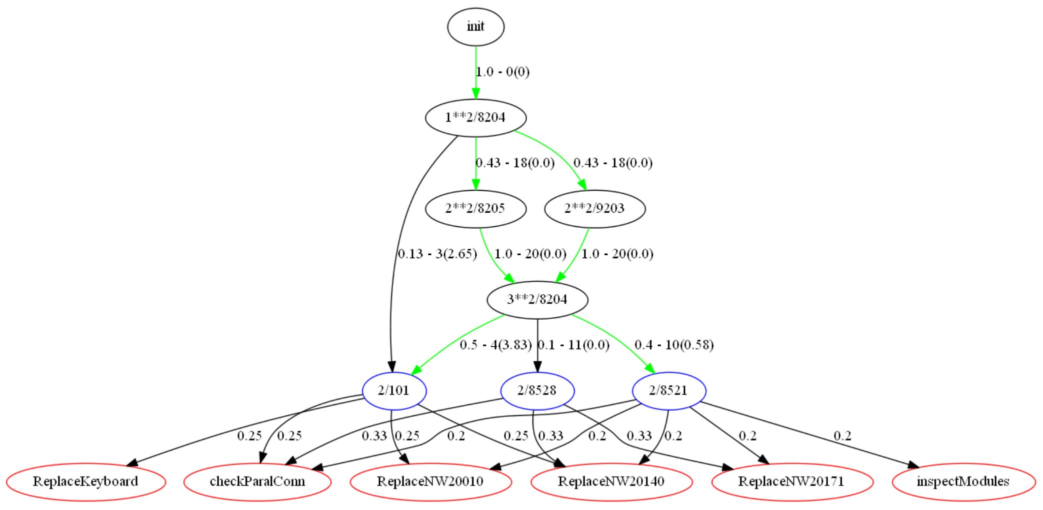


Figure A.4: Model 4.

A.5 Model 5

1. **Date :**11/05/2017
2. **Device:** 2,5
3. **ΔT:** 1000
4. **N:** 4
5. **Representation:** $Level**id/code$

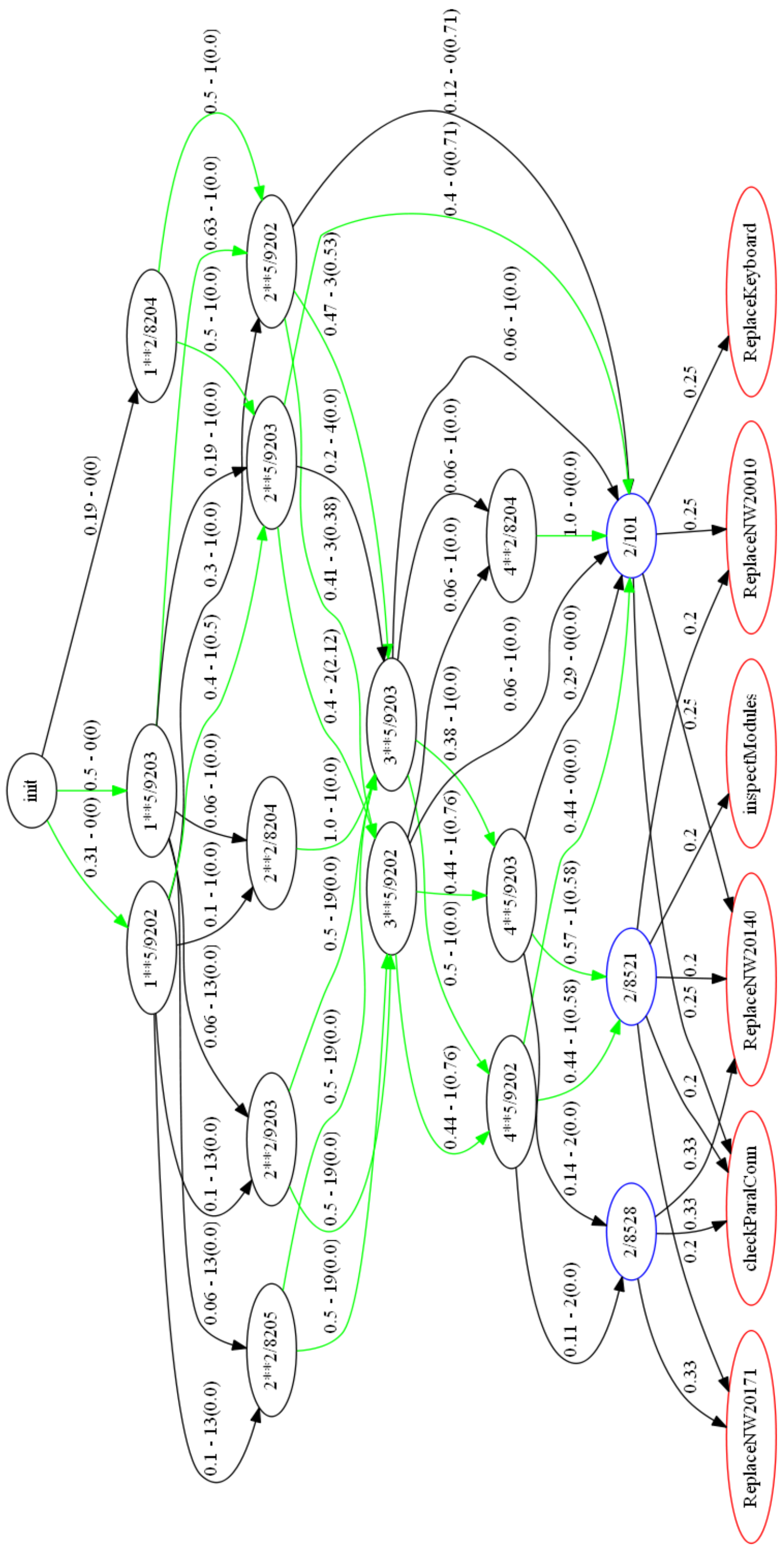


Figure A.5: Model 5.

A.6 Model 6

1. **Date :**11/05/2017
2. **Device:** 2,5
3. ΔT**:** 1000
4. N**:** 10
5. **Representation:** $Level **id/code$

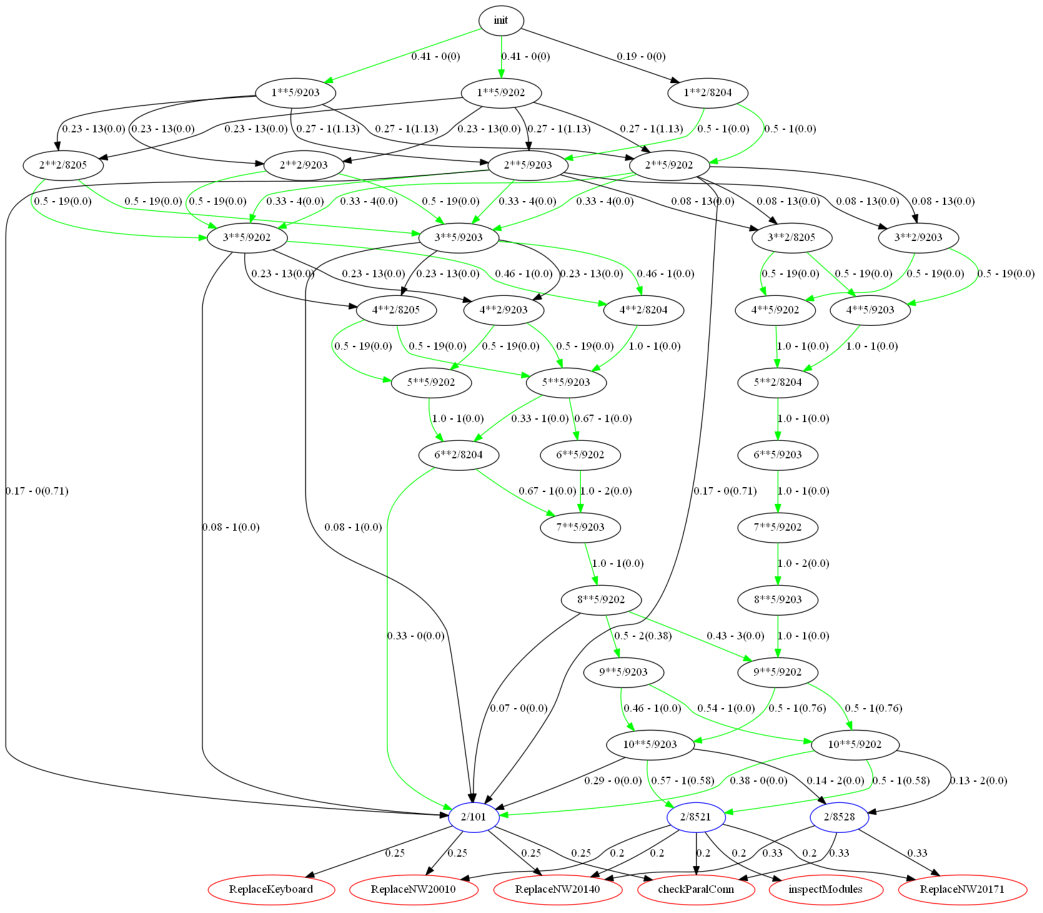


Figure A.6: Model 6.

A.7 Model 7

1. **Date :**10/05/2017
2. **Device:** 2
3. ΔT**:** 150
4. **Chains Length:** 5
5. **Representation:** $Level**id/code$

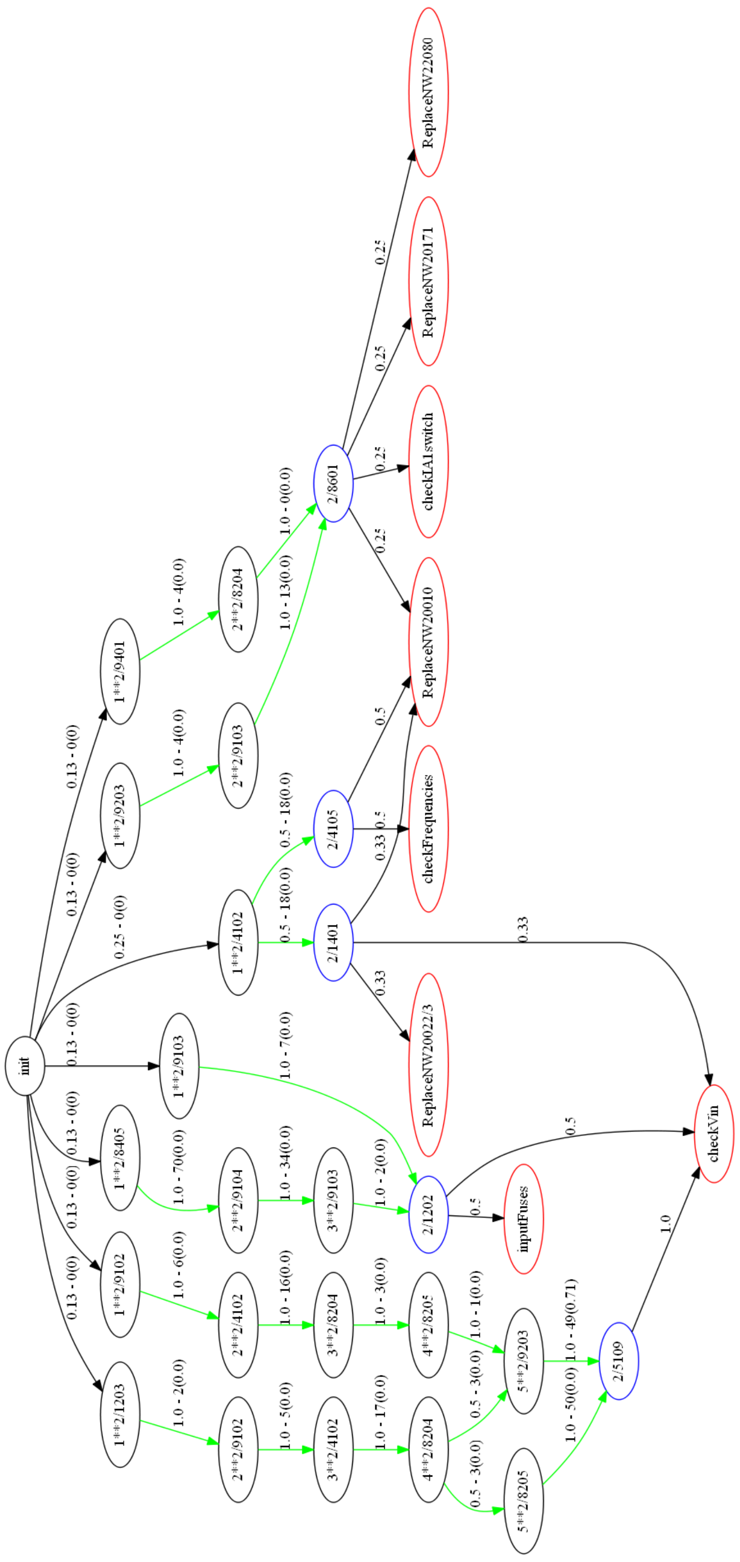


Figure A.7: Model 7.

A.8 Model 8

1. **Date :**10/05/2017
2. **Device:** 2
3. ΔT**:** 2500
4. **Chains Length:** 5
5. **Representation:** $Level**id/code$

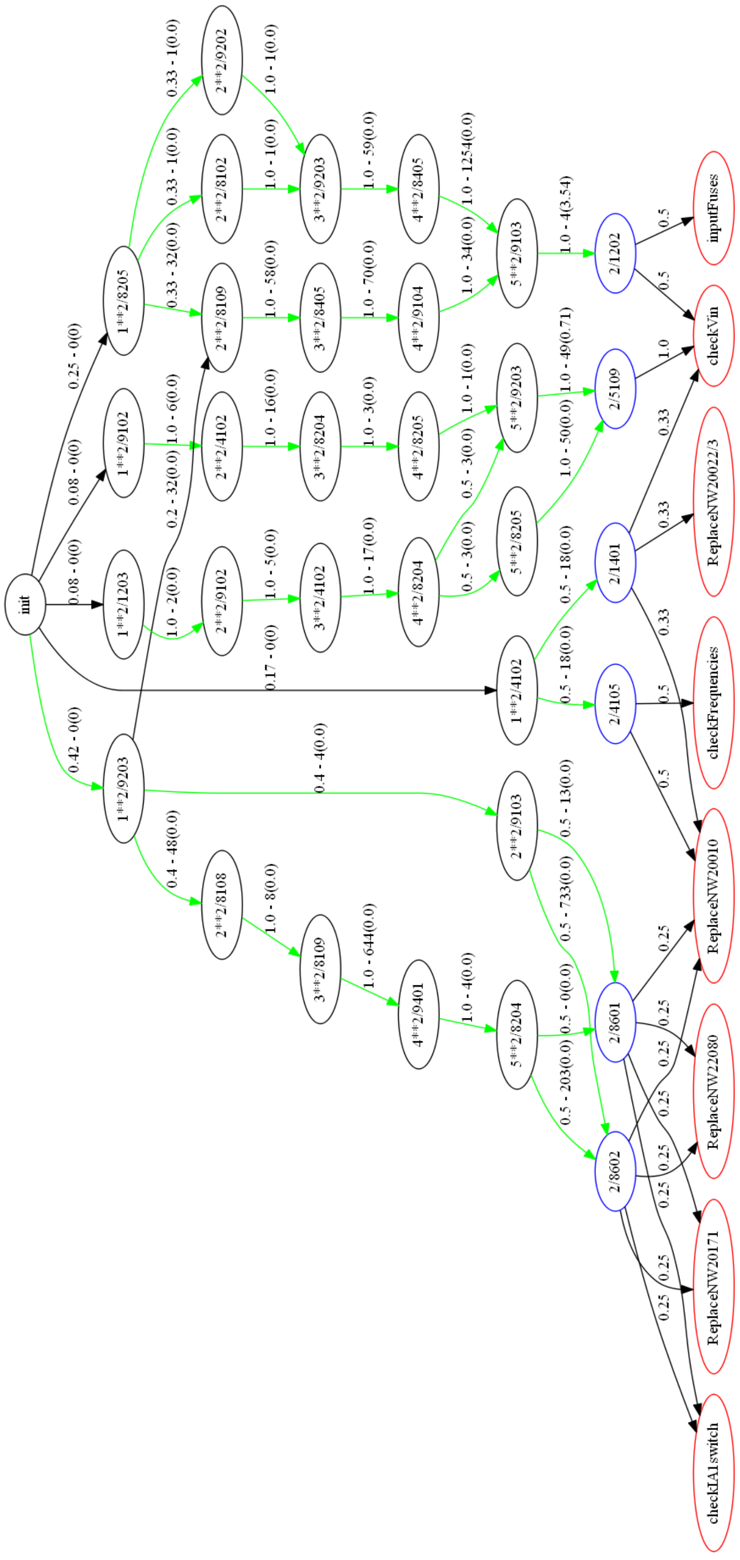


Figure A.8: Model 8.

Appendix B

MakeModel Method

In this Chapter we provide the implementation of the *MakeModel* method of the controller. This method is called by the view after the user presses the *submit* button.

The functions starts by calling the *selectDataFromDays* method which selects from the dataset only the events contained in the *selectedDays* parameter. After that, depending on the representation, the function removes the device number from the dataset.

```
def makeModel(self):
    daysToConsider = self.selectDataFromDays(self.df,
     ↪  self.data.selectedDays)
    if self.data.resetDeviceNumber == 1:
        daysToConsider = removeDev(daysToConsider)
```

The next step performed by the *Makemodel* method consists of creating the couples. For doing this operation we use the *sequenceMiningByTime* function whose has the purpose of extracting the chains and creating the couples based on the ΔT and the chain length n contained in the model of the application. The *sequenceMiningByTime* function handles all the problems highlited during the thesis.

```
sequenceMiningResult =
↪  self.sequenceMiningByTime(daysToConsider,
↪  self.data.deltaTime, self.data.n)
```

The following step consists of calculating the mean time and the standard deviation of each couple by using the method *addMeanTimeAndStdDev*.

```
meanAndStd = addMeanTimeAndStdDev(sequenceMiningResult)
```

After calculating the mean and standard deviation of each transition, we add to the set of couples the *init* state and we compute the probability.

```
sequenceWithInitState = createInitState(sequenceMiningResult)
probFGivenT =
↪  computeProbTargetGivenFeature(sequenceWithInitState[['feature',
↪  'target']])
```

The next step performed by the *MakeModel* consist of extracting a subset of couples from the total set of all the maintenance action by considering only the alarms contained in the *sequenceMiningResult*. On these couples we calculates the probability by using the same method as before.

```
sequenceWithMaintenance =
↪  self.createMaintenance(sequenceMiningResult)
probMaintenanceGivenAlarm =
↪  computeProbTargetGivenFeature(sequenceWithMaintenance[['target',
↪  'upscale']])
```

We are now able to compute the graph and save it. The function *createGraph* builds the graph and the function *drawGraph* is used to write the image of the graph on the hard disk. After that, the function write the parameters of the model in a ".txt" file with the same name as the graph. Another file created by the function is called "tmatrix.csv" and it contains the transition matrix computed by the application.

```
G = self.createGraph(meanAndStd, probFGivenT,
↪  probMaintenanceGivenAlarm)
filename = self.drawGraph(G)

filepointer = open('images/' + filename + '.txt', "w")
filepointer.write(self.data.getString())
filepointer.close()
probFGivenT.to_csv('trmatrix.csv')
```

The function returns to the view the name of the file created.

```
return filename + '.png'
```