Ricardo Miguel Nunes Camacho de Matos

Licenciado em Ciências da Engenharia Electrotécnica e de Computadores

Sequential Protocols' Behaviour Analysis

Dissertação para obtenção do Grau de Mestre em

Engenharia Electrotécnica e de Computadores

Orientador: Rodolfo Alexandre Duarte Oliveira,
Professor Auxiliar com Agregação,
Universidade Nova de Lisboa

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Setembro, 2018

Sequential Protocols' Behaviour Analysis

Este documento foi gerado utilizando o processador (pdf)LaTeX, com base no template "novathesis" [1] desenvolvido no Dep. Informática da FCT-NOVA [2].
[1] https://github.com/joaomlourenco/novathesis [2] http://www.di.fct.unl.pt

Para a minha família. Por tudo.

Acknowledgements

Começo por agradecer ao meu orientador, professor Rodolfo Oliveira por todo o tempo disponibilizado no meu acompanhamento ao longo deste trabalho, mesmo por vezes sobrecarregado, sempre se disponibilizou para qualquer esclarecimento, sem quaisquer restrições e a qualquer hora, tendo sido quem mais impulsionou a conclusão desta dissertação.
Seguidamente, gostaria de agradecer à FCT-UNL e ao DEE pelas condições proporcionadas que me permitiram desenvolver as capacidades necessárias para poder concluir os meus estudos de forma a me tornar num bom profissional. Também agradeço ao Instituto de Telecomunicações que apoiou o desenvolvimento do trabalho através dos projectos UID/EEA/50008 /2013 e CoShare (com referência LISBOA-01 -0145 -FEDER-030709).
Quero ainda agradecer aos meus pais e aos meus avós, por me terem proporcionado todas as condições necessárias para concluir esta etapa tão importante da minha vida. Garantiram sempre que nunca nada me faltasse, apoiando-me sempre incondicionalmente nas minhas decisões e por isso um obrigado muito especial por tudo. À minha irmã quero agradecer por toda a paciência e apoio que me tem fornecido ao longo deste tempo, assim como à minha namorada por toda a motivação, amor e carinho prestado ao longo desta caminhada. Aos meus restantes familiares mais próximos, por estarem presentes na minha vida e me apoiarem também neste meu trajeto.
Para finalizar, resta-me agradecer a todos os meus colegas de curso que me auxiliaram ao longo destes 5 anos e que marcaram esta minha jornada académica, ficando a vossa amizade para sempre no meu coração.

Abstract

The growing adoption of the Session Initiation Protocol (SIP) has motivated the development of tools capable of detecting valid SIP dialogues, in order to potentially identify behavioural traits of the protocol. This thesis serves as a starting point for characterising SIP dialogues, in terms of distinct signalling sequences, and providing a reliable classification of SIP sequences. We start by analysing sequential pattern mining algorithms in an off-line manner, providing valuable statistical information regarding the SIP sequences. In this analysis some classical Sequential Pattern Mining algorithms are evaluated, to gather insights on resource consumption and computation time. The results of the analysis lead to the identification of every possible combinations of a given SIP sequence in a fast manner.

In the second stage of this work we study different stochastic tools to classify the SIP dialogues according to the observed SIP messages. Deviations to previously observed SIP dialogues are also identified. Some experimental results are presented, which adopt the Hidden Markov Model jointly used with the Viterbi algorithm to classify multiple SIP messages that are observed sequentially. The experimental tests include a stochastic dynamic evaluation, and the assessment of the stochastic similarity. The goal of these tests is to show the reliability and robustness of the algorithms adopted to classify the incoming SIP sequences, and thus characterizing the SIP dialogues.

Keywords: Data mining, Sequential Pattern Mining, Session Initiation Protocol, Stochastic Classification

Resumo

A crescente adoção do Protocolo de Iniciação de Sessão (SIP) motivou o desenvolvimento de ferramentas capazes de detetar diálogos válidos de SIP, com o propósito de identificar características comportamentais do protocolo. Esta tese serve como ponto de partida para caracterizar os diálogos SIP, em termos de sequências de sinalização distintas e fornecendo uma classificação fidedigna das sequências SIP. Começamos por analisar algoritmos de identificação de padrões sequenciais de uma maneira off-line, os quais possibilitam a obtenção de informações estatísticas sobre as sequências SIP. Nesta análise, alguns algoritmos clássicos de Identificação de Padrões Sequenciais são avaliados para obter informações sobre o consumo de recursos e o tempo de computação. Os resultados da análise levam à identificação de todas as combinações possíveis de uma determinada sequência SIP de uma maneira rápida.

Na segunda etapa deste trabalho estudam-se diferentes ferramentas estocásticas para classificar os diálogos SIP de acordo com as mensagens SIP observadas. Também são identificados desvios aos diálogos SIP previamente observados. São apresentados alguns resultados experimentais, os quais adotam o Hidden Markov Model usado conjuntamente com o algoritmo Viterbi para classificar múltiplas mensagens SIP que são observadas sequencialmente. Os testes experimentais incluem uma avaliação dinâmica estocástica e a avaliação da similaridade estocástica. O objetivo destes testes é mostrar a fiabilidade e robustez dos algoritmos adotados para classificar as sequências SIP de entrada e, assim, caracterizar os diálogos SIP.

Palavras-chave: Identificação de Padrões, Identificação de Padrões Sequenciais, Protocolo de Iniciação de Sessão, Classificação Estocástica

Contents

List of Figures

List of Tables

Listings

Chapter 1

Introduction

Over recent years the need to extract and organize information has become essential, and has been heavily demanded in today's market. With the increasing number of smart devices that can communicate with each other, the size of databases has increased exponentially and the amount of information to process and analyse has simply become too much to handle.
Keeping in mind the amount of time spent analysing information and the performance of the available resources, data-mining has gained an important role in data analysis, with a plethora of algorithms capable of running these tasks autonomously .
In telecommunications the problem of data gathering and analysis has also become relevant. The amount of data and signalling transmitted between different protocols in a given period of time can be hard to analyse and, more importantly, hard to identify patterns that could be relevant to better understand the current operational conditions. Consequently, there is a need to find a reliable approach to analyse the behaviours of different protocols and take conclusions about the obtained results from the data that is available.

1.1 Motivation

This thesis is motivated by the need of analytical tools to diagnose and characterize the behaviour of the protocols currently adopted in telecommunication networks. Another motivation comes from the advantages that machine learning techniques provide in supervising signalling from multimedia communication sessions. This work is mainly considering the SIP protocol that is used to initiate and control communication sessions that support multiple multimedia services such as VoIP and IMS. The growing adoption of this protocol is associated with the need of implementing analytical tools capable of analysing SIP signalling messages. The goal is to predict the behaviour of the SIP protocol, having the intent of detecting anomalies in the communication sessions. The motivations take into account three major cornerstones:

- **Technical Diagnosis**: With the rise in the volume of data, the time used in diagnosis has become too extensive, consuming many people and resources in order to fulfil a specific analysis. The goal here is to make diagnosis faster and more accurate, making problem resolution simpler. For instance, in a specific network if packet drop rate increases we want to know what caused the increase as quickly as possible in order to solve the problem before it escalates into something more serious.
  Big Data algorithms can have an important role in selecting important data, since it can label sets of data into different groups, therefore making it easier to analyse in detail each of these individual sets, and can give additional information that otherwise could be inaccessible in naked eye.

- **Planning**: Given that observing a series of raw data can be very complex and time consuming, predicting what will happen in the future to the system according with the known data may be even more complicated. This thesis aims to aid in the decision making of plans and procedures in order to identify in an early stage what is happening in the system and if that behaviour will later affect the performance or damage the system as a whole. This may be useful in multiple scenarios. Imagine several packet drops are observed, and by analysing the pattern of the messages it is concluded that if the system keeps going on that direction it will eventually crash, within a week. With the aid of a data mining analysis it is possible to take additional decisions to avoid the crash. This can be achieved by planning the system with parameters that will directly affect the future outcome, therefore preventing the crash.

- **Security**: This thesis also intends to improve the security of a given system and prevent external intrusion to information that it may store in a database. This is accomplished again by analysing the behaviour of the system, and finding relations in the patterns that are extracted. If we get a frequent pattern that is deviating from the normal behaviour of the system, we can conclude that something is wrong and that an attack may be in course and it is needed to act on it. By making this verification we are ensuring the integrity and security of our system.

These three points are the major motivations for this thesis, but besides them we can also aim to hit the market with a solution to these problems, since protocol behavioural analysis in telecommunications is an area that has not been deeply researched. Additionally, it is vital to maintain a system functioning with little to no errors and safe from attacks, and this can only be done by analysing permanently how the system is behaving.

1.1.1 Objectives and Contributions

This subsection describes the objectives that were defined for this thesis as well as a few contributions of this work.

Objectives:

- Elaborate a bibliographic survey on different algorithms that are able to categorize large amounts of data, and extract useful information and knowledge from them. The goal here is to study which algorithms can be used to classify a series of raw data, so the analysis becomes simpler, by extracting information in form of patterns that can predict the behaviour of the system.
- Propose a data categorization framework with low computational complexity, in order to process faster.
- Be efficient and accurate in categorizing the data. To do so will require an extensive analysis on the selected algorithms for the pattern extraction to identify the most efficient ones and use only those which ensure better results.
- Apply classic algorithms to innovative scenarios in telecommunications. Here the algorithms chosen that meet the above requirements will be applied to analyse telecommunications Protocol Behaviours. To do so requires the utilization of sequential pattern mining algorithms and try to incorporate their functionalities and use them in the data selected from message logs provided by these protocols.

Expected Contributions:

- Implement different sequential pattern mining algorithms in the context of computer networks. These algorithms will be used to find frequent patterns in telecommunications protocols, such as SIP and GSM signalling.
- Classify protocol behaviours as 'normal' behaviours or as 'deviations' from their normal state. The behaviour analysis can have many different applications such as protocol's security, by analysing how a certain protocol is expected to behave, or anomaly detection.
- Identify in real time when a deviation of behaviour occurs, always with the aim of understanding why it happened in the first place and where to act to correct it.

1.2 Thesis Structure

This thesis is organized as follows. Chapter 2 outlines the related research in sequential pattern mining, where the basic notions of data mining are introduced, as well as a description of the most used sequential pattern mining algorithms. Chapter 3 reports the performance results obtained with different sequential pattern mining algorithms. Chapter 4 introduces a theoretical framework to identify SIP sequences and to measure its similarity to a known dataset of sequences. Finally, Chapter 5 presents experimental results obtained with the framework presented in Chapter 4. Chapter 6 ends the thesis by summarizing a few concluding remarks and insights for future research.

Chapter 2

Related Work

2.1 Introduction

Sequential pattern mining (SPM) was first introduced by Agrawal and Srikant [1]. In order to understand how this technique works lets start by showing a practical example. Assuming the 4 sequences: <(ab)>, <(abc)>, <(abc)d> and <(abc)dc>, the goal is to discover which of these sequences (or their subsets) are frequent. To accomplish this, it is necessary to determine the frequency with which items belong to each sequence. A minimum frequency, refered to as minimum support, is usually considered. Looking again at the sequences, and assuming a minimum support of 3, we can classify the sequence <(abc)> as frequent since it appears three times, in <(abc)>, <(abc)d> and <(abc)dc>. The same can be said of <(ab)>, which is present in all four sequences. On the other hand, considering the sequence <(abc)d>, we observe that it reaches the minimum support threshold, since it does not verify the minimum support equal to 3, appearing only in the last two sequences.

In a more formal way, sequential pattern mining is defined by several other authors [22][30] as a data-mining technique that discovers sequences of frequent item-sets. These sequences can be seen as specific patterns in a sequence database, and the goal of sequential pattern mining is to extract those sequential patterns whose support exceeds a pre-defined minimum support. All the items in one item-set must have the same time stamp or happen within a certain pre-defined window of time in order to be associated to a sequence. A typical example of a sequential pattern is "*Customers who buy a digital camera are likely to buy a color printer within a month*".

Sequential pattern mining has become a very popular data-mining technique with a vast range of applications, such as customer purchase patterning analysis, correlation analysis, and software bug tracking or in the case we present in this thesis, protocol behaviour analysis. To better introduce SPM we first need to learn some core concepts of this data-mining technique.

2.1.1 Types of Data

There are many types of data that can be mined with mining techniques. Because different data is available, the knowledge extracted will also be different, which is very useful to take different conclusions about the problem we are trying to understand.
Essentially we can categorize data into two different types: **Qualitative**, and **Quantitative**. In qualitative type data information is characterized by a symbolic value, because their attributes cannot be measured. An example of qualitative data are colors. If we want to describe a color we can define it as a code (e.g. Black #000000). By doing this, we are transforming colors into *categorical variables*, where we assign a number that without a context has no meaning at all, but grouped in the category of numbers we can understand what they mean. However, in this type of data it does not make sense to determine a sum of colors or an average, since it has no meaning.

On the other hand, quantitative data can be quantified and verified, being used in statistical manipulation. An example of such data can be seen looking at the example of a shopping cart, where we can define the price of the items and apply arithmetic. In short we can say that quantitative data defines, whereas qualitative data describes. With that in mind we can classify data-mining in different categories acording to the data they are applied to as defined in [30].

- **Relational database** – Here data is described by a set of attributes which are stored in tables, being a highly structured data repository. Data mining in these relational databases focuses mainly on discovering patterns and trends, being used in banks and in supermarkets to acquire customer behavioural information, such as customer spending habits, and profiling clients into categories such as high and low spenders.

- **Transactional database** – Refers to a collection of transactions (e.g. sale records), where data-mining enters to find a correlation between items and transaction records. These types of databases have gained popularity due to the increase in e-commerce, with a fast growing community of people that start to make all of their shopping online.

- **Spatial database** – Contains traditional data as well as geographic information of the corresponding data. It can be used to associate a certain geographic area with a specific behaviour.

- **Temporal and time-series database** – For each temporal data item, the corresponding time is associated. Can extract more useful information from an individual, by analysing it's temporal behaviour.

Looking at the problem that this thesis aims to solve, we can define our initial database as a *time-series*, since in telecommunications when a protocol is running the packets sent and received by a terminal are logged into several files across the network with all the important meta data associated to that instance such as ID, type of message, source, destination and finally a time-stamp.

Time stamp is an important attribute to any dataset and, consequentially, important to data mining since it can add more knowledge to the process of extracting information. For that reason, sequential pattern mining has an important role in these types of databases, since it is used to find the relationships between occurrences of sequential events, although it is used in a special case of time series databases known as Sequential databases that can come whit or without a concrete notion of time.

2.1.2 Types of Mining

In the context of data mining it is possible to identify several types of mining, each of which applies different techniques to the data, extracting distinct knowledge. Data mining can be classified into two separate classes acording to the data they are applied to, defined in [30] as: descriptive mining that has the goal to summarize general properties of data, and prescriptive data that performs inference on current data to make predictions of what will come next.
Given these two classes of data mining we can identify several techniques, reported with a few examples.

Association Rules – First introduced in [3], an association rule aims to extract interesting correlations, frequent patterns and associations between sets of items in a database. In a more formal explanation let $L = I_1, I_2, ..., I_n$ be a set of items. An association rule means a relationship of the form $X => I_j$, being X some set of items from L and I_j a single item not belonging to X. The rule is satisfied if for a confidence factor 0<c<1 at least c% of the transactions in a database satisfy both X and I_j.

Now looking at a more tangible example let's assume we have the following statement: "70% of the people who buy running shoes also buy a sports water bottle" Here we can observe that the antecedent of the rule is "running shoes", the consequent "sports water bottle" and finally the confidence factor is set at 70%. Given this information a sports outlet could rearrange it's stock and sales according to the customer habits. This type of mining technique is widely used in areas such as telecommunications.

Classification - Described in Han and Kamber (2000)[15], the objective of this technique is to automatically build a model that can classify a class of objects, in order to predict the classification of future objects. Data classification can be seen as a two-step process.
In the first step, a classifier is built to describe a set of predetermined data classes, being called the **learning step**. Here a learning algorithm is used to build the classifier by analysing a set of data called **training set**. In this collection of training data, the classes and concepts that they belong to are predefined and provided so we can also define this step as **supervised learning**. The final step is used to test the classification accuracy by using the model to classify other test examples. Since the classifiers tend to overfit the data, a **test set** is used to be classified by the model. The final accuracy will depend on the percentage of accurate classifications the model does to the test set.
An example of a classifier could be a model to categorize bank loan applications as *safe* or *risky*, or catalogue a car as *Sports car* or *Family van*.

Clustering - Contrary to the classification method, clustering uses an **unsupervised learning** process, which means that there are no predefined classes to the initial data set (training set) which is very common in large databases, because assigning classes to every single object can become quite costly. As defined by Han and Kamber (2000) [15] clustering is the process of grouping a set of physical or abstract objects (data) into classes of similar objects also known as **clusters**. Objects within a same cluster have high similarity, but are very dissimilar to objects in other clusters. **Similarities** are defined by functions, usually quantitatively specified as distances or other measures defined by domain experts.
Clustering applications are manly used in market segmentation, that categorize customers into different groups so that organizations can provide personalized services to each group.

Regression analysis - This method is the most widley used aproach for *numeric prediction*, which is the the task of predicting continuous (or ordered) values for a given input [15]. Regression involves learning a function that maps a data item to a real-valued prediction variable [10], and can be used to model the relationship between one or more *independent/predictor* variables and a *dependent/response* variable. In the context of data mining, the predictor variables are the attributes of interest describing the tuple (i.e., making up the attribute vector). Generally, the values of the predictor variables are known, being the response variables predicted by these methods.
Regression has a vast range of applications focusing on prediction. The applications of regression analysis can range from Buiseness such as predicting customer demands for a new product or in medicine by estimating the probability that a patient will survive given the results of a set of diagnostic tests [10]. Many regression-based methods exist

such as Poisson regression, log-linear models and non-linear regression, although most of the problems can be solved simply by using linear regression.

Summarization - Consists in methods used to find a compact description for a subset of data[10]. This type of techniques are often used in interactive exploratory data analysis and automated report generation. Data summarization is very important in data preprocessing and is used for measuring the central tendency of data (e.g. mean, median and mode), measuring the dispersion of data (e.g. range, variance and standard deviation) and graphical representations such as histograms and scatter-plots are also useful to facilitate the visual inspection of the data [15].

Anomaly detection - This method builds models of normal network behaviour defined as *profiles* which are then used to detect the presence of new patterns that significantly deviate from the generated profiles [15]. These deviations may represent actual intrusions in the system or simply new behaviours that are not included in the profile. The advantage of this technique stands on the fact that it can detect novel intrusions that have not yet been observed, a human analyst needs to sort through all the deviations to ascertain which ones are real intrusions. A limitation of this method is the high percentage of false positives, which can be somewhat corrected by adding the new behaviours of the system to the profile. The new patterns of intrusion can be added to the set of signatures for misuse detection.

All these mining techniques are still used until this day and remain effective depending on the paradigm they are inserted into, having many research papers dedicated to study each of these techniques in detail [30]. The next section of this chapter will bring more light to the case in study in this thesis, sequential pattern mining.

2.2 Introduction to Sequential Pattern Mining

2.2.1 Problem definition

This section defines the problem of mining sequential patterns, and also give some insights on the concepts used in this area. The basis for this definition was taken from [22], and can be viewed as follows. Given:

(i) A set of sequential records (sequences) representing a sequential database D.

(ii) A minimum support threshold.

(iii) A set of k unique events or items $I = i_1, i_2, \dots, i_k$.

The problem of sequential pattern mining consists on finding the set of all frequent sequences S, in the given database that supports the minimum threshold (minimum support), in order to describe the data or predict future data. In order to better understand this problem we introduce different definitions that represent the roots of different techniques capable of mining sequential patterns.

Definition 1 *Let $I = I_1, I_2, I_3, ..., I_n$ be the set of all **items**. A **item-set** can be viewed as a non-empty, unordered collection of items.*

The item-set is denoted (x_l x_2 ... x_q), where x_k is an item. The brackets are omitted if an item-set has only one item, that is, (x) is written simply as x. An example of an item-set can be seen in a grocery store, where the items can be represented by a list of products bought at the same time such as bread, milk and cheese, represented by (abc). Note that the items all belong in the same event since they were all bought at the same transactional time.

Definition 2 *A **sequence** is a lexicographically ordered list of item-sets (events), such as S = <e_1 e_2 e_3 ... e_l> where the order of the item-sets has an important role, in this case $e_1$occurs before e_2, which occurs before e_3, and so on. The event e_j is also called an element of S [15].*

Let S be represented by <(ab)b(bc)c(cd)>. An item can occur only once in a given item-set, but it can have multiple instances in different item-sets of a sequence. In sequence S, the item b has three instances, appearing in the item-sets (bc), (b) and in (ab). However, if we were to add the item-set (dcd) to the sequence S it would not be possible, since it contains two instances of the same item.

Definition 3 *Given a sequence S, its **support**, denoted as $\sigma(S)$, can be defined as "The total number of sequences of which S is a subsequence divided by the total number of sequences in the database D", whereas the **support count** of a sequence S is the total number of sequences in D of which S is a subsequence.*

A sequence can only have the classification of *frequent* if its support count (frequency) is greater or equal to the user-defined *minimum support*. An example of what a sequential database may look like is shown in table 2.1.

Table 2.1: Sequential database.

Sequence ID	**Sequence of items**
1	<a(abc)(ac)d(cf)>
2	<(ad)c(bc)(ae)>
3	<(ef)(ab)(df)cb>
4	<eg(af)cbc>

Looking at table 2.1 we can observe that the sequential database is composed of a set of tuples, <SID,S>, where SID is the Sequence ID, and S is the Sequence of items.

Definition 4 *A subsequence S' of S is denoted by S' $\subseteq$ S. Formally, a sequence a=<$a_1, a_2 ... a_n$> is a subsequence of b=<$b_1 b_2 ... b_m$>, and b is a supersequence of a, if there exist integers $1 \leq i_1 < i_2 < \ldots < i_n \leq m$ such that $a_1 \subseteq b_{i1}$, $a_2 \subseteq b_{i2}$, $\ldots$, $a_n \subseteq b_{in}$ [4].*

Finnally, we can have a formal definition of **Sequential pattern**, which is a sequence of itemsets that frequently occurred in a specific order. All items in the same itemsets are supposed to have the same transaction-time value or be included within a time gap [30], which will be defined as *max-gap*. Observing a client transaction sheet all of its transactions can be viewed together as a sequence, where each individual transaction can be represented as an item-set of that sequence, that are ordered in a certain transaction-time.

2.2.2 Taxonomy

This thesis adopts the taxonomy defined in [22], where sequential pattern mining algorithms are categorized in three main categories: *Apriori-based*, *Pattern-growth* and *Early-prunning*. This taxonomy is used to classify sequential pattern mining algorithms based on the characteristics that the different techniques provide. It also helps to better understand and distinguish these techniques before-hand, having some previous knowledge of the characteristics and capabilities of a given algorithm, by looking at what category it belongs.
This section gives examples in some of the key characteristics from each of the categories mentioned above. The focus is on the characteristics that are intrinsic to the algorithms, which include, SPADE[29], GSP[26] and PrefixSpan[17].

Apriori-based: Algorithms that fit into this category depend mainly in the apriori property which states that *"All nonempty subsets of a frequent itemset must also be frequent"* [22]. Besides that apriori algorithms are also described as being *antimonotonic* or *downward-closed*, which means that if a given sequence cannot pass a pre-defined minimum support test, all of its supersequences will also fail the test, and consequently will be discarded as candidates of frequent sequences. There are three key characteristics that are used in apriori-based methods which are discussed bellow:

1. **Breadth-first search:** This characteristic is used to describe apriori-based algorithms because all of the k-sequences are constructed together in each kth iteration of the algorithm as it travels trough the data. So suppose the goal is to construct the length-3 sequences. In this method, all of the length-2 sequences must be solved first in order to proceed. This makes available a lot of information that can be used in pruning, but comes with a trade-off between information availability and memory consumption.

2. **Generate-and-test:** Introduced by the Apriori algorithm in [3], uses exhaustive join operators, where the pattern continues growing one item at a time, being always tested against the minimum support to ensure the given sequence is frequent. This causes the generation of an enormous number of candidate sequences, that can consume a lot of memory in the early stages.

3. **Multiple scans of the database:** As the name suggests, this characteristic is based on scanning the original database every time it needs to ascertain if the generated candidate sequences are frequent. This scanning keeps on going as long as there exists candidate sequences that need to be verified. This method is very costly in terms of processing time and I/O. A solution to this limitation comes with the introduction of the next category.

Pattern-Growth: Algorithms that fit this category usually display better performance than apriori-based, both in processing time and in memory consumption. This method was implemented to serve as a solution to the problem of *generate-and-test* that was present in the early apriori algorithms. The goal of this technique is to avoid the candidate generation step altogether, and the search is only done in a restricted portion of the initial database. In order to fulfil the needs of pattern-growth algorithms different key characteristics must be considered, including:

1. **Candidate sequence pruning:** Pattern-growth algorithms like PrefixSpan and others, attempt to use a data structure that allows pruning of candidate sequences in the early stages of mining. Algorithms that do so have a smaller search space, which in return results in a more directed and narrower search procedure. This characteristic is also present in early-pruning techniques, that will be discussed later in this section. The difference is that early-pruning algorithms try to predict future candidate sequences, in an effort to avoid generating them in the first place.

2. **Search space partitioning:** This characteristic can be seen in some apriori-based algorithms like SPADE, in spite of being a characteristic of pattern-growth. As the name suggests it provides partitioning of the search space generated, being mostly used in large sets of candidate sequences. This characteristic plays an important role in pattern-growth algorithms, since it is used by nearly everyone of them, with the goal to generate as few candidate sequences as possible. This in extent provides an efficient memory management. There are different ways to partition the search space. SPADE represents the search space as a *lattice* structure which will be discussed in more detail in later sections. Others such as WAP-mine[24] use a tree-projection structure (this algorithm will not be covered in this thesis).

3. **Suffix/Prefix growth:** Used in algorithms that are dependent on projected databases (PrefixSpan), it first starts by finding the frequent k-sequences and then chooses

each frequent item as a prefix or a suffix, after that the candidate sequences are built around these items and are mined recursively. This is done taking in account that frequent subsequences can always be found by growing a frequent prefix/suffix. This characteristic greatly reduces the amount of memory required to store the different candidate sequences that share the same prefix/suffix.

4. **Depth-first traversal:** In this case each pattern is constructed separately, in a recursive way. All the subsequences are processed at each level before moving on to the next [28]. This characteristic uses less memory and less candidate sequence generation than breath-first search, having in extent a more directed search space. SPAM [5] is an algorithm that relies heavily in this characteristic. SPADE can also employ this method.

Early-Pruning: These types of algorithms are defined as using some sort of *position induction* to prune candidate sequences at a very early stage in mining. This technique used in the LAPIN algorithm as defined in [27], is used during the mining process and checks the items last position to see if it is smaller than the current prefix position. When this condition is verified, the item cannot appear behind the current prefix in the same customer sequence. By doing this, the support counting is avoided as much as possible and the generation of infrequent candidate sequences is kept at a minimum. Characteristics of this category are as follows:

1. **Support counting avoidance:** The idea behind this characteristic is to compute support of a candidate sequence without carrying a count throughout the mining process, and thus avoid scanning the sequence database each time it is needed to compute this support. The advantage of this characteristic is the removal of the sequence database from memory once it stores candidate sequences along with their corresponding support count.

2. **Vertical projection of the database:** This characteristic is also used in the SPADE algorithm [29] needing only to visit the sequence database once or twice, which then proceeds to construct a bitmap or position indication table for each frequent item, thus, obtaining a vertical layout of the database which is represented in tables 2.3 and 2.4. These tables are used in the mining process to generate candidate sequences and count their support. This characteristic has the advantage of reducing memory consumption, but on the other hand requires quite a lot computational power.

3. **Position-coded:** As refereed above, postion-coded (or position induction) is used in early-pruning algorithms to look-ahead and avoid generating infrequent candidate sequences, thus, avoiding the calculation of its support.

2.2.3 Structure definition

Being SPM a data-mining technique it has to follow the same structure as any data-mining algorithm would do. The operation cycle of data mining is divided in 3 stages [30]:

- **Preprocessing** - It is performed before any data-mining techniques are put into work, having the goal of cleaning and selecting the data for use from a raw database.

- **Data mining process** - At this stage the different data mining techniques are used in order to extract information and knowledge from the selected data.

- **Postprocessing** - Evaluates the mining result according to the user requirements.

The work-flow of the previously mentioned processes begins with cleaning and integrating the initial databases where we will get our raw data. The need to do so arises from the fact that information may come from several databases with different names to define the same concept, or even in the same database some inconsistencies may be found with missing or duplicate information. Suppose we have a supermarket that needs to check the sales revenue from a certain item in its store and there are lots of brands on sale but all of them have the same base vending item. In this case, we need to group all sales of each brand together in order to have a notion of the big picture and get a general overview of the sales of the item to analyse.

Since not all of the data in the database is related to our mining task, after cleaning and integrating the data, the next step is to select the task related data. Taking the example of this thesis we shape the information in order to obtain a sequential database, with sequences from telecommunication protocols, which will leave our database not only smaller, but ready to be mined.

The second process involves the use of data mining techniques applied to the selected data in order to extract different information or different knowledge [30]. The range of information extracted may vary depending on the adopted algorithm. In the particular case of Sequential pattern mining the knowledge we acquire consists in frequent item-set patterns.

Finally, we evaluate the knowledge achieved with the mining algorithm in order to assert that our goals have been reached. The evaluation may depend on what the end result is expected to have and varies with the user requirements.

2.2.4 Architectures

Being Sequential pattern mining a particular case of data mining, we can describe its architecture in a typical data mining system. Data mining can have a broad view in functionality. It is described in [15] as the process of discovering interesting knowledge from large amounts of data stored in databases, data warehouses, or other information repositories. So a typical data mining system can have several components distributed across the different processes associated to this technique. Taking into account the structure defined for SPM we can look at the architecture as follows:

- **Pre-processing:** This step intends to treat the data to be mined, so it starts with a *Data Repository*. This can be either one or a set of databases/other kind of information repositories where the information to be analysed is going to be extracted. Usually, the data contained on these repositories may be in a "raw" state, meaning that before extracting information, data cleaning and integration techniques may need to be done. These techniques involve removing any noise or inconsistent data that can be found in the initial repositories, combining distinct data sources into a whole and begin the selection of the pieces of data that are relevant for our mining task.
  After the initial "cleanse", the information extracted comes in the form of *clean data*. The data extracted is then transformed to a format that is ready to be mined, by performing different techniques such as summary or aggregation operations.

- **Data mining process:** This step is where the user interacts with the system, by giving the initial parameters to start the sequential pattern mining. These parameters can vary from minimal supports, to the selection of the Algorithm used.

- **Data Post-processing:** After applying the SPM techniques on the selected data, some knowledge will come out of that mining, in the form of frequent sequential patterns. These results are then evaluated, asserting if the outcome is the desired. If the information extracted is insufficient or has no substantial value to the problem the system is trying to solve, the data mining process repeats until these final conditions are reached. In the repeating process the user inserted parameters are changed in order to obtain different results.

Figure 2.1 shows a summarized visual representation of the core steps that are present in any sequential pattern mining architecture.

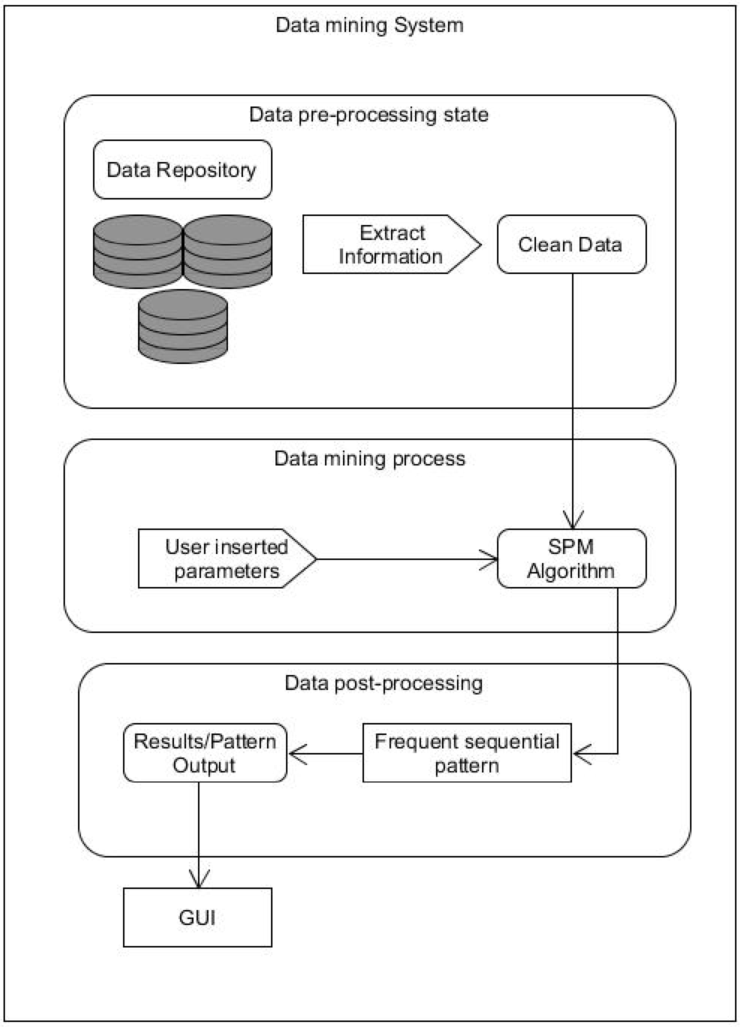


Figure 2.1: Sequential pattern mining Architecture

2.3 Sequential Pattern Mining Algorithms

This thesis will cover three of the most popular algorithms in sequential pattern mining, that possess the most bibliography available which will later be used to implement the protocol's behavioural analysis. The objective of this section is finding the differences between each of these implementations. It will include a brief overview of the selected algorithms, ending with several evaluation results capable of comparing their performance.

2.3.1 GSP

Generalized Sequential Patterns (GSP), proposed by Agrawal and Srikant (1996) [26], is categorized in the proposed taxonomy as an *apriori-based* algorithm. GSP uses the downward-closure property of sequential patterns and adopts a multiple-pass, candidate generate-and-test approach[18]. The first pass over the data determines the support of each item, which means it counts the number of occurrences of the item in the data-sequences. At the end of this first pass, the algorithm knows which items pass the minimum support test, and consequently determine which of them are frequent. This pass generates a seed set which represents the frequent sequences found, that are to be used on the next iteration of the algorithm. The seed set is used to generate candidate sequences that have one more item than the seed set, and consequently all candidate sequences will have the same number of items. The support of the candidate sequences is found during the pass over the data, being based on the minimum support. These candidate sequences are tested to determine if they are frequent or not. If they become frequent they will be used as the seed for the next candidate generation. The algorithm only stops when no frequent sequences nor candidate sequences are generated at the end of the pass.

GSP is described as having two sub-processes:

Sequence candidate generation: In this process candidate k-sequences are generated based on the set of all frequent (k-1)-sequences. First lets introduce the notion of a *contiguous subsequence* as it is presented in [26].

Definition 5 *Given a sequence S = <S_1 S_2...S_n> and a subsequence C, C is a contiguous subsequence of S if any of the following conditions hold:*

1. *C is derived from S by dropping an item from either S_1 or S_n.*
2. *C is derived from S by dropping an item from an element S_i which has at least 2 items.*
3. *C is a contiguous subsequence of C', and C' is a contiguous subsequence of S.*

For example, consider the sequence S = <(ab)(cd)(e)(f)>. The sequences <(b)(cd)(e)>, <(ab)(c)(e)(f)> and <(c)(e)> are some of the contiguous sequences of S. However <(ab)(cd)(f)> and <(a)(e)(f)> are not.

Candidates are generated in two steps, similarly to the candidate generation procedure for finding association rules described in [2].

- **Join Phase**- here a sequence S_1 joins another sequence S_2 both belonging to the seed set L_k, if the subsequence obtained by dropping the first item in S_1 is the same as the subsequence formed by dropping the last items in S_2. The generated candidate sequence by joining S_1 and S_2 is the sequence S_1 extended by the last item of S_2 (e.g., *abcd* is the result of joining *abc* with *bcd*).

- **Prunning Phase**- those candidate sequences that have a contiguous subsequence whose support count is less than the minimal support (infrequent) are deleted.

An example of this can be seen in Table 2.2. In the join phase, the sequence <(1, 2)(3)> joins with <(2)(3, 4)> to generate <(1, 2)(3, 4)> and with <(2)(3)(5)> to generate <(1,2)(3)(5)>. In other hand the sequence <(1, 2)(4)> does not join with any sequence since there is no sequence of the form <(2)(4, x)>. In the prune phase the sequence <(1,2)(3)(5)> is dropped since there is no support for its contiguous subsequence <(1)(3)(5)>.

Table 2.2: Candidate Generation Example.

Frequent sequences	**After join**	**After prunning**
<(1, 2)(3)>	<(1, 2)(3, 4)>	<(1, 2)(3, 4)>
<(1, 2)(4)>	<(1, 2)(3)(5)>	
<(1)(3, 4)>		
<(1, 3)(5)>		
<(2)(3, 4)>		
<(2)(3)(5)>		

Candidate support count: This final sub-process starts by using an adaptation of the hash-tree structure of [2] to reduce the number of candidates to check the support count. After that the **Contain Test** is introduced to check if a data sequence D contained a candidate sequence S, being composed of two phases:

- **Forward phase:** GSP finds successive elements of the candidate sequence S in the data sequence D as long as the difference between the end-time of the element just found and the start-time of the previous element is less than the user defined time constrain *max-gap*. If the difference is more than max-gap, the algorithm switches to the backward phase. If an element is not found, the data-sequence does not contain S.

- **Backward phase:** The algorithm backtracks and "pulls up" previous elements. Suppose the current element is represented by S_i and the end-time $(S_i)=t$. The algorithm checks if there exists transactions containing S_{i-1} and their transaction-times are after t-*max-gap*. Since after pulling up S_{i-1} the difference between S_{i-1} and S_{i-2} may not satisfy the gap constraints, the algorithm moves backwards until either the max-gap constraint between the element just pulled up and the previous element is satisfied, or the first element has been pulled up. Then the algorithm switches to the forward phase. If any element can not be pulled up the data-sequence D does not contain S.

2.3.2 SPADE

SPADE is an algorithm developed by Zaki (2001) [29], and it relies on the *Lattice Theory* defined in [7]. The goal is to find frequent sequences using efficient lattice search techniques and simple join operations. All the existing sequences are usually discovered with only two or three passes over the database. Besides that, SPADE utilizes combinatorial properties to decompose the original problem into smaller portions, which can be independently solved in main-memory. The sequential database is transformed into a vertical data format as in the example in Table 2.3, where the new database becomes a set of tuples of the form <*itemset* : (*sequence_ID*,*time_ID*)> [18] and each item id is associated with the corresponding sequence and time stamp. An example of the tuples present in the itemset <(a)> are (1,23), (6,6) and (21,2).

Table 2.3: Vertical database.

<(**a**)>	<(**b**)>	<(**c**)>	<(**d**)>	<(**e**)>
(SID , TID)	(SID , TID)	(SID , TID)	(SID , TID)	(SID , TID)
(1 , 23)	(2 , 36)	(1 , 26)	(6 , 12)	(6 , 23)
(6 , 6)	(7 , 18)			(21 , 26)
(21 , 2)				

SPADE can be seen as a three step method. Firstly, it scans the vertical database to compute frequent 1-sequences in an apriori-like way. For each database item, the id-list is read from the disk to memory, and then the id-list is scanned and the support is incremented for each new sequence ID (SID) found.

The next step involves finding the frequent 2-sequences. A vertical-to-horizontal transformation is preformed on the database. For each item i, its id-list is scanned into memory. We then look for each (SID,TID) pair abbreviated as (s, t). We group those items with the TID (*i*,*t)* and order in increase of SID, and the resulting database is shown in table 2.4.

Table 2.4: Vertical to Horizontal database.

SID	**(Item,TID) pair**
1	(a,23) (d,26)
2	(a,19) (b,36)
6	(a,6) (e,12) (g,23)
7	(b,18)
21	(a,2) (h,26)

In the final step the 2-sequences are used to construct the lattice, the only problem is that the lattice can grow very big and cannot fit in the main memory. In order to solve this problem, Zaki (2001) [29] proposed that the lattice can be decomposed into disjoint subsets called *equivalence classes*, by assigning sequences with the same prefix items to the same class. With this partition, each subset can be loaded and searched seperatly in main memory. The lattice is then traversed for support count and enumeration of frequent sequences[22]. There are two methods used to traverse the lattice, which are **breadth-first** (BF) and **depth-first** (DF) used in the SPAM algorithm [5].

In breadth-first search, the classes are generated in a recursive bottom-up manner [30] (e.g., to generate the 3-item sequences all the 2-item sequences have to be processed). Depth-first search only needs one 2-item sequence and a k-item sequence to generate a (k+1)-item sequence. Pruning is also applied in this third step, using the apriori property [2] when intersecting id-lists (temporal joins) and counting support[22]. Comparing the two methods, we can conclude that Depth-first search dramatically reduces computation and the number of id-lists generated, needing less memory. However, breadth-first search has the advantage of having more information to prune candidate k-item sequences.

2.3.3 PrefixSpan

PrefixSpan [17] is based on the general idea of examining only the prefix subsequences and project only their corresponding postfix subsequences into projected databases. In each projected database, sequential patterns are grown by exploring only local frequent patterns. This algorithm employs database projection in order to shorten the database on the next pass, which in return speeds up computation. In this method, there is no need for candidate generation, since it recursively projects the database according to their prefix.

To illustrate the idea of projected databases, consider <a>, <(ac)> and <(ac)h> from the sequential database in Table 2.5a. These are *prefixes* of the sequence <(ac)h(efg)>. On the other hand we can call <(_c)h(efg)> the *postfix* of the sequence <a>. An example of a Projected Database, assuming a support count of 2, can be seen in Table 2.5b, containing a portion of all the lenght-1 sequences.

There are three different projection methods that can be used in PrefixSpan, which are *level-by-level*, *bi-level* and finally *pseudo projection* [30]. The first method presented acts in three steps:

1. Scan the sequential database to obtain the length-1 sequences.
2. Divide the search space into different partitions, where each partition corresponds to the projection of the sequential database that contains the length-1 sequences as

Table 2.5: PrefixSpan database examples

Sequence ID	**Sequence**
1	<ab>
2	<(ac)h(efg)>
3	<(acb)>
4	<b(ci)a>
5	<dg>

(a) Sequence Database

Length-1 sequence	**Projected Database**
<a>	<b> ; <(_c)h(efg) ; <(_cb)>
<b>	<(ci)a>
<c>	<h(efg)> ; <b> ;<(_i)a>
<(ac)>	<d(efg)> ; <(_b)>
<g>	

(b) Level-by-Level Projected Database

a prefix (e.g., for the frequent 1-sequence <a> as the prefix the projected database is shown in Table 2.5b). The projected database only contains the postifx of these sequences.

3. Find subsets of sequential patterns where these subsets can be mined by constructing projected databases, and mining each one recursively. This process is repeated until the projected database is empty or no more frequent length-*k* sequential patterns are generated.

In this method, there is no candidate generation process, although it costs time and space to construct the projected databases.

The next method, *bi-level* projection, relies on the S-Matrix, introduced in [16], and was proposed to reduce the number and size of projected databases. It starts by performing the same first step as the previous method, which is finding all length-1 sequential patterns. Assuming the database *S* in Table 2.5 we have: <a>, <b>, <c>, <(ac)> and <g>. In the second step a 5x5 lower triangular matrix *M* is constructed as shown in Table 2.6, that represents all the supports for the length-2 sequences, for example M[a,c]=(2,1,2) counts the support for <ac>, <ca> and <(ac)> respectively. Since the information in cell M[c,a] is symmetric to that in M[a,c], a triangle matrix is sufficient, this matrix is called *S-Matrix*[17]. After this the S-Matrix for the frequent length-2 sequences is constructed, and the process are iterated until no other frequent sequences can be found or the projected database becomes empty. The use of this matrix provides fewer and smaller projected databases.

Table 2.6: S-Matrix

a	0				
b	(2,1,1)	0			
c	(2,1,2)	(1,1,1)	0		
(ac)	(0,0,0)	(0,1,1)	(0,0,0)	0	
g	(1,0,0)	(0,0,0)	(1,0,0)	(1,0,0)	0
	a	b	c	(ac)	g

Pseudo projection is used to make the projection more efficient when the projected database can be fitted in main memory. In this method, no physical projected database is constructed in memory, instead each postfix (projection) is represented by two pieces of information: pointer to the sequence in the database and offset of the postfix in the sequence for each size of projection [22].

2.4 Conclusions

This section presents a comparison between the sequential pattern mining algorithms discussed before, focused on the computation time and memory consumption.

GSP is based on the *apriori-based* category, as it explores a candidate generation-and-test approach [2] to reduce the number of candidates to be examined. However, this technique inherits some costs that are independent of the implementation techniques used [25].

1. **Excessive candidate sequence generation.** The set of candidate sequences includes all the possible permutation of elements and repetition of items in a sequence, which in turn generates an explosive set of candidate sequences.

2. **Multiple scans of the database.** The length of a candidate sequence grows by one each time the database is scanned. In a short, to find the sequential pattern with a length of l the database needs to be scanned at least l times.

3. **Difficulty in mining long sequential patterns.** A long sequential pattern contains a combinatorial explosive number of subsequences, and such subsequences must be generated and tested in the a priori-based mining. Thus, the number of candidate sequences is exponential to the length of the sequential patterns to be mined.

The implications of these problems make *apriori-based* algorithms, such as GSP, computationally expensive due to maintaining the support count for each subsequence being mined, which leads to a huge memory consumption in the early stages of mining. The explosive number of candidate sequences also introduces I/O delay, due to the need to utilize a secondary storage in case the number of sequences gets to big. Despite that, GSP and its predecessor *AprioriAll* [1] generate all the sequential patterns although being inefficient. They are also the basis of further researches on mining sequential pattern, since they first introduced sequential pattern mining.

SPADE is classified as a *Apriori-Based and Pattern-Growth Hybrid* since it has key characteristics from both classes. Experiments confirm that SPADE outperforms GSP by a factor of two [29]. It relies on the same generate-an-test characteristic from GSP, which in turn

leads to an excessive generation of candidate sequences. However, SPADE finds a solution to this problem by including some Pattern-Growth characteristics such as search-space partitioning, which includes the vertical database layout defined in [28]. This technique reduces scans of the sequence database, making the algorithm faster. Also, the inclusion of lattice structure partitioning allows to load the decomposed search space into main memory to be mined separately. This strategy provides a more efficient memory management and minimizes I/O costs [29].
On the other hand, the basic search methodology of this algorithm relies on *breadth-first search* and *Apriori pruning*. Despite the pruning, it has to generate large sets of candidates in breadth-first manner in order to grow longer sequences. Thus, most of the difficulties suffered in the GSP algorithm occurs in SPADE as well [15]. One way to minimize this problem is to use a *depth-first search* strategy that requires less memory.

PrefixSpan falls under the *Pattern-Growth* category, and like many other algorithms of this type the key idea for this method is to avoid the candidate-generation step altogether, which is the most time consuming part of apriori algorithms such as GSP [30]. It employs the database projection technique to make the database for the next pass much smaller turning the algorithm faster because the search space is reduced at each step (increasing the computation time in the presence of small support thresholds) [4]. This characteristic greatly reduces memory consumption in terms of candidate sequences.
The main cost of PrefixSpan lies in the projected database generation process, which can be computationally expensive if the databases have to be generated physically. To solve this problem, *pseudo-projection* was suggest to replace the physical projection of a sequence by a sequence identifier. However, this method can only be applied if the projected databases can be fitted into main memory. PrefixSpan also outperforms other algorithms if the sequential pattern is very long, since it can handle its treatment more efficiently.

A performance comparison of GSP, SPADE, and PrefixSpan shows that PrefixSpan has the best overall performance [25]. SPADE, although weaker than PrefixSpan in most cases, outperforms GSP. The comparison also found that when there is a large number of frequent subsequences the three algorithms run slowly [18].

Chapter 3

Off-line Evaluation of Mining Algorithms

3.1 Introduction

This chapter focuses on SIP signalling anomaly detection based on probabilistic hypotheses, which may also be adopted to support SIP signalling behaviour prediction, this work is a first step to analyse SIP dialogues in terms of occurrence of signalling patterns.

We start to describe the considered SIP dialogue and the different SPM algorithms capable of identifying similar SIP signalling sequences and their occurrence rate. After describing how SIP signalling messages are filtered, we evaluate the computation performance of the SPM algorithms for multiple SIP dialogues. The evaluation metrics include the computation time and memory required to identify all SIP sequences and subsequences that form the SIP dialogue. The performance results show that both computation time and memory usage linearly grow with the number of SIP dialogues, allowing the use of the adopted algorithms for high number of SIP dialogues. This conclusion is of particular relevance to future adoption of SPM algorithms in real-time SIP signalling anomaly detection schemes.

This chapter is organized as follows. Section 3.2 identifies related work on SIP anomaly detection. Section 3.3 introduces the SIP architecture adopted in this work. The evaluation of the performance achieved by the SPM algorithms applied to the SIP dialogues is described in Section 3.4. Finally, Section 3.5 concludes the chapter.

3.2 SIP Anomaly Detection

The majority of literature on SIP anomaly detection is particularly oriented to the problematic of malformed SIP messages. The detection of malformed SIP messages has been studied in multiple works, since attacks may be attempted by simply changing the text-based SIP message headers. Malicious SIP messages are usually detected through intrusion detection systems (e.g. firewalls) [21], learning techniques [23] and/or identification of deviations from a priori statistics [19]. The focus of our work is not on the malformed SIP messages but rather on the identification and characterization of SIP dialogues. In [11] support-vector-machine classifiers were adopted to label incoming SIP messages as *good* or *bad*. A SIP message lexical analysis was developed to filter the messages that are not formed according to the standard and in a second stage a semantic filter was applied to the stream of the surviving messages to remove syntactic errors. The work in [14] models the SIP operation as a discrete event system where the state transitions of the SIP dialog are described through a probabilistic counting deterministic timed automata. The description includes the characterization of the SIP sequences and their timmings, which are used a posteriori to detect deviations from the models. Contrary from [14], in our work we do not assume a fixed probabilistic model of the SIP operation. Instead, we are interested in monitoring the SIP messages to characterize the SIP dialogues through the sequences and subsequences that compose them, as well as their occurrence probability.

3.3 System Description

3.3.1 Experimental Testbed

The data repository used in this work was obtained through the generation of SIP messages considering multiple SIP dialogs. The SIP dialogs were generated with SIPp [13], which is a free Open Source SIP traffic generation tool. SIPp allows the creation of customized XML dialogues to define which messages are exchanged between the SIP end users, allowing the occurrence of errors in the transmission of the SIP messages.

The structure of the experimental testbed is illustrated in Figure 3.1 and is based on the architecture described in [9], which merges a GSM radio network with an IMS core network. The OpenBTS architecture [6] is also integrated, which includes a SIP server (Asterisk) a Short Message Service (SMS) server (SMQueue) and a SIP authorization server (SipAuthServe). The Asterisk SIP server is connected via an Access Point to the SIP generator (SIPp). SIP messages can also be generated by GSM terminals supported by a GSM cell implemented through Software Defined Radio. The GSM terminals are supported by the OpenBTS software package, which converts the GSM signalling in SIP signalling and vice versa. Additionally, an IMS core is also available in the experimental testbed, allowing the generation of SIP messages that can be routed to the Asterisk SIP server.

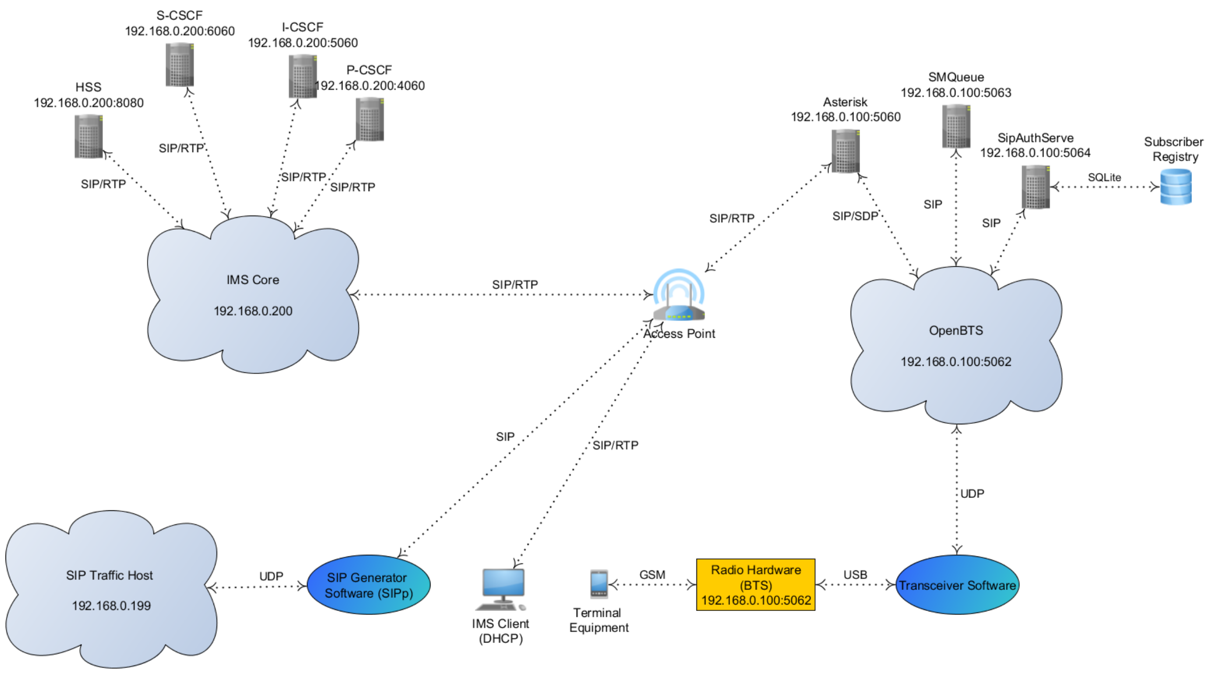
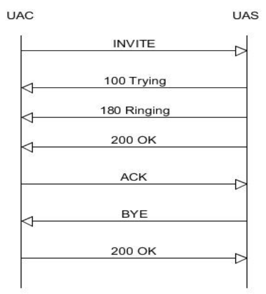


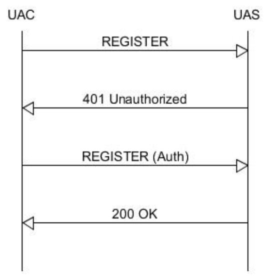
Figure 3.1: SIP Experimental Testbed.

3.3.2 Data Acquisition and Preparation

In the architecture in Figure 3.1, SIPp was used to establish SIP dialogues with the Asterisk Server. The flow of the SIP messages starts from the SIPp Generator, being then routed to the Asterisk server, which analyses the received SIP messages and acts accordingly.



(a) INVITE Request.



(b) REGISTER Request.

Figure 3.2: SIPp Custom Dialogues.

To generate traffic in the SIP Traffic Host a few SIP dialogues were defined beforehand, as shown in Figure 3.2. These dialogues were used to generate the SIP sequences to mine. In addition, some errors were introduced in the SIP dialogues, forcing the loss of packets. The loss of packets was implemented in the SIPp XML file, with the intent of simulating real-time events and represent deviations that could occur during the SIP dialogue. The generated data from these dialogues was recorded in internal logs generated by the SIPp application, detailing each of the SIP messages sent and received. The logs were then cleaned and converted into a more compact format to be mined by the sequential pattern mining algorithms studied in this work (GSP, SPADE and PrefixSpan). To achieve this goal, the SIPp log files were processed using bash scripting with the intent of isolating each message, whether it is a SIP REQUEST or a SIP RESPONSE, by their type (e.g. INVITE, REGISTER, 200 OK), and by Call-ID. The Call-ID is a globally unique identifier of a SIP call. A set of received and sent SIP messages can contain the same Call-ID, therefore representing a time-ordered sequence of messages that constitute a given SIP dialogue. The aggregation of messages by Call-ID and by type represent the time-ordered SIP sequence to be mined.

Assuming the dialogue in Figure 3.2a, which represents a successful INVITE request, an ordered sequence of events can be defined through the signalling messages exchanged between the client and the server. The session starts by sending an INVITE message, which then waits for the server response. After the acknowledgement by the client originating the invite, the session is established until one of the actors (caller/callee) ask to end the call by sending a BYE message. A possible sequence of SIP messages forming the SIP dialogue can be represented by: <INVITE, 100 Trying, 180 Ringing, 200 OK, ACK, BYE, 200 OK>. This sequence of events represents the INVITE dialogue, which is associated to an unique Call-ID. Adopting a specific and predefined alphabet to represent the different types of SIP messages, such as the one represented in Table 3.1, the SIP dialogue can be simply represented by the sequence <1, 15, 16, 21, 2, 3, 21>.

Table 3.1: SIP message alphabet.

SIP message	Character numeration
INVITE	1
ACK	2
BYE	3
100 Trying	15
180 Ringing	16
200 OK	21

3.3.3 SPM Algorithms

The goal here is to quantify the performance of the sequential pattern mining algorithms, a java implementation of these algorithms was used in the performance tests. The SPMF library [12] was adopted in our work, which contains an implementation of the three algorithms (GSP, SPADE and PrefixSpan). The input file format for the algorithms consists of a text file where each line represents a SIP sequence. Each item from a sequence is a positive integer and the items from the same itemset within a sequence are separated by single space. The value −1 indicates the end of an itemset. The value −2 indicates the end of a sequence (e.g. ”1 -1 15 -1 16 -1 21 -1 2 -1 3 -1 21 -1 -2” stands for the sequence <1, 15, 16, 21, 2, 3, 21>, which corresponds to the SIP INVITE dialogue in Figure 3.2a).

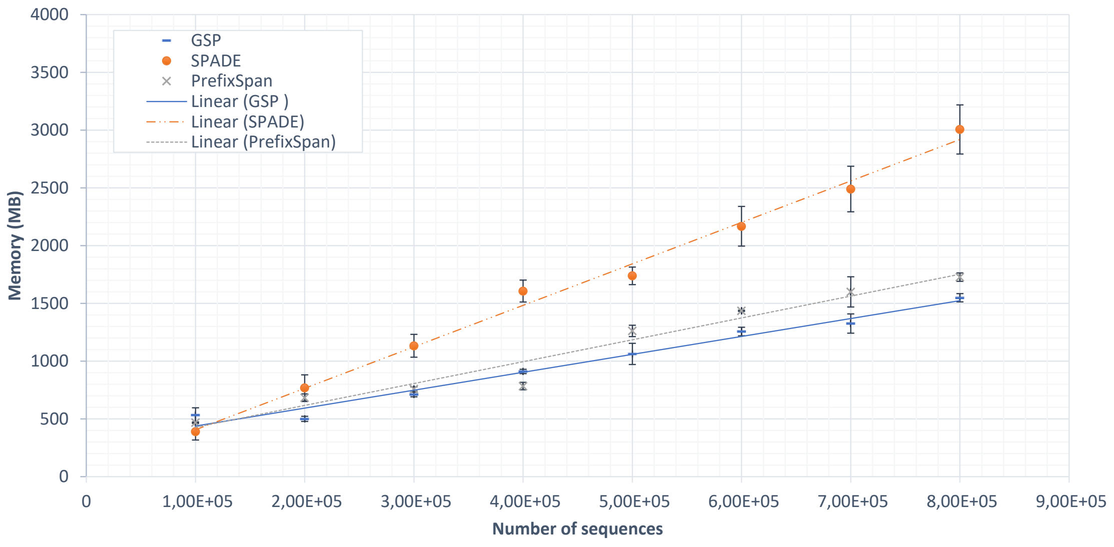
3.4 Performance Evaluation

Several dialogues were considered in the mining task. The outcome of the mining algorithms consists on all frequent SIP signalling sequences and subsequences found in the logs, allowing a descriptive and frequentist analysis. Since both algorithms receive the same sequences as input, they will reach the same outcome of frequent subsequences, therefore the performance evaluation is centred on the computation time and memory required by each algorithm. To evaluate the performance of the selected algorithms, two simple dialogues were chosen. The first one involves a longer SIP dialogue, i.e. having higher number of exchanged SIP messages, in which the establishment of a session can have up to a length-9 SIP sequence. The second dialogue consists in a two message SIP dialogue resulting in a shorter sequence. The goal is to analyse how each sequential pattern mining algorithm behaves in the presence of different sizes of sequences, which would result in the mining of a different number of subsequences. The sequential databases (obtained from the logger) for each of the dialogues contain multiple samples of the same sequence (e.g. < 5 -1 21 -1 >).

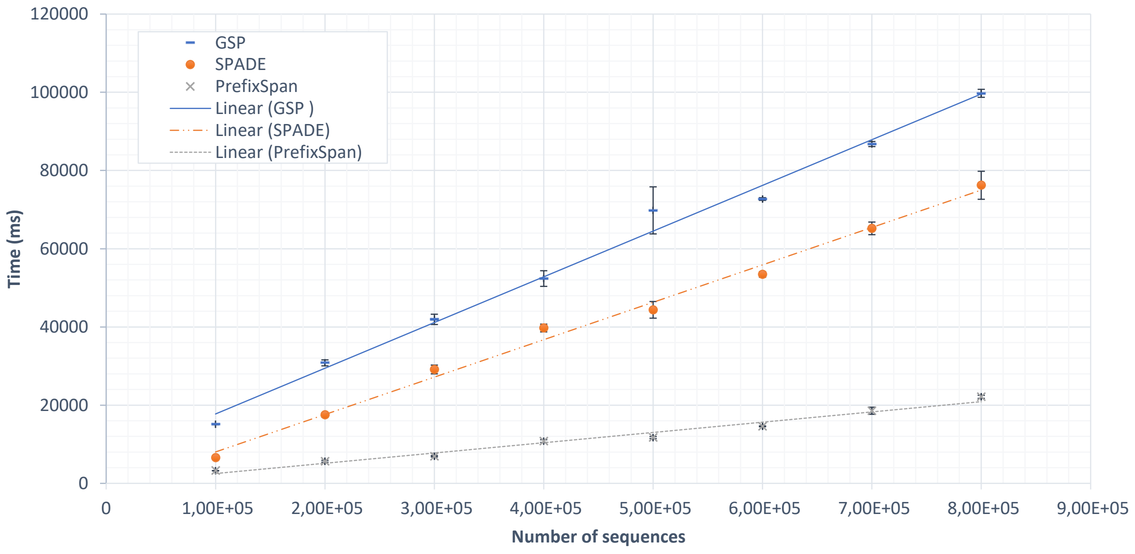
To conduct this experiment, two types of analysis were performed for each algorithm: memory usage and processing time. Each algorithm was computed five times for each sequential database, since the memory usage and computation time are not deterministic. The results obtained in each computation were then used to determine the average, the standard deviation and the error interval for a confidence level of 95%. The SPMF software was run in a 64 bit Windows 10 OS system with 8 GB of RAM and running over a Intel(RM) Core(TM) i7-6700HQ CPU @ 2.60GHz.

3.4.1 Long sequence

In this test, the size of the sequential database ranges from 100k sequences to 800k, being its size restricted by the available RAM. Figure 3.3 represents the memory usage and computation time for the different number of sequences. For each SPM algorithm, the markers with the error bars represent the average values, while the lines approximate the linear trend that best fit the average values. Analysing the results depicted in Figure 3.3b, it can be seen that in terms of computational time, PrefixSpan is by far the fastest algorithm. PrefixSpan also confirms this observation as the number of sequences in the database increases from 100k to 800k. Regarding the SPADE algorithm, although slower than PrefixSpan still outperforms GSP in terms of computing time. In Figure 3.3a, GSP appears to be the most efficient in memory consumption, while SPADE is the one that consumes more RAM. An explanation for this can reside in the fact that the test was run with a data-set that did not use prunning, since the same sequence was repeated several times in the database. By doing so the support count was virtually non-existent since every sequence would pass the support test. In this case, it is expectable that SPADE



(a) Memory usage



(b) Computation time.

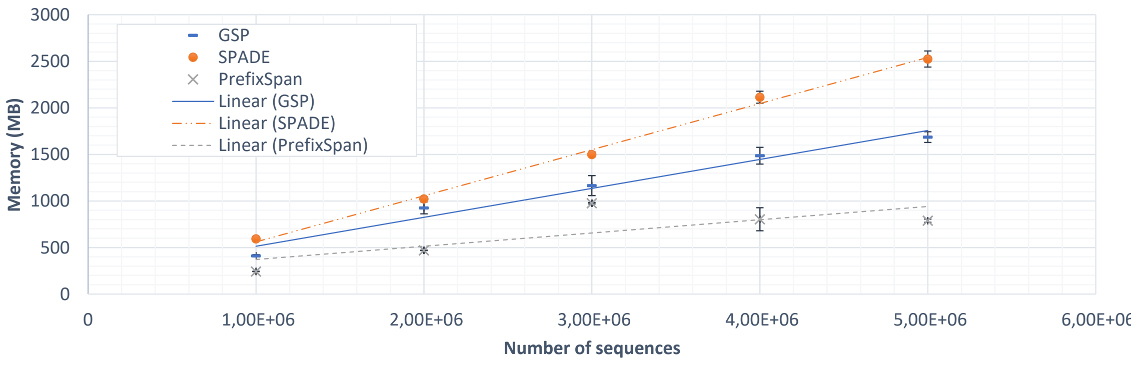
Figure 3.3: Long sequence performance analysis

under-performs because the lack of prunning leads to higher memory consumption.

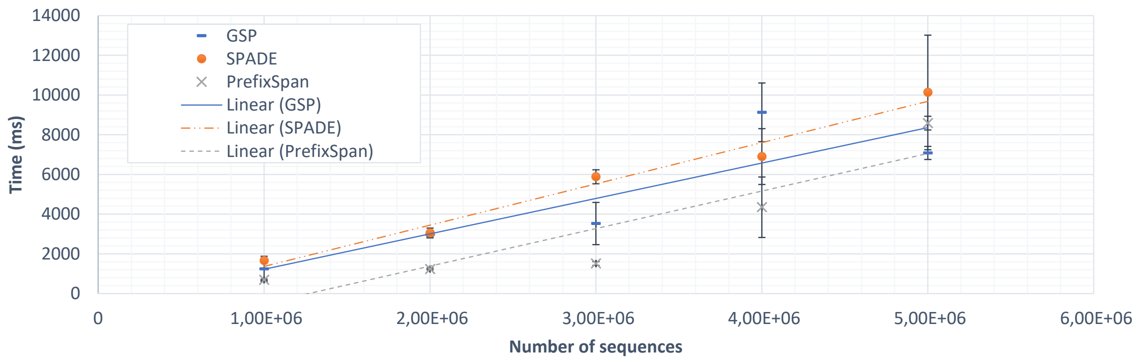
3.4.2 Short sequence

In this case the size of the database ranged from 1 million sequences to 5 millions, due to fact that smaller sequences would result in fewer subsequences, consuming less memory. The results obtained in the short sequence dialogue, plotted in Figure 3.4, show that the computation time increases with the number of sequences considered in the sequence database. A linear trend is observed in the three algorithms as the number of sequences grows, similar to the observed in longer sequences. However, unlike in the previous test, SPADE is now the slowest algorithm with an exception to a few points, were GSP obtains a similar computation time. Despite the experimental results pointing to a change in behaviour of the algorithms in face of smaller sequences, by analysing the pattern deviation of the obtained results we can observe that there is a big uncertainty (see the error interval). The uncertainty could be explained by the fact that the implementation of the algorithms are not efficient enough to handle small sequences that do not require pruning, since they generate additional resources (vertical and projected databases) that are not needed in this case.

In terms of memory, PrefixSpan proved to outperform the other algorithms when handling smaller sequences (despite the lack of pruning) and SPADE exhibits the lowest performance.



(a) Memory usage



(b) Computation time

Figure 3.4: Short sequence performance analysis.

3.5 Final Remarks

From the experiments conducted we can conclude that sequential pattern mining algorithms could potentially be a great aid in identifying behavioural anomalies in telecommunication protocols, and categorizing their data.

Given a SIP message log in the form of a sequence S, these techniques can be used to discover every possible combination of subsequences that can be formed from S, in a relatively short time, especially when the PrefixSpan algorithm is adopted. This is observed for both long and short sequences.

The subsequences extracted from the SIP dialogue can also add a great value as they represent statistical information of the larger sequences, and because each subsequence has a probability to belong to a known sequence. The off-line analysis of the SPM algorithms can support other algorithms to predict SIP dialogues in real-time, by using the probabilities of observing certain sequences and subsequences. SPM algorithms are also useful in the classification of each dialogue by their frequent sequences, since for a given sequence S the sequential pattern mining algorithms always generate the same subsequences that can be built by decomposing the sequence S. The set of subsequences can be seen as a *Fingerprint* of the original sequence and may be used to identify a longer sequence that corresponds to a certain type of SIP dialogue (e.g. anomalous or non anomalous).

As mentioned before, this chapter represents a first step to analyse SIP dialogues in terms of occurrence of signalling patterns.

CHAPTER 4

SIP's Behaviour Stochastic Model

When receiving a SIP message, it is important to understand if the message is expected to appear in the current dialogue, in order to assure the normal behaviour of the protocol. A good way to ensure the good functioning of the SIP session is to classify the received messages, taking in to account the classification of previously known sequences.

This chapter tackles the formal steps to draw a solution for the classification problem. The **Hidden Markov Model** or **HMM** concept is introduced. The **HMM** is a **sequence model**, which is defined as a model whose job is to assign a label or class to each unit in a sequence [20], resulting in mapping a sequence of observations into a sequence of labels. This chapter will cover the definition of Markov Chain in Section 4.1, the definition of a HMM and their adaptation to the SIP Dialogue considered in this work.

4.1 Markov Chains

A **Markov Chain** is defined by a set of states and a set of transitions between states, where each arc is associated with a probability (the probabilities on all arcs leaving a node must sum to 1), and in which the input sequence uniquely determines which states the chain will go through [20]. A Markov chain also has the characteristic of only assigning probabilities to unambiguous sequences, due to the fact it can not represent inherently ambiguous problems.

The event of observing a given SIP message can be seen as a probabilistic case, therefore it can be defined as a Markov chain. Figure 4.1 shows a Markov chain used to assign probabilities to different types of dialogues of SIP messages.

In a formal definition, a Markov Chain can be viewed as a probabilistic model, being

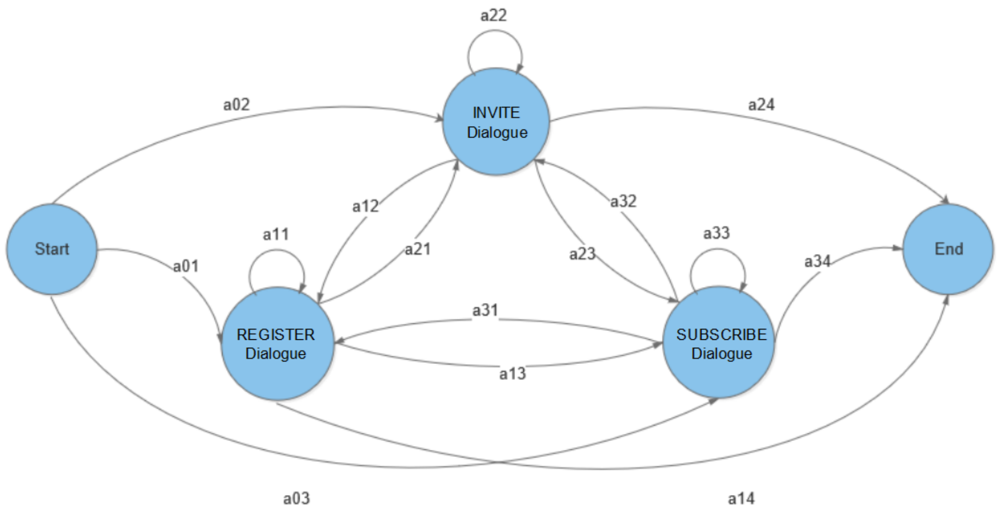


Figure 4.1: Markov Chain for SIP dialogues.

specified by the following components:

(i) $Q = q_1, q_2, \dots, q_n$: A set of N **states**.

(ii) $A = a_{01}, a_{02}, \dots, a_{n1}, \dots, a_{nn}$: A **transition probability matrix** A, each a_{ij} representing the probability of moving from state q_i to state q_j.

(iii) q_0, q_f: A special **start state** and **end state** that are not associated with observations.

Figure 4.1 represents the SIP dialogues (as well as start and end states) as circular nodes in the graph, with the transitions between each node as edges. In a Markov chain, the probability of a particular state depends only on the previous state:

Definition 6 ***Markov Assumption:*** $P(q_i|q_1 \dots q_{i-1}) = P(q_i|q_{i-1})$

An alternative representation of the Markov chain excludes the start and end states, and instead represents the distribution over initial states and accepts states explicitly.

(i) $\pi = \pi_1, \pi_2, \dots, \pi_n$: An **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have π_j=0, meaning that they cannot be initial states. Also, the sum of all π_i must be equal to 1, since π_1=$P(q_i|\text{Start})$.

$$\sum_{i=1}^{n} \pi_i = 1$$

(ii) $QA = \{ q_x, q_y, ... \}$: A set $QA \subset Q$ of legal **accepting states**.

Considering these points, the probability of state 1 (REGISTER dialogue) being the first state can be represented either as a_{01} or as π_1.

4.2 Hidden Markov Model

A Markov chain is useful to compute the probabilities of a sequence of events. However, there are some events that can not be directly observable in the world. Such is the case of a SIP dialogue, since we can not observe these dialogues but instead receive SIP messages as input, and infer the correct SIP dialogues from the sequence of observed messages. In the **Hidden Markov Model**, we have both **observed** events, such as the SIP messages received as input, and **hidden** events, which refer to the SIP dialogues that are represented as causal factors in our probabilistic model.

Introducing a formal definition of a hidden Markov model, focusing on how it differs from a Markov chain, it is specified by the following components:

(i) $Q = q_1, q_2, \dots, q_n$: A set of N **states**.

(ii) $A = a_{11}, a_{12}, ..., a_{n1}, ..., a_{nn}$: A **transition probability matrix** A, each a_{ij} representing the probability of moving from state q_i to state q_j.

(iii) $O = o_1, o_2, ..., o_T$: A set of T **observations**.

(iv) $B = b_i(o_t)$: A sequence of **emission probabilities**, each expressing the probability of an observation o_t being generated from a state i.

(v) q_0, qF : A special **start state** and **end state** that are not associated with observations, together with transition probabilities a_{01}, a_{02}, ..., a_{0n} out of the start state and a_{1F}, a_{2F}, ..., a_{nF} into the end state.

There is also an alternative representation similar to what happens in Markov Chains, which does not rely in start or end states, and focuses on a distribution of initial accepting states,represented by the π notation.

(i) $\pi = \pi_1, \pi_2, \dots, \pi_n$: An **initial probability distribution** over states. π_i is the probability that the Markov chain will start in state i. Some states j may have π_j=0, meaning that they cannot be initial states. Also, the sum of all π_i must be equal to 1, since $\pi_1 = p(q_i|\text{START})$.

$$\sum_{i=1}^{n} \pi_i = 1$$

(ii) $QA = \{ q_x, q_y, ... \}$: A set $QA \subset Q$ of legal **accepting states**.

Taking into account this definition, it is now simpler to use HMMs to analyse a SIP session. By receiving a sequence of observations O, where each observation is represented by the incoming SIP messages (e.g. INVITE, 200 OK, ACK, ...), the goal is to identify the correct hidden sequence H of SIP dialogues. In the particular case of SIP, the hidden sequence H must always contain the same repetition of hidden states throughout the sequence, and should be the same size of our dialogue. To better understand this point, let us assume the INVITE dialogue which is defined by the following sequence according to the adopted alphabet: <1, 15, 16, 21, 2, 3, 21>. Here, since this sequence is a length-7 sequence and is classified as "Dialogue 1", the output of our hidden sequence H must be a length-7 repetition of the states that compose the "Dialogue 1". If this does not occur, we can conclude that an error has happened in the categorizing procedure due to the presence of an abnormal/unknown sequence.

A hidden Markov model instantiates two assumptions. The first one, similar with a Markov chain, states that the probability of a particular state depends only on the previous state, as defined by the Markov Assumption in Definition 6. Finally, the second assumption states that the probability of an output observation o_i depends only on the state q_i that produced the observation o_i and not on any other states or any other observations.

Definition 7 ***Output Independence:*** $P(o_i|q_1, ..., q_i, ..., q_T, o_1, ..., o_i, ..., o_T) = P(o_i|q_i)$

In Figure 4.2 a simplified HMM of a SIP session is shown. In this example, only two hidden states are considered, which correspond to the INVITE and REGISTER dialogue. The observations are drawn from adopted alphabet for this work, where each observation o_k corresponds to a certain SIP message(alphabet).

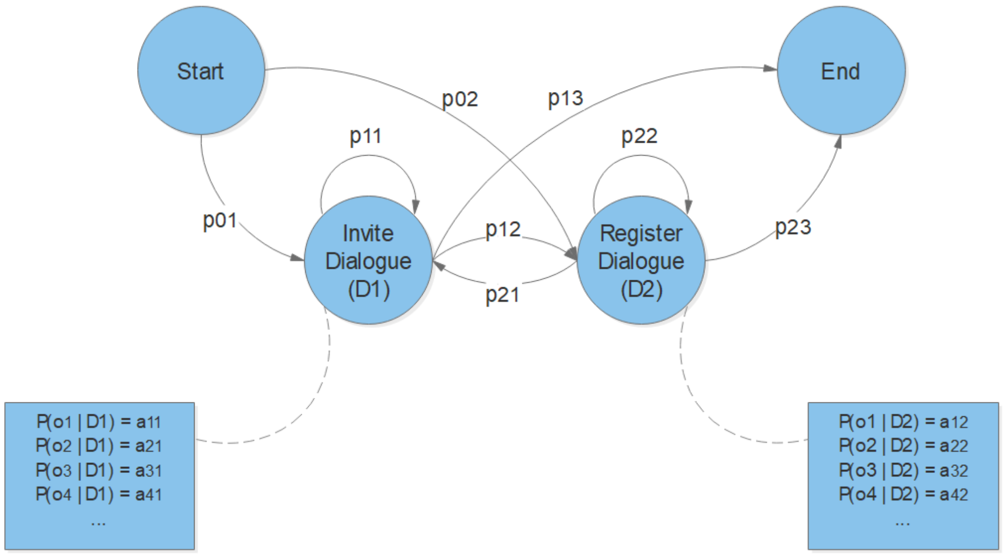


Figure 4.2: Hidden Markov Model for relating SIP messages received to the known Dialogues.

In this HMM, there is a (non-zero) probability of transitioning between any two states. Such an HMM is called a **fully connected** or **ergodic HMM**.

4.3 Solution: Viterbi Approach

As it has been stated in this work, the task of classifying a given sequence of observed SIP messages as a known SIP dialogue can be modelled as a HMM. However, in this particular case, it is important to take into account the likelihood of the previous states, to guarantee the choice of the best SIP dialogue. In order to make this classification in real-time, we have modified an implementation of the Viterbi Algorithm, which was implemented in Matlab.

The Viterbi algorithm is a very common **decoding** algorithm for HMM. In the SIP domain, given a sequence of observed SIP messages, the task of the decoder is to identify the best hidden SIP dialogue sequence. As such, the Viterbi can be seen as a kind of **dynamic programming** algorithm [20]. This kind of algorithms use tables to keep record of intermediate values as it builds up the probability of the observation sequences, and make use of a dynamic programming trellis.

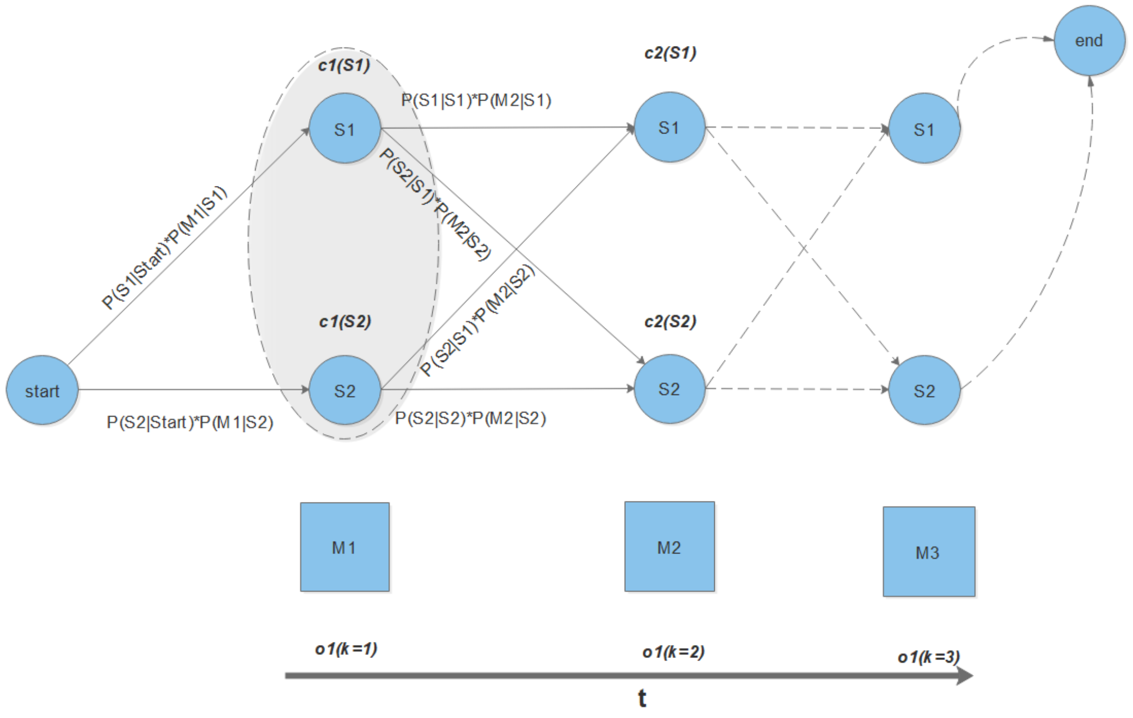


Figure 4.3: The Viterbi trellis.

In Figure 4.3 the Viterbi trellis is presented. Hidden states are represented by circles while the squares represent the observations of the system (SIP Messages in our case) that occurred in a given instance k. The goal is to compute the best hidden state sequence of SIP dialogues (S1, S2) for the given observation sequence of SIP messages (M1, M2, M3). The viterbi algorithm processes the observation sequence left to right, filling out the trellis, where each cell $c_k(j)$ represents the probability that the HMM is in state j after seeing the first k observations. The possible hidden states that can be chosen in a given instant are shadowed by a gray ellipse in the k=1 iteration, where the cell belonging to the path with the higher probability will be chosen. The value of each cell is computed recursively, taking into account the most probable path to lead to this cell.

Listing 4.1: Viterbi Pseudo-code

```
function VITERBI ($O$,$Q$, $\pi$ ,$Y$,$A$,$B$):$X$
    for each state i∈ {1,2,...,$K$}do
        $T_1$[i,1] ← $\pi_i \cdot B_{iy1}$
        $T_2$[i,1] ← 0
    end for
    for each observation  $i$=2,3,... ,$T$ do
        for each state $j$ ∈ {1,2,... ,$K$} do
            $T_1$[$j$,$i$] ← $\max_k$ ($T_1$[$k$,$i$-1] · $A_{kj}$ · $B_{jy_i}$)
            $T_2$[$j$,$i$] ← $\text{argmax}_k$ ($T_1$[$k$,$i$-1] · $A_{kj}$ · $B_{jy_i}$)
        end for
    end for
    $z_T$ ← $\text{argmax}_k$ ($T_1$[$k$,$T$])
    $x_T$ ← $Q_{zT}$
    for $i$ ← $T$,$T-1$,...,2 do
        $z_{i-1}$ ← $T_2$[$z_i$,$i$]
        $x_{i-1}$ ← $Q_{z_i-1}$
    end for
    return $X$
end function
```

In Listing 4.1 a Pseudo-code representation of the Viterbi algorithm is given. The Viterbi algorithm is used for finding the most likely sequence of hidden states (called the **Viterbi Path**) that results in a given sequence of observations. Looking at Figure 4.3, some relations can be extracted as follows:

This algorithm generates a path X=(x_1, x_2, ..., x_T), where each $x_k \in Q$=(q_1, q_2, ..., q_N), that represents a sequence of states, or in our case represents the known SIP dialogues. The input observations are represented by Y=(y_1, y_2, ..., y_T), with $y_k \in O$= (o_1, o_1, ..., o_T) representing the observation space, the observation space represents the SIP alphabet present in Appendix B, whereas the input sequence Y represents the given SIP messages at the time t. In the algorithm, two 2-dimensional tables of size NxT are constructed, where N represents the number of states (known dialogues) and T represents the observation dimension.

- Each element of $T_1[j,i]$ represents the probability of the most likely path so far.
- Each element of $T_2[j,i]$ represents the most likely path so far, in form of an array.

With this characterization, it is possible to use the Viterbi algorithm to start classifying our SIP sequences.

4.4 Data Acquisition and Preparation

In order to acquire relevant information from our SIP sequences using the Viterbi algorithm, some definitions must be taken into consideration.

Definition 8 *Let $S = < m_1, m_2, ..., m_{k-1}, m_k >$ be a SIP sequence of size k, where each m_n represents a SIP message that belongs to a specific Call-ID. Sequence S represents the SIP dialogue, and the SIP messages m_n represent a specific observation.*

Looking at definition 8, we can define a SIP dialogue from a given Call-ID as the observation input that will be fed to the Viterbi algorithm, which will later be classified.

Definition 9 *Given a set of experimental SIP dialogues {Dialogue e_1, Dialogue e_2, ..., Dialogue e_{t-1}, Dialogue e_t}. The set of experimental SIP Dialogues represent the set of hidden states from our HMM. Each SIP dialogue can be viewed as a SIP sequence.*

By defining each SIP dialogue as a hidden state, it becomes possible to compute the best SIP dialogue, for a given Sequence of observations, thus characterizing our SIP sequences. After defining the observation sequences and hidden states, some other definitions can be created.

Definition 10 *Each transition from a SIP dialogue to another is represented by a probability. The probabilities for each transition are stored in an Transition Matrix, and are dependent on a data repository R.*

Definition 11 *The probability of a given SIP message m_n belonging to a certain SIP Dialogue Dg is represented in the Emission Matrix. Each dialogue can have multiple SIP messages, and the same type of SIP message can occur in multiple SIP dialogues.*

These definitions help make a better characterization of the SIP dialogues, thus modeling the SIP problem as an Hidden Markov Model. To conduct the real-time classification of received SIP messages, a Matlab implementation of the Viterbi algorithm was used by adapting the original one available at [8]. The inputs of the algorithm are as follows:

(i) Set of **SIP states** represented by the different dialogues proposed in this thesis;

(ii) **Observation sequence**, which represents our SIP message input that will be analysed in order to determine the current dialogue;

(iii) **Start probabilities** for each of the defined states;

(iv) **Transition matrix** which stores the transition probability of transiting from each of the different states to one another;

(v) **Emission matrix** that stores the probability of observing a certain SIP message *o* from a given state *s*;

These inputs are then sent to the *forward_ viterbi.m* script (available in the Appendix C) which contains the Matlab implementation of the Viterbi algorithm. The algorithm then returns an output array with the most probable sequence of hidden states. To fulfil these requirements, a few Matlab scripts were written to define each of the probabilities and matrices required. The main script can be seen as a two part model, which contains the pre-processing step to create the input variables and feed them to the Viterbi algorithm, and the post-processing step which acts upon the output received from the algorithm and makes changes to the matrices according to the received data. Each of these steps will de detailed further in the thesis.

4.4.1 Pre-Processing Step

To generate the input variables the *run.m* script was created (see Appendix C). The script starts by defining the initial set of known sequences, being gathered before running the Viterbi algorithm. These sequences are used to serve as reference for the classification of the incoming observation input, which will be classified as a known SIP dialogue. The known sequences are extracted from an external text file named *data.txt* and stored into an array that will be used to aid in the construction of the input matrices.

Alongside the sequences, their Dialogue IDs are also retrieved from an external text file, to generate the hidden states. The set of hidden states in this algorithm are represented by the different SIP dialogues defined in this work, and each of the SIP messages are characterized by our created alphabet, which identifies each of the different types of messages. Without loss of generality, in the tests conducted in this work a few experimental dialogues were defined in order to simplify the generation of the results.

After retrieving all the external data needed for the algorithm, a synthetic database needs to be generated in order to simulate a real dataset of SIP message exchange in a given telecommunications network. To do so, some samples were randomly generated from the known sequences. Each sample element represents a certain SIP dialogue, which is associated to a known sequence. With this in mind, the samples can be seen as a randomly sorted set of SIP dialogues. For this thesis the starting dataset contained 100 randomly generated dialogues, which are stored in a vector S with the size 1×100. The samples are only generated once in order to keep the same matrices throughout the tests, so after generating the Sample array this variable is saved and loaded when need to the Matlab environment.

The final steps for the preparation of the data involve defining the **start probabilities** for each of the hidden states, and creating both the **Emission Matrix** and **Transition Matrix**, with the help of initial generated samples of data. The generation of these variables goes as follows:

- The selection of the start probabilities is a simple straightforward method, and involved going through the sample array and calculating the occurrence of each dialogue in terms of percentage. In an initial phase of the tests each of the samples were generated randomly, although having the same probability of being chosen, which led to the starting probabilities between the different dialogues to be similar to each other. Later in this thesis, depending on the type of test being conducted, this percentage can be chosen according to the needs of the evaluation being made.

- For the Transition Matrix, first a matrix of size N was created, where N refers to the dimension of the hidden states, and the transitions between different states are counted, as well as transitions when staying within the same hidden state (e.g. an Invite dialogue is terminated and another Invite dialogue to a different Call-ID is established). After counting these transitions, the probabilities for each of them are calculated and inserted into the Transition Matrix.

- In the Emission Matrix, the goal is to calculate the probabilities of receiving an observation o in a give state s. To do so, a matrix of size $N \times K$ is constructed, which allows to traverse the data vector S and count the number of occurrences of a given observation from our known alphabet in our different SIP states. Finally, after the occurrence the final probabilities are calculated to determine the Emission Matrix.

All these input variables are then fed to the Viterbi algorithm. The algorithm then calculates the probabilities for each of the hidden sequences according to the received observation sequence, in order to fill out the trellis. As a result of the calculations, the algorithm returns an array of possible dialogues for the given sequence of SIP messages. The next section will cover post-processing done to the data in order to confirm the data and change the matrices accordingly.

4.4.2 Post-Processing Step

Once the algorithm is finished calculating the best sequence of SIP dialogues, three distinct variables are returned as output: **argmax** which contains the best path for the hidden states, **valmax** that represents the highest probability correspondent to the chosen path, and **total** that represents the sum of all the probabilities, including the observations. In this thesis however only the argmax variable will be analysed, since it contains the most-likely classification of the SIP observations. Then, the first step of the post-processing is analysing what the algorithm output is, and check if the classification was successful or not.

However, some notions need to be gathered in order to correctly processes the output, since the observation input can vary widely, and can be categorized as three distinct types of SIP sequences:

- **known sequences** - The observation sequence was composed of a set of SIP messages that would result in one of the known SIP dialogues.
- **Error sequences** - The received sequence contained a deviation from one of the SIP dialogues, where some SIP messages where either missing or added to the original dialogue, causing a deviation error from the known sequences.
- **Unknown sequences** - The Viterbi algorithm received a sequence with SIP messages that were not initially present when creating the matrices, thus generating an unknown situation.

Each of these different types of messages produces different types of categorizations by the algorithm. In some experiments, it was found that for an observation of a **known sequence** with size k, the Viterbi algorithm would return a sequence of size $k+1$, where each element would refer to the correct SIP dialogue, having thus a classification of the most probable dialogue for each observation, and showing in the end the next most probable state for the observation (e.g. when receiving an observation <1, 15, 16, 21, 2, 3, 21> which corresponds to Dialogue 1, the output of the algorithm would generate a vector of size 8 where each item contains the classification "Dialogue 1", as demonstrated in Figure 4.4).

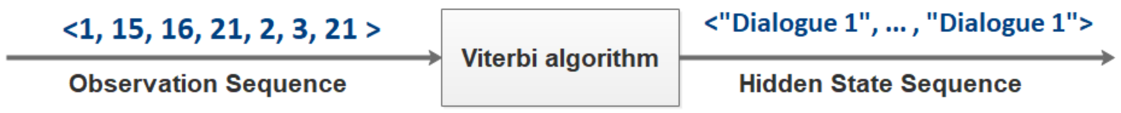


Figure 4.4: Matlab Viterbi Implementation Scheme.

With this knowledge of the input and output behaviour, a few flaw detections can be made by analysing the output vector, in two different ways:

1. **Size of output sequence:** The size of the output sequence needs to be the same size as the attributed dialogue classification, excluding the extra element for the next state (if the observation is a replica of a "Dialogue 1" sequence, the size of the output should be the same size as a typical "Dialogue 1" sequence, which is lenght-7, plus the classification for the next hidden state, which ends with a lenght-8 sequence).
2. **Output sequence content:** All the elements from the output sequence should belong to the same SIP dialogue, due to a given dialogue only having one corresponding sequence that represents it. Therefore, if the algorithm returns a path with several dialogues, we can assume that an error has occurred either in the classification by the algorithm, due to the probabilities, or by a SIP sequence error.

For the **unknown sequence**, the output of the viterbi algorithm returns an empty vector, due to the fact that if in the Emission and Transition matrices there are no occurrences of a given SIP message, the algorithm gives a probability of 0 to that message, which results in not finding any fitting dialogue to classify the observed event.

Returning to the post-processing step, which is implemented in the *run.m* script, after receiving the output variables, the argmax array is analysed for its content, and checked for each of the three types of possible SIP sequences.

If argmax returns an empty array, it can be concluded that the observed sequence represents an unknown sequence. Therefore, the observation sequence is stored in the *data.txt* file, and a new dialogue ID is generated for the observation. The sequence is saved using the *saveSequence.m* script which preforms the previous actions, as well as storing the observation ID in the sample array, in order to recalculate the probabilities of the matrices.

Next, the elements of the argmax array are verified for similarity. In the case where all elements are equal to one another, the observation sequence is checked in the known sequence array to see if it matches the given classification. If the observation is equal to the given SIP dialogue, a correct classification has occurred, and the dialogue ID is saved in the sample array to solidify the probability of the given SIP dialogue.

However, if the observation differs from the classified dialogue sequence, we can conclude to have occurred an incorrect classification. In this case, the observation sequence is checked in the full extent of the known sequence array to see if it has any occurrence in the array. If the observation sequence exists, then only its ID is saved in the sample array. If the dialogue does not exist at all, the full sequence is saved as it happens in the case of an unknown sequence. The same process described in this paragraph is made when the elements from the argmax array are different from each other.

At the end, after finishing the analysis of the output array, the number of correct classifications is counted in order to understand how much iterations are needed for the algorithm to correctly classify a given sequence. In the next chapter, an example of some of the tests performed with Viterbi algorithm are presented.

Chapter 5

Experimental Results

This chapter is focused on testing the SIP stochastic model, that was presented in the previous chapter, analysing the performance of the suggested implementation for the Viterbi algorithm, when in the presence of different types o SIP sequences. In the following sections a description of the different tests and an analysis of the data gathered will be presented, starting with a dynamic evaluation in Section 5.1, and a similarity evaluation in Section 5.2, and ending with some final remarks on the conclusions taken from each experiment in Section 5.3. These tests were conducted in a 64 bit Windows 10 OS system with 8 GB of RAM, and running over a Intel(RM) Core(TM) i7-6700HQ CPU 2.60GHz.

5.1 Stochastic Dynamics Evaluation

To generate the initial database for the SIP tests, a Matlab script was used to generate a set of 100 randomized samples from the known sequences in Table 5.1. The samples are identified by their *dialogue ID* throughout the sample array and are generated according to their corresponding percentage (e.g. sequence <1, 15, 16, 21> represented by the experimental dialogue ID "1"(Dialogue e_1) accounts to 60% of the samples in the sample array). After the initial generation, the samples were randomly sorted to simulate a real data set of SIP dialogues, which in turn provided transitions in the sample array that were considered to calculate the Transition matrix in the pre-processing step.

The first test to be conducted aimed to gather insights on the performance of the **Viterbi** algorithm in the presence of new datasets unknown to the algorithm. The initial sample set of known sequences consists in the length-4 sequences described in Table 5.1. Each of the three sequences are differently distributed accordingly to their percentages throughout the *sample array*, which will influence the creation of the Emission and Transition matrices.

Table 5.1: Known Sequences.

Dialogue ID	SIP Sequence	Percentage
e_1	<1, 15, 16, 21>	60%
e_2	<1, 15, 4, 21>	30%
e_3	<6, 30, 6, 21>	10%

The proposed test involves generating 100 samples of an unknown sequence, and then counting the number of iterations needed for the algorithm to correctly classify the new sequence. In Figure 5.1, we can observe the number of iterations needed to correctly classify each of the different sequences. In a first observation, it can be seen the differences in the convergence of each of the sequences to their correct classification, which is directly related to the probabilities of the Emission and Transition matrices.

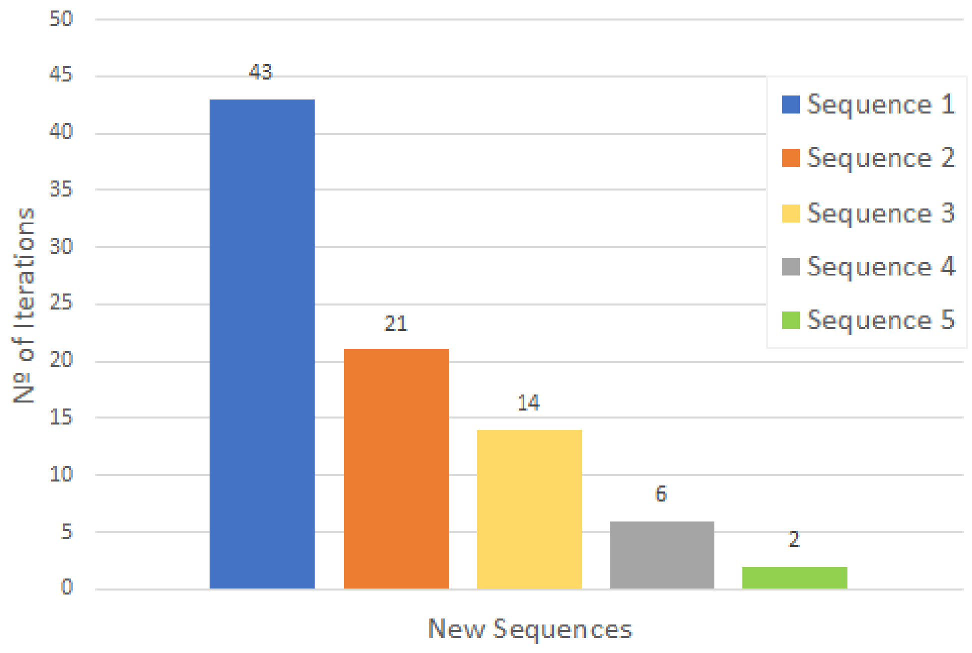


Figure 5.1: SIP Dynamic Evaluation.

In **Sequence 1**, which represents the sequence <1, 21, 16, 21>, as shown in Table 5.2, it took 43 iterations for the Viterbi algorithm to successfully identify the sequence. Looking to the known sequences, we can observe that sequence <1, 15, 16, 21>, or in short **Dialogue** e_1 is the most frequent sample in our database with 60% of occurrences. This makes it the sequence with the higher number of transitions in the database, which in turn leads to a higher probability of transitioning from a given sequence to a Dialogue e_1 message. Also, each of the elements (SIP messages) in **Dialogue** e_1 have a greater chance of belonging to this sequence, due to the number of occurrences of this sequence in the sample array. In return, this leads to the probabilities in the emission matrix to rise for this messages, making it more likely to belong to the SIP Dialogue e_1.

Table 5.2: Sequence Mapping.

Sequence 1	<1, 21, 16, 21>
Sequence 2	<1, 21, 4, 21>
Sequence 3	<1, 15, 30, 21>
Sequence 4	<6, 15, 6, 21>
Sequence 5	<7, 15, 40, 21>

This leads to the algorithm taking much longer to correctly identify the new sequence, since in the emission matrix there is more probability of addressing the new input to Dialogue e_1, rather than to point to the newly observed Sequence 1. Since Sequence 1 contains the SIP message with the code 16, which is present in **Dialogue** e_1, the algorithm in the early stages classifies the new input as a Dialogue e_1. Once the sample array is enriched with the new sequence and the emission and transition matrices are reconstructed, the probabilities become more balanced and the algorithm correctly classifies the inputs.

Next in Figure 5.1, we have **Sequence 2**, which consists in the SIP sequence <1, 21, 4, 21>. The number of iterations for this sequence to converge now decreases to 21, since its similar candidate is Dialogue e_2, represented by <1, 15, 4, 21> with an occurrence of 30% in the sample array. Once again, the decisive element is SIP message 4, that only occurs in Dialogue e_2, but since this dialogue has less representation in the sample array, the algorithm is much faster at correctly identifying the new input.

In **Sequence 3** the case is different, since it contains elements from both dialogue e_1 and e_1 the probabilities from the emission matrix disperse through both dialogues, making the probabilities low. By doing so the algorithm needs less iterations to identify **Sequence 3**. In **Sequence 4**, the algorithm converges quickly due to the fact that dialogue e_3 represented by sequence <6, 30, 6, 21> has a low occurrence in the sample matrix, therefore the algorithm has less difficulty in detecting the new SIP dialogue.

Finally, for the last SIP dialogue, **Sequence 5** represents an unknown sequence that contains SIP messages that were not previously observed by the algorithm thus not showing in the Emission Matrix, namely message 7 and 40. Since the algorithm has no previous representation of this kind of sequence, after the first reconstruction of the matrices, the algorithm immediately detects the new sequence from then on.

5.2 Stochastic Similarity Evaluation

Another important feature to consider in this experimental test, is to verify the robustness of the algorithm when processing similar sequences with slight deviations from one another. In this section, the goal is to preform a similarity test to the Viterbi algorithm. This is done by checking if the presence of nearly identical sequences will have any effect in the correct classification of the input sequence. In this case scenario, the initial database is equally repartitioned, where each sequence is given the same amount of samples, as shown in Table 5.3.

Table 5.3: Similarity Test Database.

Dialogue ID	**SIP Sequence**	**Percentage**
e_1	<2, 15, 5, 21>	33%
e_2	<1, 20, 7, 21>	33%
e_3	<1, 18, 4, 41>	33%

The core of the test however stays unchanged, which involves generating 100 samples of new sequences that are similar with each other, and with the database sequences. For this test the proposed input sequences are represented in Table 5.4. Each sequence differs at least one item from each other, and every SIP message form the input sequences needs to be present in the database, in order to maintain as much similarity between the sequences as possible.

Table 5.4: Similarity Sequences.

Sequence 1	<1, 18, 4, 21>
Sequence 2	<1, 18, 5, 21>
Sequence 3	<1, 20, 5, 21>
Sequence 4	<2, 20, 5, 21>

Figure 5.2 shows the number of iterations per sequence, required for a valid classification adopting the Viterbi algorithm. Unlike the results presented in the dynamic test of Figure 5.1, here the number of iterations remains practically constant for each of the different sequences, with the exception of Sequence 4 that required less iterations. This results may be explained by the composition of the similarity database. Since each known sequence is equally repartitioned in the sample array, the probabilities of the Transition matrix do not vary too much for each transition, making the probabilities similar, which leads to each new added sequence to require nearly the same amount of iterations to converge.

Another important aspect is the Emission matrix that reflects the same experience. Each item present in the three known sequences from our similarity database, does not appear more than two times, which leads to the probabilities being more centred around only one sequence. This is the reason why **Sequence 4** requires less iterations to be correctly classified, since the SIP message that corresponds to the character 2 is only present in the first dialogue <2, 15, 5, 21>.

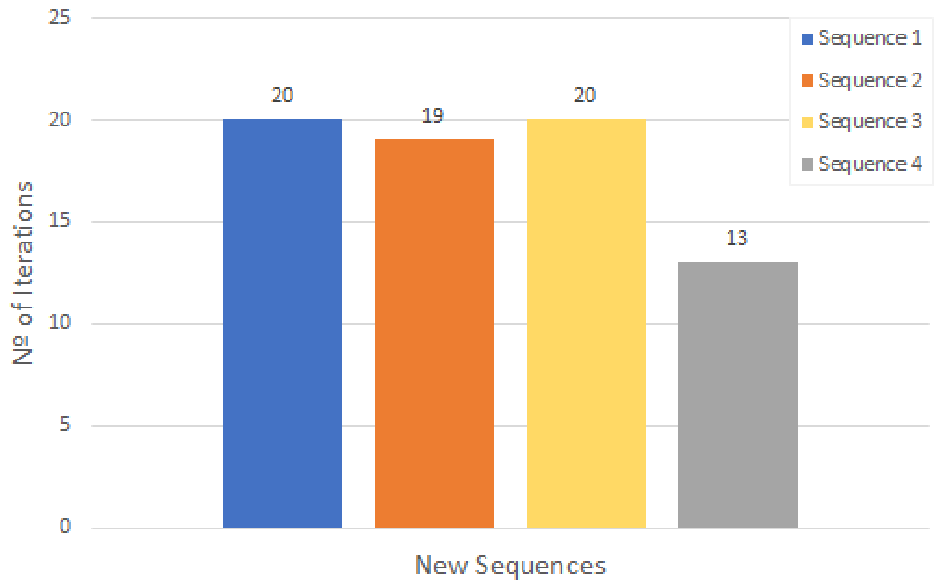


Figure 5.2: SIP Similarity Evaluation.

5.3 Final Remarks

From these experiments, some evidence was gathered for accepting the Viterbi algorithm as a reliable technique for classifying sequences of SIP messages. By defining each hidden state as a SIP dialogue, and having a data set that contains information about each known sequence, it is possible to classify a given sequence as one of these established states.

The Viterbi algorithm proved to be efficient both in the dynamic evaluation and in the similarity test, needing only a few iterations to correctly identify the received SIP sequences as their attributed dialogue. These tests are a step to guarantee the robustness of this technique, while showing the adaptability to different case scenarios that could occur in a real SIP dialogue between two distinct users.

Chapter 6

Conclusions

This work presents a characterization of SIP dialogues, in order to provide a tool capable of diagnosing and detecting erratic SIP sequences. In the tests preformed using the algorithms studied in this work, we demonstrate different tools capable of classifying different sequences of SIP messages. The next sections include some final remarks regarding the developed work in this thesis, as well as some contributions made, ending with some analysis on the future work this thesis leaves to be explored.

6.1 Final Remarks

Starting with a bibliographic survey of data mining techniques that were capable of analyse big bulks of sequential data, a study on sequential pattern mining algorithms was preformed in Chapter 2. These algorithms provide techniques capable of detecting frequent patterns of data, which could prove to be useful in order to characterize SIP dialogues. To do so, three classical sequential pattern mining algorithm were studied and tested in depth, namely the GSP, SPADE and PrefixSpan algorithms. Each of these algorithms were studied to better understand their functionality, and learn what different benefits they could provide to our solution.

A few experimental tests were conducted in Chapter 3, where an off-line analysis of the SPM algorithms was made. In these tests, the three studied algorithms were evaluated in terms of performance for their processing time, and memory consumption. The goal of this test was to understand the efficiency of each algorithm, and learn what useful knowledge could be mined from the data. Some materials included in Chapter 3 were published in a paper in **INForum** 2018 conference, in Appendix D. The work developed aided in the characterisation of the SIP sequences, adding valuable statistical information from the different subsequences mined with the proposed algorithms.

The statistical information studied in Chapter 3 was not enough to fully characterize the SIP dialogues. With that in mind, a new bibliographic survey was performed, covering methods capable of classifying incoming sequences of data. The solution we have adopted in this work was based on **Hidden Markov Models** (HMM). The HMM allowed the definition of a sequence model capable of characterizing each SIP message in a sequence. In the study, we were able to define SIP sequences as a set of states, with the use of Markov Chains. This technique was used to determine transitions between each of these states and allowed to classify the SIP dialogues.

The conducted tests in Chapter 5 focused on evaluating the robustness of the Viterbi algorithm in the presence of different types of input SIP sequences. First, a dynamic test was made using random SIP sequences, and finally a similarity test was conducted in order to determine the number of required iterations to classify the SIP sequences. The tests proved to be effective and were fast to classify the input sequences. In terms of contributions, Chapter 5 provides a correct classification of sequences of SIP messages, thus being an effective tool capable of characterising the SIP dialogues in a fast and efficient way.

6.2 Future Work

After finishing the tests for the different studied algorithms, it was shown that the implementations adopted in the work were capable of characterising SIP dialogues in a short window of time. The different techniques were capable of not only providing interesting statistical information from off-line data repositories, as well as successfully classifying incoming sequences of SIP messages. It would be interesting in the future to include the off-line statistical information in the real-time classification of the sequences in order to determine the performance gain in terms of computation time, where the off-line evaluation can be seen as an intermediate step, by priorly identifying non-anomalous patterns of SIP dialogues. Another important aspect to consider is analysing if the apriori knowledge of the different subsequences of messages, and their statistical information, could help to prematurely detect deviations in the real-time classification of the SIP sequences. This behaviour detection in real time could potentially be a great aid in order to make the protocols more secure, by preventing exterior attacks, thus turning the telecommunications protocols more efficient and safe.

Bibliography

[1] R. Agrawal and R. Srikant. “Mining sequential patterns.” In: *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*. IEEE. 1995.

[2] R. Agrawal, R. Srikant, et al. “Fast algorithms for mining association rules.” In: *Proc. 20th int. conf. very large data bases, VLDB*. Vol. 1215. 1994.

[3] R. Agrawal, T. Imieliński, and A. Swami. “Mining association rules between sets of items in large databases.” In: *Acm sigmod record*. Vol. 22. 2. ACM. 1993.

[4] C. Antunes and A. L. Oliveira. *Sequential pattern mining algorithms: trade-offs between speed and memory*. 2004.

[5] J. Ayres, J. Flannick, J. Gehrke, and T. Yiu. “Sequential pattern mining using a bitmap representation.” In: *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2002, pp. 429–435.

[6] D. A. Burgess, H. S. Samra, et al. “The openbts project.” In: *Report available at http://openbts. sourceforge. net, http://openBTS. org* (2008).

[7] B. A. Davey and H. A. Priestley. *Introduction to lattices and order*. Cambridge university press, 2002.

[8] David Conger. *"Forward Viterbi Algorithm"*. Accessed: 2018-06-26. URL: https://www.mathworks.com/matlabcentral/fileexchange/24516-forward-viterbi-algorithm.

[9] A. F. R. Estevam. “GSM-SIP Gateway.” Master’s thesis. FCT - UNL, 2017.

[10] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. “From data mining to knowledge discovery in databases.” In: *AI magazine* 17.3 (1996), p. 37.

[11] R. Ferdous, R. L. Cigno, and A. Zorat. “On the Use of SVMs to Detect Anomalies in a Stream of SIP Messages.” In: *2012 11th International Conference on Machine Learning and Applications*. Vol. 1. 2012, pp. 592–597. DOI: 10.1109/ICMLA.2012.109.

[12] P. Fournier-Viger, A. Gomariz, T. Gueniche, A. Soltani, C.-W. Wu, and V. S. Tseng. “SPMF: a Java open-source pattern mining library.” In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3389–3393.

[13] R. Gayraud, O. Jacques, and C Wright. *SIPp: traffic generator for the SIP protocol*. http://sipp.sourceforge.net/. 2009.

[14] D. Golait and N. Hubballi. "Detecting Anomalous Behavior in VoIP Systems: A Discrete Event System Modeling." In: *IEEE Transactions on Information Forensics and Security* 12.3 (2017), pp. 730–745. ISSN: 1556-6013. DOI: 10.1109/TIFS.2016.2632071.

[15] J. Han and M. Kamber. *Data Mining: Concepts and Techniques. Series Editor Morgan Kaufmann Publishers*. 2000.

[16] J. Han, J. Pei, B. Mortazavi-Asl, Q. Chen, U. Dayal, and M.-C. Hsu. "FreeSpan: frequent pattern-projected sequential pattern mining." In: *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM. 2000, pp. 355–359.

[17] J. Han, J. Pei, B. Mortazavi-Asl, H. Pinto, Q. Chen, U. Dayal, and M. Hsu. "Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth." In: *proceedings of the 17th international conference on data engineering*. 2001.

[18] J. Han, H. Cheng, D. Xin, and X. Yan. "Frequent pattern mining: current status and future directions." In: *Data Mining and Knowledge Discovery* 15.1 (2007).

[19] N. Hentehzadeh, A. Mehta, V. K. Gurbani, L. Gupta, T. K. Ho, and G. Wilathgamuwa. "Statistical Analysis of Self-Similar Session Initiation Protocol (SIP) Messages for Anomaly Detection." In: *2011 4th IFIP International Conference on New Technologies, Mobility and Security*. 2011, pp. 1–5. DOI: 10.1109/NTMS.2011.5720662.

[20] D. Jurafsky and J. H. Martin. *Speech and language processing*. Vol. 3. Pearson London: 2014.

[21] H. Li, H. Lin, H. Hou, and X. Yang. "An Efficient Intrusion Detection and Prevention System against SIP Malformed Messages Attacks." In: *2010 International Conference on Computational Aspects of Social Networks*. 2010, pp. 69–73. DOI: 10.1109/CASoN.2010.23.

[22] N. R. Mabroukeh and C. I. Ezeife. "A taxonomy of sequential pattern mining algorithms." In: *ACM Computing Surveys (CSUR)* 43.1 (2010).

[23] M. Nassar, R. State, and O. Festor. "Monitoring SIP Traffic Using Support Vector Machines." In: *Proceedings of the 11th International Symposium on Recent Advances in Intrusion Detection*. RAID '08. Cambridge, MA, USA: Springer-Verlag, 2008, pp. 311–330. ISBN: 978-3-540-87402-7. DOI: 10.1007/978-3-540-87403-4_17. URL: http://dx.doi.org/10.1007/978-3-540-87403-4_17.

[24] J. Pei, J. Han, B. Mortazavi-Asl, and H. Zhu. "Mining access patterns efficiently from web logs." In: *Knowledge discovery and data mining. Current issues and new applications* (2000).

[25] J. Pei, J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, and M.-C. Hsu. "Mining sequential patterns by pattern-growth: The prefixspan approach." In: *IEEE Transactions on knowledge and data engineering* 16.11 (2004), pp. 1424–1440.

[26] R. Srikant and R. Agrawal. "Mining sequential patterns: Generalizations and performance improvements." In: *Advances in Database Technology—EDBT'96* (1996).

[27] Z. Yang, Y. Wang, and M. Kitsuregawa. "LAPIN: Effective sequential pattern mining algorithms by last position induction." In: (2005).

[28] M. J. Zaki. "Efficient enumeration of frequent sequences." In: *Proceedings of the seventh international conference on Information and knowledge management*. ACM. 1998, pp. 68–75.

[29] M. J. Zaki. "SPADE: An efficient algorithm for mining frequent sequences." In: *Machine learning* 42.1 (2001).

[30] Q. Zhao and S. S. Bhowmick. "Sequential pattern mining: A survey." In: *ITechnical Report CAIS Nayang Technological University Singapore* (2003).

Appendix A

SIP Dialogue Diagrams

A.1 Dialogue 1

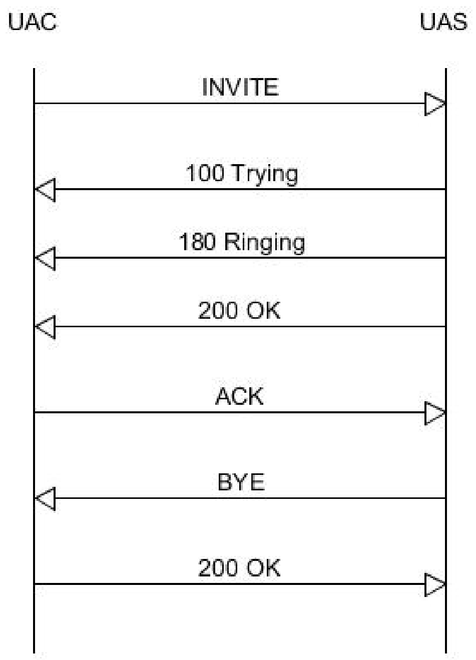


Figure A.1: Dialogue 1 (Successful INVITE request).

A.2 Dialogue 2

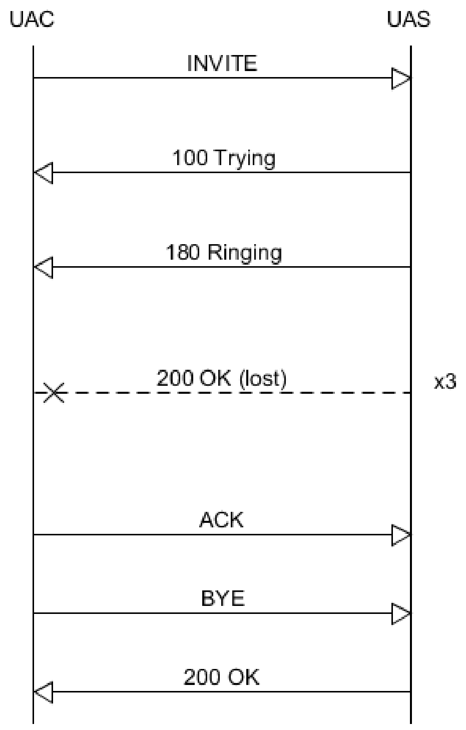


Figure A.2: Dialogue 2 (INVITE: 200 OK lost).

A.3 Dialogue 3

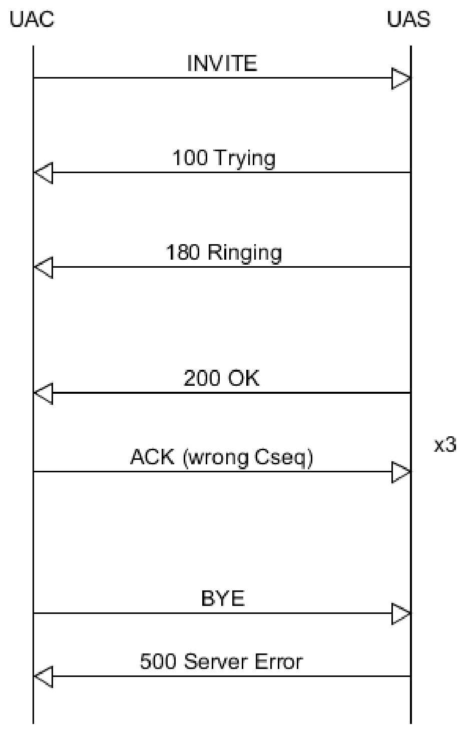


Figure A.3: Dialogue 3 (INVITE: ACK wrong cseq).

A.4 Dialogue 4

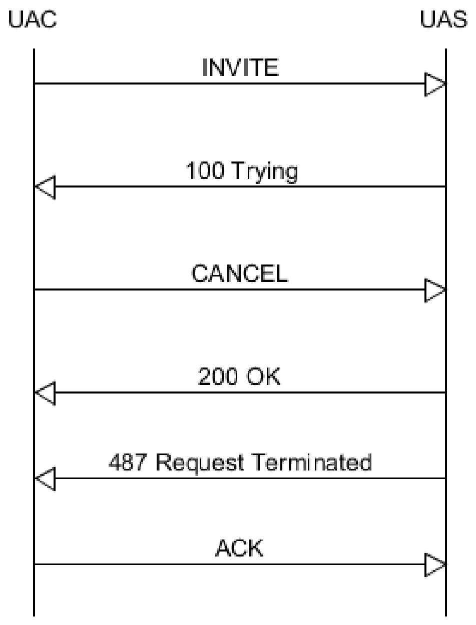


Figure A.4: Dialogue 4 (INVITE: CANCEL after 100 Trying).

A.5 Dialogue 5

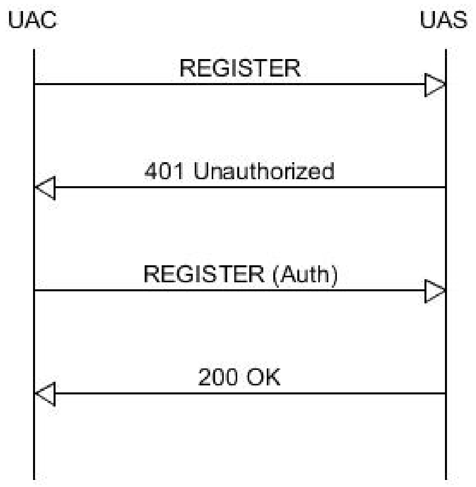


Figure A.5: Dialogue 5 (Successful REGISTER request).

A.6 Dialogue 6

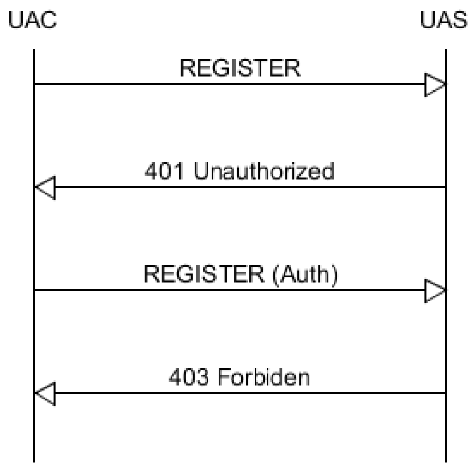


Figure A.6: Dialogue 6 (REGISTER: 403 Forbidden).

A.7 Dialogue 7

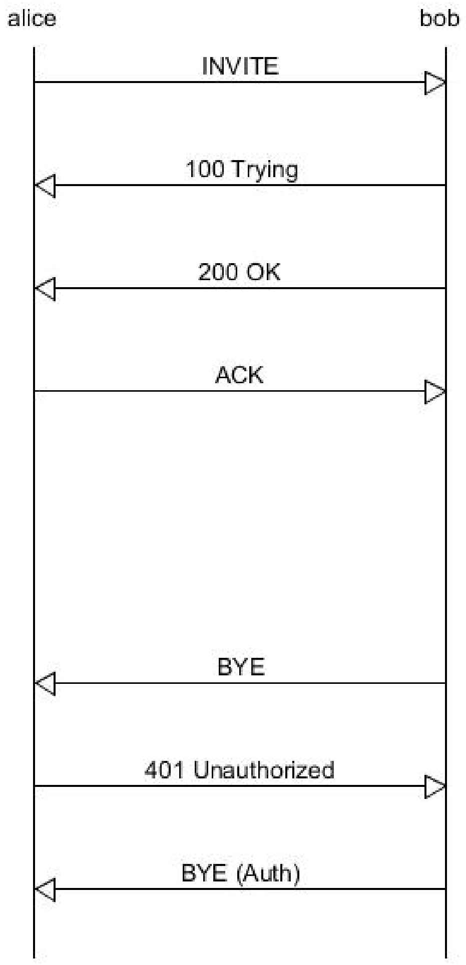


Figure A.7: Dialogue 7 (UAC Digest Leak).

A.8 Dialogue 8

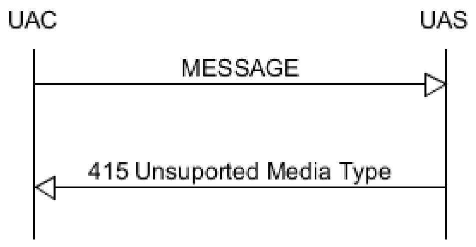


Figure A.8: Dialogue 8 (Bad Message).

A.9 Dialogue 9

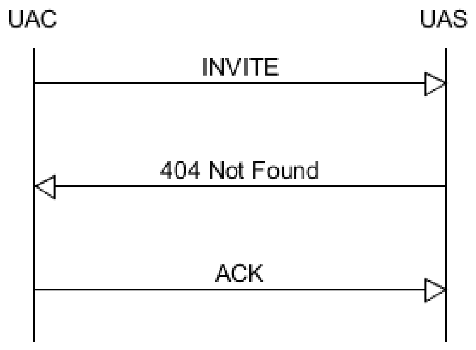


Figure A.9: Dialogue 9 (INVITE: 404 Not Found).

A.10 Dialogue 10

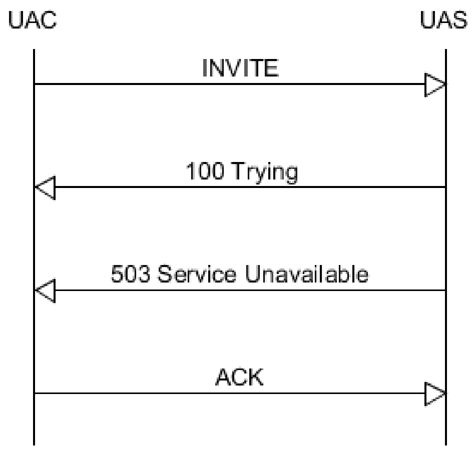


Figure A.10: Dialogue 10 (INVITE: 503 Service Unavailable).

A.11 Dialogue 11

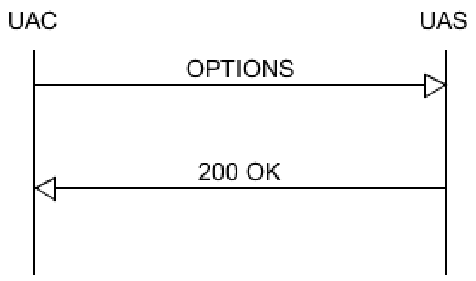


Figure A.11: Dialogue 11 (OPTIONS).

A.12 Dialogue 12

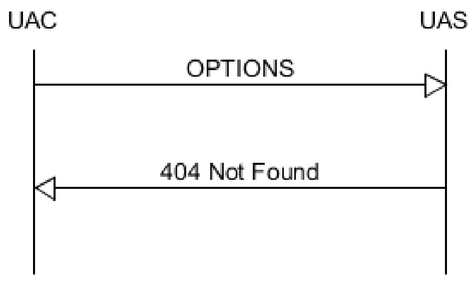


Figure A.12: Dialogue 12 (OPTIONS: 404 Not Found).

A.13 Dialogue 13

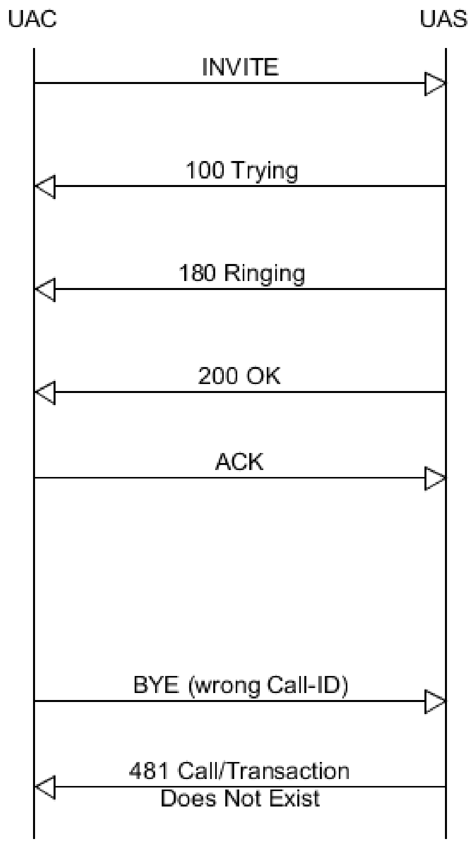


Figure A.13: Dialogue 13 (INVITE: wrong Call ID in bye).

A.14 Dialogue 14

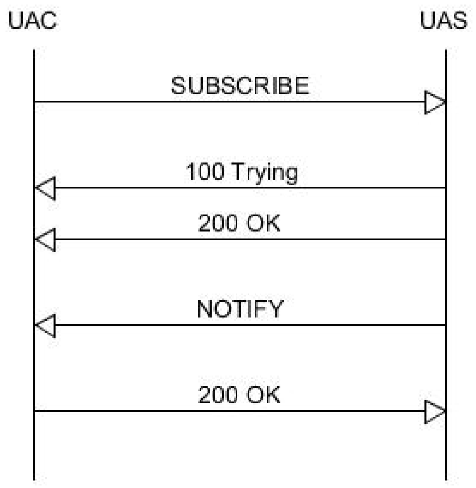


Figure A.14: Dialogue 14 (SUBSCRIBER).

A.15 Dialogue 15

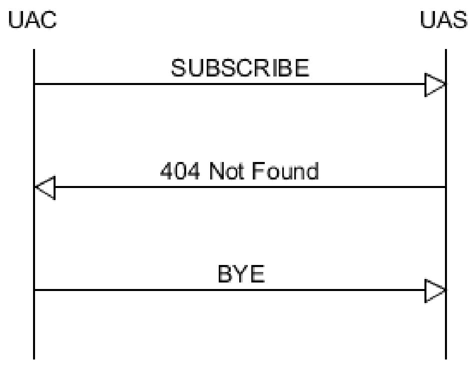


Figure A.15: Dialogue 15 (SUBSCRIBER: 404 Not Found).

Appendix B

SIP Alphabet Numeration

B.1 Requests

- 1 – INVITE
- 2 – ACK
- 3 – BYE
- 4 – CANCEL
- 5 – OPTIONS
- 6 – REGISTER
- 7 – PRACK
- 8 – SUBSCRIBE
- 9 – NOTIFY
- 10 – PUBLISH
- 11 – INFO
- 12 – REFER
- 13 – MESSAGE
- 14 – UPDATE

B.2 Provisional Responses

- 15 – 100 Trying
- 16 – 180 Ringing
- 17 – 181 Call is being forwarded
- 18 – 182 Queued
- 19 – 183 Session in Progress
- 20 – 199 Early Dialog Terminated

B.3 Successful Responses

- 21 – 200 OK
- 22 – 202 Accepted
- 23 – 204 No notification

B.4 Redirection Responses

- 24 – 300 Multiple Choices
- 25 – 301 Moved Permanently
- 26 – 302 Moved temporarily
- 27 – 305 Use Proxy
- 28 - 380 Alternative Service

B.5 Client Failure Responses

- 29 – 400 Bad Request
- 30 – 401 Unauthorized
- 31 – 402 Payment required
- 32 – 403 Forbidden
- 33 – 404 Not Found
- 34 – 405 Method Not allowed
- 35 – 406 Not Acceptable

- 36 – 407 Proxy Authentication Required
- 37 – 408 Request Timeout
- 38 – 409 Conflict
- 39 – 410 Gone
- 40 – 411 Length Required
- 41 – 412 Conditional Request Failed
- 42 – 413 Request Entity Too Large
- 43 – 414 Request-URI Too Long
- 44 - 415 Unsupported Media Type
- 45 – 416 Unsupported URI Scheme
- 46 – 417 Unknown Resource-Priority
- 47 – 420 Bad Extension
- 48 – 421 Extension Required
- 49 – 422 Session Interval Too Small
- 50 - 423 Interval Too Brief
- 51 - 424 Bad Location Information
- 52 – 428 Use Identity Header
- 53 – 429 Provide Referrer Identity
- 54 – 430 Flow Failed
- 55 – 433 Anonymity Disallowed
- 56 – 436 Bad Identity-Info
- 57 – 437 Unsupported Certificate
- 58 – 438 Invalid Identity Header
- 59 – 439 First Hop Lacks Outbound Support
- 60 - 440 Max-Breadth Exceeded
- 61 – 469 Bad Info Package
- 62 – 470 Consent Needed

- 63 – 480 Temporarily Unavailable
- 64 – 481 Call/Transaction Does Not Exist
- 65 – 482 Loop Detected
- 66 – 483 Too Many Hops
- 67 – 484 Address Incomplete
- 68 – 485 Ambiguous
- 69 – 486 Busy Here
- 70 - 487 Request Terminated
- 71 – 488 Not Acceptable Here
- 72 – 489 Bad Event
- 73 – 491 Request Pending
- 74 – 493 Undecipherable
- 75 – 494 Security Agreement Required

B.6 Server Failure Responses

- 76 – 500 Server Internal Error
- 77 – 501 Not Implemented
- 78 – 502 Bad Gateway
- 79 – 503 Service Unavailable
- 80 – 504 Server Timeout
- 81 – 505 Version Not Supported
- 82 – 513 Message Too Large
- 83 – 580 Precondition Failure

B.7 Global Failure Responses

- 84 - 600 Busy Everywhere
- 85 – 603 Decline
- 86 – 604 Does Not Exist Anymore
- 87 – 606 Not Acceptable
- 88 – 607 Unwanted

Appendix

C Matlab Scripts

C.1 run.m Script

Listing C.1: run.m script

```
function run(obs)
%read data from text file
M = csvread('data.txt');

%size of Matrix M in rows

for k=1:size(M,1),

    known_seqs{k} = M(k,:); % get rows from Matrix and store in cell array
    known_seqs{k}(known_seqs{k}==0) = []; % remove zeros from cell

    id_nok(k)=k; %get id for each sequence
end

scenario_id = csvread('scenario_id.txt');

for k=1:length(scenario_id),
    string = strcat('Scenario',32,num2str(scenario_id(k)));
    string = cellstr(string);
    states{k} = [string{:}];  %get id for each sequence
end

N=length(states); %dimension of hidden states
K=70;% emission alphabet

%create sample matrix

```

```
    num_samples = 100;
  generateSamples(num_samples);
    load samples.mat

    S_seq = known_seqs(samples);
    S_index  = id_nok(samples);

    %start probability for each state
    start_p = zeros(1,length(states));

%get values of start probability from sample matrix

    for k=1:length(states),
        start_p(k) = sum(S_index(:) == k)/num_samples;
    end

%create Transition Matrix

A =zeros(N);
for k=2:length(S_index),  %transictions between different sequences
  A( S_index(k-1), S_index(k)) = A( S_index(k-1), S_index(k)) +1;
end

for k=1:length(S_index), %transictions when staying within the same sequence
     A(S_index(k), S_index(k))=A(S_index(k), S_index(k)) + length(S_seq{k})-1;
end

%Transition Matrix

trans_p = normalizeByRows(A);
disp('Transition␣Matrix');
disp(trans_p);

%create Emission Matrix

B=zeros(N,K);

for k=1:length(S_index), %transictions when staying within the same sequence
   for m=1:K,
     num_elems= length(find(S_seq{k}==m));
     B(S_index(k), m) = B(S_index(k), m) + num_elems;
   end
end

%Transition Matrix

emit_p=normalizeByRows(B')';
disp('Emission␣Matrix');
disp(emit_p);

```

```
%Run Viterbi algorithm

%Pr(obs) path Pr(path)
[total, argmax, valmax] = forward_viterbi(obs,states,start_p,trans_p,emit_p);

disp(argmax);
disp(total);
disp(valmax);

%Post Processing

%Check if sequence is unknown.

if isequal(argmax,[]),
    saveSequence(obs,scenario_id,samples);
    classification(0); %Inorrect classification - 0
else
    %Check if all positions the same
    if all(strcmp(argmax, argmax{1}))
        disp('Same␣elements');

        %get position in states matrix of the given scenario
        scenario = [argmax{1}];
        index = find(strcmp(states, scenario));

        if isequal(obs, known_seqs{index})
            disp('Correct␣classification');
            saveSample(samples,index);
            classification(1); %Correct classification - 1
        else
            disp('Incorrect␣classification');
            classification(0); %Inorrect classification - 0

            index = findSequence(known_seqs,obs);

            if index ~[],
                disp('Correct␣scenario:');
                saveSample(samples,index);
                disp(states(index));
            else
                disp('Non␣existent␣scenario.␣Storing␣in␣database');
                saveSequence(obs,scenario_id,samples)
            end
        end
    else
        disp('Not␣same␣elements');
        index = findSequence(known_seqs,obs);
        classification(0); %Inorrect classification - 0

        if index ~[],
            disp('Correct␣scenario:');
            saveSample(samples,index);
            disp(states(index));
        else
            disp('Non␣existent␣scenario.␣Storing␣in␣database');
            saveSequence(obs,scenario_id,samples);
        end
    end
end

clearvars
end
```

C.2 generateSamples.m Script

Listing C.2: generateSamples.m script

```
function generateSamples(num_samples)
    samples = [];
    samples = [samples ones(1,floor(0.33*num_samples))];
    samples = [samples 2*ones(1,floor(0.33*num_samples))];
    samples = [samples 3*ones(1,floor(0.33*num_samples))];
    samples = samples(randperm(length(samples)));
    save samples.mat samples
end
```

C.3 classification.m Script

Listing C.3: classification.m script

```
function classification(class)
    %store sequence
    fid = fopen('classification.txt', 'a+');
    fprintf(fid, '%d␣\n',class);
    fclose(fid);
end
```

C.4 saveSequence.m Script

Listing C.4: saveSequence.m script

```
function saveSequence(obs,scenario_id, samples)
  %store sequence
    fid = fopen('data.txt', 'a+');
    fprintf(fid, '\n');
    fprintf(fid, '%d,',obs);
    fclose(fid);

    %store scenario id
    new_id = scenario_id(end) + 1 ;

    fid = fopen('scenario_id.txt', 'a+');
    fprintf(fid, ',%d',new_id);
    fclose(fid);

    %save samples
    samples = [samples new_id];
    save samples.mat samples
end
```

C.5 findSequence.m Script

Listing C.5: findSequence.m script

```
function res=findSequence(known_seqs,obs)
index = [];

for k = 1:numel(known_seqs)
  if isequal(known_seqs{k}, obs)
     index = k;
     break;
  end
end

res=index;

end
```

C.6 normalizeByRows.m Script

Listing C.6: normalizeByRows.m script

```
function res = normalizeByRows(A)

[l c] =size(A);
for k=1:l,
    A(k,:)= A(k,:)/sum(A(k,:));
end
A(isnan(A))=0;
res=A;

end
```

C.7 forward_viterbi.m Script

Listing C.7: forward_viterbi.m script

```
function[total,argmax,valmax]=forward_viterbi(obs,states,start_p,trans_p,emit_p)

   T = {};
   for state = 1:length(states)
       T{state} = {start_p(state),states(state),start_p(state)};
   end
   for output = 1:length(obs)
       U = {};
       for next_state = 1:length(states)
           total = 0;
           argmax = [];
           valmax = 0;
           for source_state = 1:length(states)
               Ti = T{source_state};
               prob = Ti{1};
               v_path = Ti{2};
               v_prob = Ti{3};
               p=emit_p(source_state,obs(output))*trans_p(source_state,next_state);
               prob = prob*p;
               v_prob = v_prob*p;
               total = total + prob;
               if v_prob > valmax
                   argmax = [v_path, states(next_state)];
                   valmax = v_prob;
                   disp('next␣and␣source');
                   disp(argmax);
               end
           end
           U{next_state} = {total,argmax,valmax};
       end
       T = U;
   end
   %total = 0;
   argmax = [];
   valmax = 0;
   for state = 1:length(states)
       Ti = T{state};
       prob = Ti{1}; v_path = Ti{2}; v_prob = Ti{3};
       total = total + prob;
       if v_prob > valmax
           argmax = v_path;
           valmax = v_prob;
       end
   end
end
```

Appendix D

INForum 2018 Paper

This appendix includes the paper accepted in the INForum 2018 conference (with peer review), which was presented in an oral presentation.

Offline Sequential Analysis of Session Initiation Protocol Messages

Ricardo Matos and Rodolfo Oliveira

Departamento de Engenharia Electrotécnica, Faculdade de Ciências e Tecnologia, FCT, Universidade Nova de Lisboa, 2829-516 Caparica, Portugal
IT, Instituto de Telecomunicações, Portugal

Abstract Session Initiation Protocol (SIP) has been massively adopted for establishing and controlling communication sessions that support multimedia services, including but not limited to Voice over Internet Protocol (VoIP) or IP Multimedia Subsystem (IMS) services. The growing adoption of the SIP protocol has motivated the development of analytical tools capable of diagnosing and/or identifying potential problems caused by inconsistent SIP signaling. Focusing on SIP signaling anomaly detection based on probabilistic hypotheses, which may also be adopted to support SIP signaling behavior prediction, our work is a first step to characterize SIP dialogs in terms of distinct signaling sequences and on their frequency probability. We start by describing the considered SIP scenario and different Sequential Pattern Mining (SPM) algorithms capable of identifying similar SIP signaling sequences and their frequency. After describing how SIP signaling messages are filtered, we evaluate the SPM algorithm's computation performance to characterize different SIP signaling flows. The evaluation metrics include the computation time and memory required to identify all SIP sequences and subsequences of the SIP flows. The evaluation results show that both computation time and memory usage linearly grow with the flow of SIP messages. This conclusion is of particular relevance to future adoption of such algorithms in real-time SIP signalling anomaly detection schemes.

1 Introduction

In recent years, the usage of machine learning and data mining techniques has been massively adopted, mainly due to the increase in processing power and to the scientific breakthroughs in data science. Machine learning is also being adopted in the telecommunications arena, as it can bring many advantages in the processing of bulks of data that are generated by a plethora of different sources.

In this work we are motivated by the advantages of adopting machine learning techniques to supervise the signaling of multimedia communication sessions. The SIP protocol [1] is used to initiate, maintain and terminate real-time sessions, being adopted by multiple VoIP clients and other multimedia services (e.g. IMS). The growing adoption of the SIP protocol requires analytical tools capable of identifying inconsistent sequences of SIP signaling messages and their possible