Facultat d'Informàtica de Barcelona

# Learning user behaviours from website visit profiling

*Author:*
Alberto Gutiérrez Torre

*Supervisor:*
Josep Lluís Berral García
*Tutor at FIB:*
Ricard Gavaldà Mestre

FIB - Ingeniería en informática
Proyecto de final de carrera, 37,5 créditos

June 19, 2014

**DADES DEL PROJECTE**

| | |
|---|---|
| *Títol del projecte:* | Learning user behaviours from website visit profiling |
| *Nom de l'estudiant:* | Alberto Gutiérrez Torre |
| *Titulació:* | Enginyeria Informàtica |
| *Crèdits:* | 37,5 |
| *Director:* | Josep Lluís Berral García |
| *Departament:* | Llenguatges i Sistemes Informàtics |

**MEMBRES DEL TRIBUNAL** *(nom i signatura)*

| | |
|---|---|
| *President:* | Jorge Castro Rabal |
| *Vocal:* | Cristina Barrado Muxi |
| *Secretari:* | Ricard Gavaldà Mestre |

**QUALIFICACIÓ**

*Qualificació numèrica:*

*Qualificació descriptiva:*

*Data:*

**Acknowledgements**

I want to thank the crew of Revistes Campus Nord for all these years of publications that made me grow in many ways, even with this project. I also want to thank all the guys from office Omega S108 who helped me improving myself in the field of researching, specially Jorge Muñoz who gave me material for research and advised me and Alex Vidal who was my faithful proofreader.

Thanks to Mans Hulden for developing Treba software. Thanks to RDLab people for letting me use the cluster and work into this project in laboral hours when needed. Thanks to all my friends for supporting me this long time, specially my dear Emma.

Last but not least, I must thank to Ricard Gavaldà for his time spend on advising and correcting leading this project towards the correct way and Josep Lluís Berral as I owe this project to him.

# Contents

# Chapter 1

# Introduction

Everyday, tons of events and information are registered in files and logs from any technical or business system, which eventually grow to levels where it would be reckless for a single or a small group of experts to analyze them.

Analysing this data could bring us some interesting information about what is happening on it, and help us improve it. Trying to analyze all this data, most of which could not lead to infer useful information and can be time consuming, might be a bad idea. On the other hand, we should focus our attention on that information that can provide us the knowledge we require to improve. Our *time is money* but *information is power*.

As time is quite valuable in our quest for knowledge, fields like *Data Mining* focus on the development of algorithms to make easier these analyses. *Data Mining* is a sub-field of computer science which aims to extract information from a data sources and transform it into an understandable structure for further analysis.

## 1.1 The problem

Nowadays, the Internet is a massive net of content exchange. It has changed a lot since its birth in the late 60s and, with it, so has changed the lifestyle of a big part of the world's population. Where and when we get information, how do we see the world, our behavior. Traffic on the Internet may be seen as random or chaotic because of its massive usage but, as for most of human acts, there are underlying patterns. The Internet can be modeled as a playground of users. Users have perceptions, intentions and goals. The delivered web pages and content are part of what users perceive. The service from these web pages and these contents are part of the goals the users aim to obtain, for example check the email or download a file. Finally, what users

decide to do in the website is the traffic produced. This traffic produced towards the goal is the user intentions.

Website administrators and web operators would like to know which kind of users visits the website, for example in order to optimize contents. The administrator should know the goals of the users for this. To know the goals of the users is impossible, because their minds can not be read; however their intentions can be guessed. If their intentions are correctly interpreted, their goals can be deduced.

The register of those intentions (the log files) can be massive, as the Internet is, therefore users can not be analyzed one by one. An automatic method to classify users in groups is needed. If users are classified in groups by the resemblance of their actions, the analysis can be easier. This means classifying by more or less general behaviors, an abstraction of the intentions. Having those behaviors grouped by similarity, the website administrators can attempt to give labels for each one, and then infer what each set of user is trying to obtain, thus, inferring their goals.

This knowledge obtained is *actionable knowledge* which is knowledge that can be used to take actions. Commonly, this actions are focused in resolving problems or taking advantages. We will illustrate this concept with some web-related examples where we take advantatges from the knowledge:

- Caching: Given a website were there are 100 pages but only 10 stand as the most used pages, we would want to this 10 pages to load faster. If we have the knowledge that these 10 pages are the most accesses, we can establish a cache system that improves this situation.

- Precomputed queries: If we have a web system were we can query to a static database and there is a set of most common queries, these queries can be precomputed for faster user experience and have less CPU cost. If we perform an analysis on which queries are more used, we may establish a ranking ordered by usage and, for example, precalculate the 10 first queries.

- Dynamical pricing: Imagine we have an e-commerce shop. We know that a lot of users take a look at a product page but most of them do not buy. If we are aware of this situation, we may do a promotion on this item to increase sells.

- Page reorganization: If we have a complex website, we may want to look at how users use their pages. Sometimes, complex websites make hard to find the piece of information you want because of the organization. If there is a way to know how users travel through the website

and a pattern is found, the page can be reorganizated to improve user experience.

This kind of situations are common. Apart from these, there are more different situations, depending on the page and what the administrator want or expects. This actionable knowledge is what we want to obtain from the study of user behaviours.

## 1.2 About this project

We expect to find various tendencies on web sites and use them to group users and find patterns. This will lead to know more about the users and how the traffic is done on the web sites. This approach is important as actionable knowledge can be obtained and, with it, the web pages can be improved The project consists in the development of the software to perform the analysis and analyze data-sets with individual user behaviors in order to see if the software is useful and how can we improve it.

The aim of this project is to design a tool to extract actionable knowledge from a web server log. As we mentioned, data mining is in charge of treating data and obtaining knowledge from massive amounts of data, and preparing it to use for our purposes. For this task, as in other ones like optimization of processes or trend analysis in business, we are applying algorithms and techniques from this field, like modeling and prediction, that will accomplish our goals much faster than with human analytic processes. This will provide useful knowledge for website administrators to know better their website and their users from a new point of view.

Using logs as input may not be the best solution for acquiring data with precision, however it is the best way for the convenience of the web administrators. There are other methods that use cookies or javascript code to gather data, but this would require modifications of the websites and then wait to gather the data. Moreover, even using these methods we have no warranty that the issues we have using logs as data set are avoided.

As logs are gathered automatically by servers, the administrators can use the results of this project to analyze their web sites without any previous preparation nor modification in their web sites (not counting at this moment with privacy issues).

The two specific techniques that we are using are clustering and Finite State Automaton (FSA) inference. These two techniques will provide different points of view of the same data, allowing to perform a better analysis of the data. They will be defined in the next Chapters 5 and 6 respectively.

## 1.3 State of the art

Web analysis is not a new concept at all, however it is still an area under development. For example, one of the most popular and recognized website traffic analyzer is Google Analytics. The way Google Analytics gathers data is by a JavaScript code that the administrators must insert in their pages. This script is in charge of sending the information needed for the analysis. Google analytics gives you various different metrics of your website like, for example, the country of the users viewing your site, their age and even what page are they watching in real time if this feature is enabled. The traffic shape and the user passing by can be known taking a look to a feature of this tool where lists all the pages and the transitions between them. However, different types of user or close relations between them can hardly be inferred from this analysis. This is because the traffic is all mixed and analyzing every branch can be exhausting as this tool was made to know where does most of the traffic go.

The work presented in this project has also some points in common with the area of *process mining*. Process mining is a relative young research discipline that sites between machine learning and data mining on the one hand and process modeling and analysis on the other hand [1, 2]. Similar to our work, the idea of process mining is to use the unbiased information recorded in the event logs to model the process *as it is*, in contrast with the possible biased idea the process owner has of his own process. Process mining techniques can be classified in three groups depending on their final goal:

- Discovery: A discovery technique takes and event log and produces a model without using any a-priori information.
- Conformance: An existing process model is compared with an event log to check if the reality, as recorder in the log, conforms to the model and vice versa.
- Enhancement: The idea is to extend or improve an existing process model using information about the actual process recorded in the log.

Similar to process mining discovery approaches, our approach also aims to discover a model of the reality using the unbiased information recorded on the log. And in both process mining and data mining, large volumes of data considered require efficient and automatic techniques. However, process mining techniques are clearly process-oriented, in other words, processes become first class citizens. On the other hand, we propose an user-driven perspective, where the modeling and the analysis is done focusing on the user behavior.

The work of Nicolás Poggi et al. [3–5] studies some of cases of interaction in e-commerce, and attempts to decide if a specific user is interesting for our commercial goals or not. These works aim is to find business processes in website logs using the tools like ProM [6] in order to create models to predict the workload of a server. The main approach is closer to our idea than the one of Google Analytics. However what we want to find is not business processes, we want to find how users behave in our website and create groups in order to infer how behaves each group. Moreover, we aim to cover general web-sites providing different kinds of users and behaviors, not just e-commerce sites, although they are the most in need of these solutions.

There is a sub-field of *data mining* that is closer to us than *process mining*, *web mining*. As described in the work of Bing Liu [7], *web mining* aims to discover useful information or knowledge from web structure, page content and usage data. These three sources produce a subdivision in areas:

- Web structure mining: Web structure mining discovers useful knowledge from hyperlinks, which represent the structure of the web site.
- Web Content mining: Web content mining extracts useful information from the content of the page.
- Web usage mining: Web usage mining refers to the discovery of user access patterns form web logs.

From the three categories, the one closest to the project is the *web usage mining*. Inside of this sub-area there exists a part dedicated to the analysis of sequential and navigational patterns, which is exactly what we are doing but with a user behaviour perspective.

Web Utilization Miner (WUM) is a web miner focused on analysis of navigational patterns introduced in the work of Myra Spiliopoulou et al. [8]. In this approach, traffic is represented as a trie which is ordered tree data structure that is used to store a dynamic set or associative array where the keys are usually strings though for data fast retrieval. This trie is obtained from a web traffic log.

The major feature of WUM is the MINT query language that implements. This language allows to do different queries on the the trie structure in order to analyze the traffic. This approach is quite interesting, however is not what we are looking for. The main difference with our approach is that we offer a set of algorithms to process data. Moreover, our approach does not need of specific knowledge of a query language as the representation of the results is directly shown to the user. This way our application can be used by all web administrators, who does not need to know about anything related

to the process done with data to use the results. This decision makes our framework less flexible, but more usable for users without task knowledge.

For various different uses, like measuring the performance of a web server, there are tools created to do traffic simulation, like HTTPerf [9]. This simulation is done on a web server using a model to follow.

## 1.4 Our main goals

We can now state more precisely our main goal, which is to build a piece of software that, given a register of the traffic, outputs models useful for web site analysis. These models will be of two kinds: group models and models that represent transition between groups.

This goal is split in other goals that define the structure of this document:

- Understanding how the web traffic is.
- Selecting a source of data.
- Finding suitable techniques for the modeling and test them.
- Implementing these techniques in an efficient and scalable way to produce an user-friendly piece of software. At the end of the project we will have a prototype of the software that fulfils our requirements.

As we are trying to infer behaviours for users that are not categorized as we do not have any other information about them, the validation of the results must be done by the administrators of the websites.

We have presented our project, our context, how it differs from other present processes and applications available or in development and our goals. The project itself is worth of study as it is, somehow, a new concept and can be useful for administrators.

## 1.5 Structure of this document

This document is divided in 8 chapters, including this one, and an appendix.

- Chapter 2 describes the methodology used, the planning for the project and the cost study.
- Chapter 3 introduces our dataset type, access log provided by Apache servers, and their characteristics.

- Chapter 4 describes design and implementation of the software. On the design part we focus on what shape takes the web traffic, what drawbacks comes with the usage of this kind of log for the analysis and the architecture we have developed to cover this.
- Chapter 5 introduces the concepts of clustering and how we will be using clustering for our goals and how we will represent the results obtained.
- Chapter 6 introduces the concept of finite-state machines and the finite-state machine inference algorithms, how we will be using them and how we will represent the result.
- Chapter 7 covers the experiments we will perform with the provided data-sets and the explanation of the results we have obtained.
- Chapter 8 covers the conclusions and the future work line of this project.
- Finally, appendix A features a study used to define the values of variables used for the tuning of the algorithm explained in Chapter 5.

# Chapter 2

# Project planning

In this chapter we will show the methodology which we followed during the project and the project planning, which includes the planned and final schedules and the deviation between, as well as the project's estimated cost.

## 2.1 Methodology

As the project is going to be developed by only one person, the selected methodology will be a prototype-based work-flow based on Extreme Programming [10] methodology, which aims for simplicity and an effective communication with the client or, in this case, the supervisor of the project. The basis of this way of working is to build constantly prototypes of the working features. Prototype-based work-flow has five steps as shown in Figure 2.1. These five steps are defined as follows:

- Requirement Analysis: Given each problem, we must understand it and analyze which features we must provide. After this process we will have a list of features to design.
- Feature Design: Each feature must be designed in order to provide solution to the proposed problem. The design does not have to be formal, it is a draft for the programmer to know what to do and how.
- Code: After designing, the feature is implemented in the language chosen for the project. At this point external libraries may be chosen to solve the task.
- Prototype Test: Finalized the implementation, we must test the prototype to ensure it works as it has to. From this state we can go to the other ones depending on the situation.

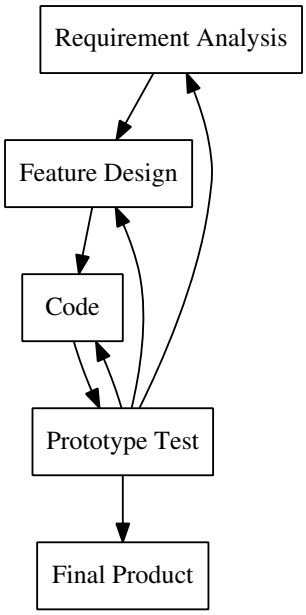



Figure 2.1: Prototype-based workflow

If a bug is found or performance optimization is needed, we have to go back to the code stage. If a bug of another feature is found, we must go back to the code state of that feature.

If the feature is ready and we have another feature to do in the list, we go back to feature design stage to design the new feature. If the current feature needs to be redesigned, we also go back to that state.

If the features from the list are covered and there is a requirement to be analyzed or a new situation, we go back to requirement analysis stage.

If everything is fine, we end the project in the final product stage.

- Final Product: Finally, we will obtain the final product, which will be the one served to the client.

An important detail of this way of working is that the communication with the supervisor is constant. In Prototype Test Stage, the supervisor is reported with the changes, what works and what problems have been seen.

## 2.2 Schedule

The schedule is based on how many months can we spend for the project. As we started the project at the start-middle of the summer, the schedule starts in July and ends in the first week of January, as PFC's presentations are done during this month.

We prepared the schedule splitting the different tasks in months and giving months from September to December the same number of hours. The hour estimation is done with the credits/hour ratio. There have to be about 20 hours of work per credit, therefore 37,5 credits are 750 hours.

| **Month** | **Work to do** | **Man-hours** |
|---|---|---|
| July 2013 | Knowing problem. Design. Parser. | 40 h. |
| August 2013 | | 80 h. |
| September 2013 | Knowing the problem, design and parser implementation. | 145 h. |
| October 2013 | Data structure implementation and graphical representation. Clustering study, implementation and testing. | 145 h. |
| November 2013 | FSA inference study, implementation and testing. | 145 h. |
| December 2013 | Experiments and report writing. | 145 h. |
| January 2014 | Project presentation. | 10 h. |
| | | **750 h.** |

Table 2.1: Time study

The hours are divided in an incremental way from July to September. From September to December, we will work 145 hours per month which are around 6 hours per day working 6 days of the week.

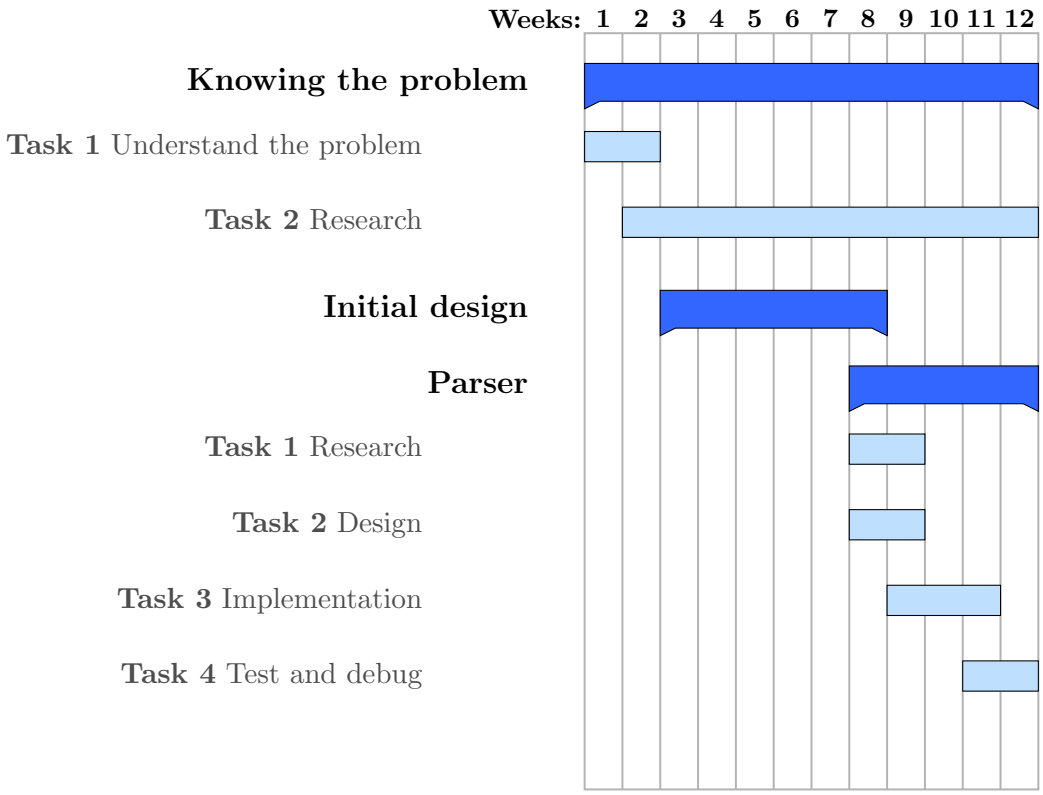



Figure 2.2: Schedule for July 2013-September 2013

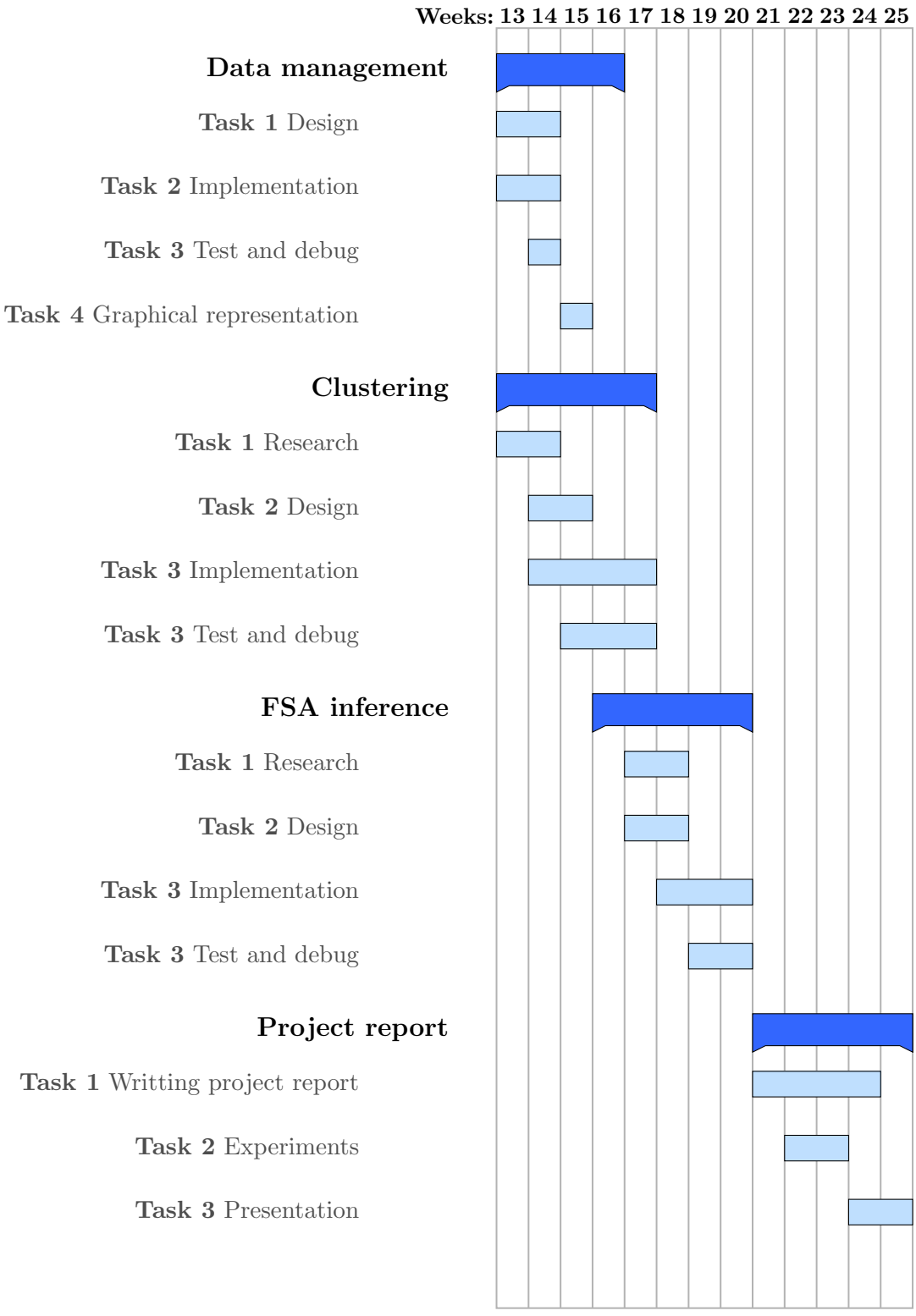



Figure 2.3: Schedule for October 2013-January 2014

## 2.3 Final schedule

This is the real schedule that we have done in the project. We show the deviations and timing of each task.

**Deviations** Several deviations from the original planning were made. The following list illustrates the reasons:

- Japanese course: I toke lessons of japanese at Escuela Oficial d'Idiomes (EOI) starting on September. This would have been correctly foreseen if the difficulty and the hours needed for this was the same as in the previous course. However, Japanese 3B proved to be even more difficult than the previous 3A. It was foreseen that it would take 5 hours per week for classes and 5 hours per week to study. However, there were the double of weekly exams in this course, leading to afternoons of study. Per week around a day or two of work was lost depending on the week. Because of this reason, I left japanese to dedicate more time to the project around last week of November. Looking at the hours done, we can see that the hours spent on the project goes up.

- Part-time job: I started working in a part-time job at the campus for 4 hours/day starting on March. This was notable on the first month of work. However, I could get handle of it during the following months. We did not foresee the part-time job in the original schedule.

- Extra curricular tasks: I am a very active person in the associative life of the campus, which keeps me busy some times. In this subject I could not foresee that I was going to be needed in the FIB's student magazine for the full design and layout adjustment of the two magazines printed on this curricular year. This made me slowdown during December and March-April. In this task I used around 120 hours of my time.

- Parser: The parser took more than 1 month to do because of new extensions we added that were not planned but were useful for the project.

- Data: The data classes were unexpectedly hard to manage and lead to a lots of bugs which required days of debug. In fact, these classes were the ones that caused more deviation in our project.

- Clustering: This part was more or less adjusted in time to the foreseen schedule. However, the problems with data block slowed down the

development and some problems with the clustering itself slowed down the development.

- FSA analysis: This one took a bit longer than foreseen because of the previous slowdown due to of data classes and clustering. Also, we tried to implement Treba as a library, but we could not do it. This process wasted about a week of work.
- Code optimization: This part of the project was not in the foreseen schedule. However, the process of optimization was worth of it as we got a much better performance.
- Project report: The expected time for a PFC report to be done is around one month but in this case took over 2 months to complete this document due to my inexperience on writing this kind of report. Moreover, English is not my mother language, so expresivity is less fluent.

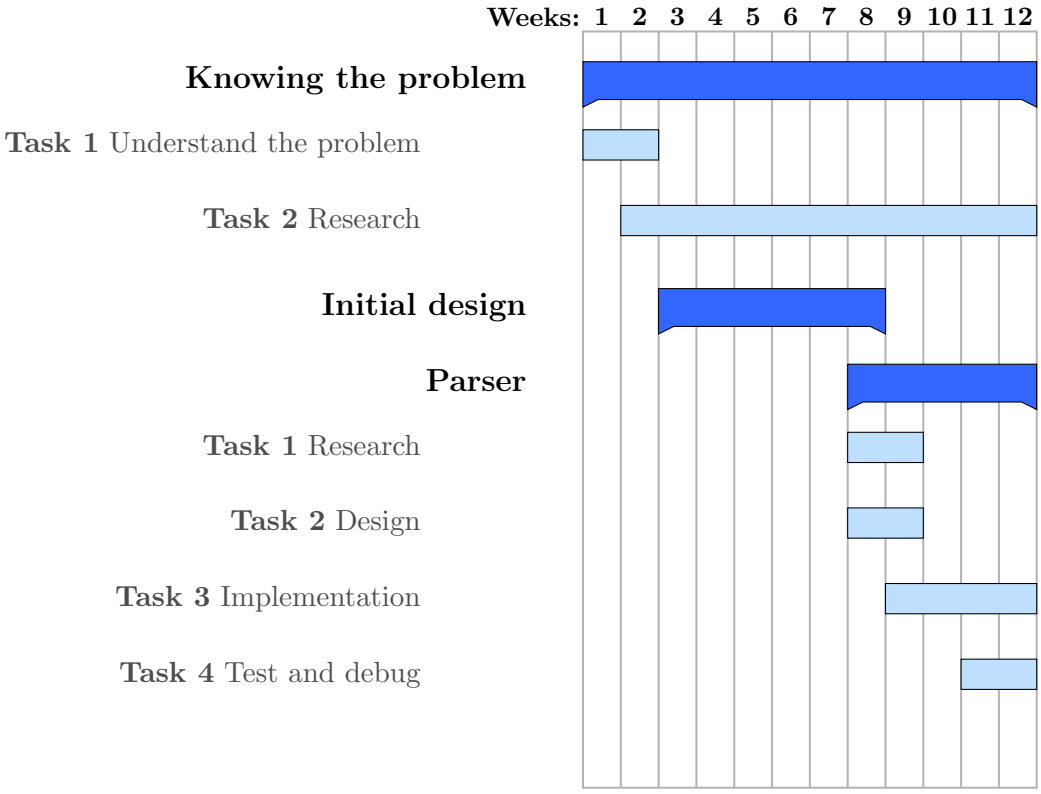



Figure 2.4: Schedule for July 2013-September 2013

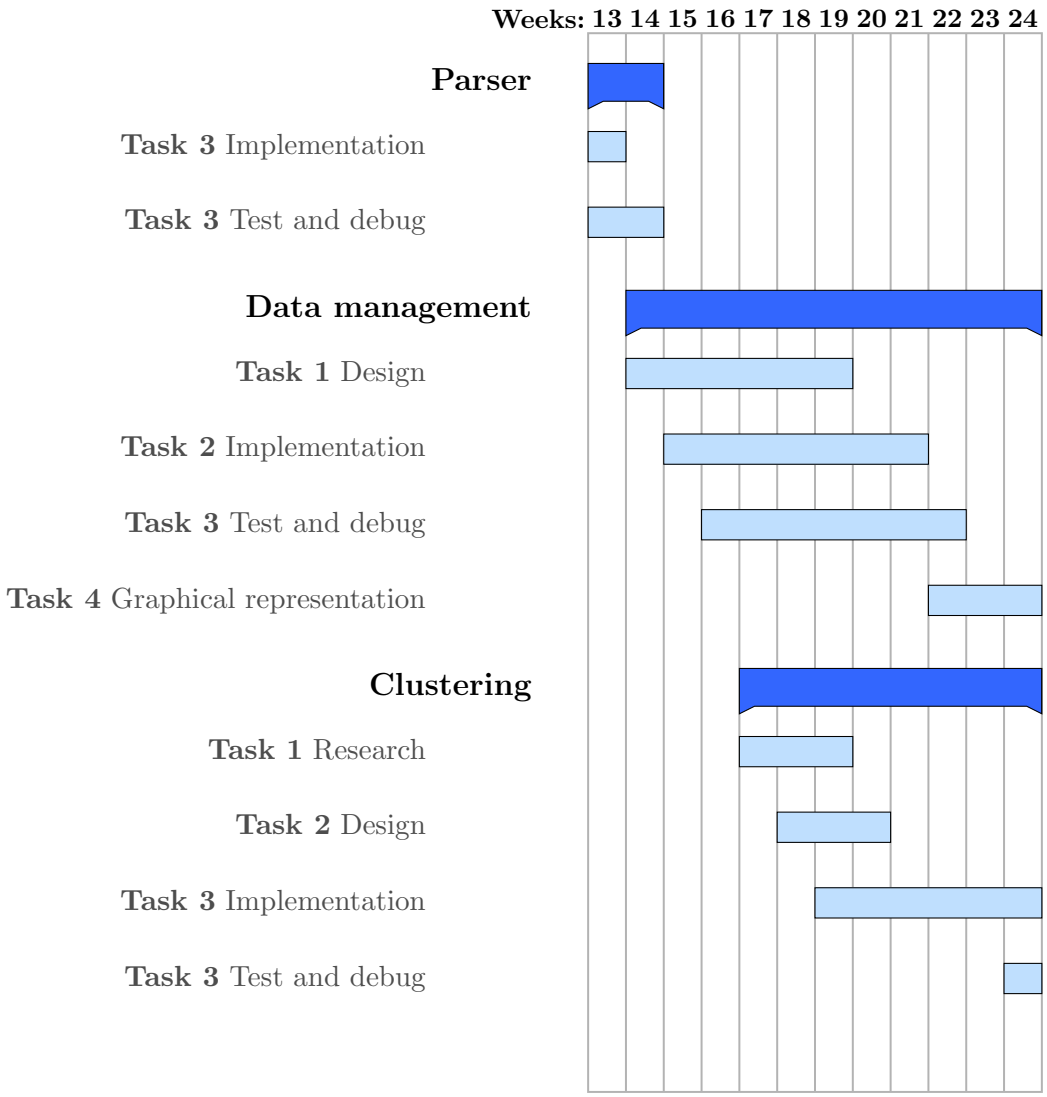



Figure 2.5: Schedule for October 2013-December 2013

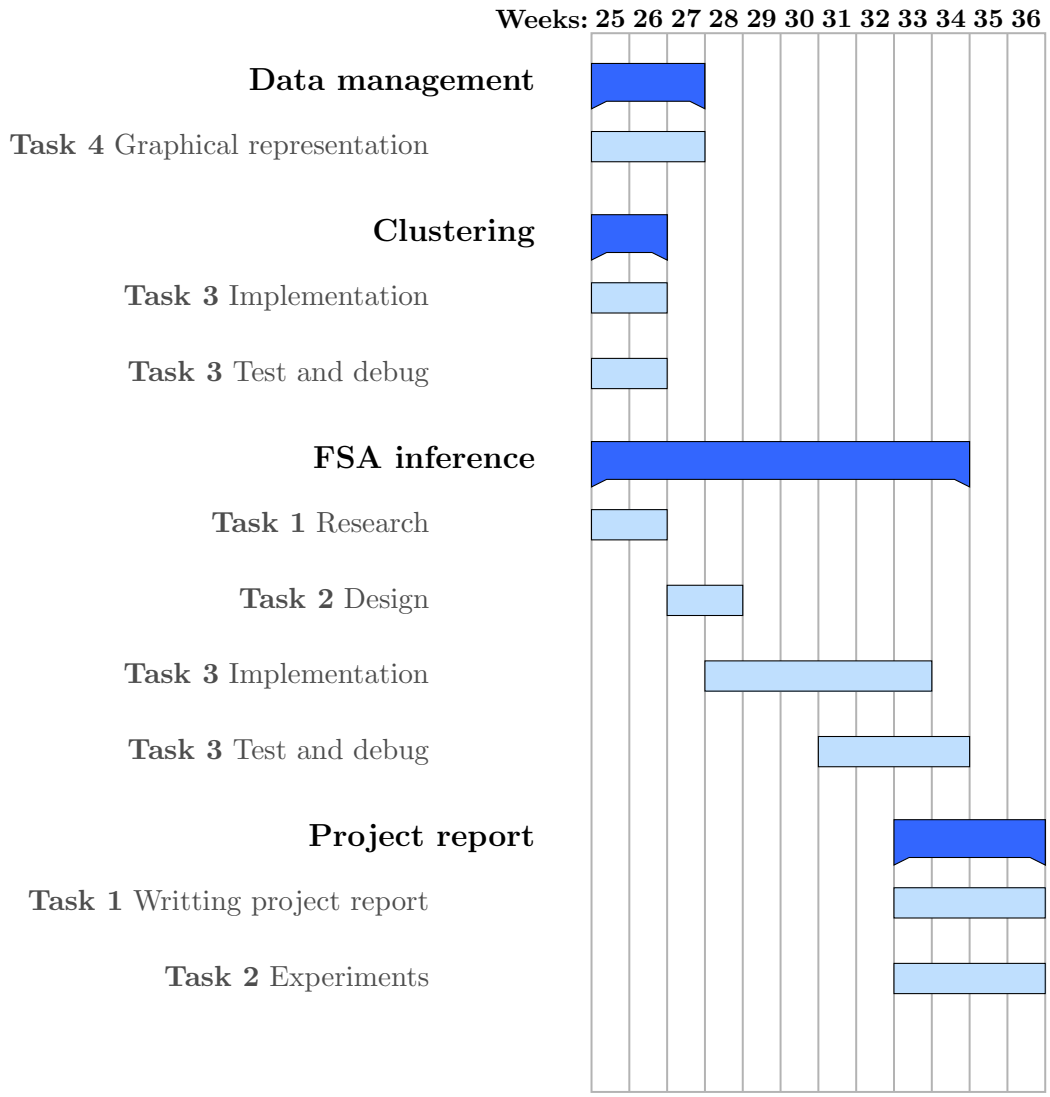



Figure 2.6: Schedule for January 2014-March 2014

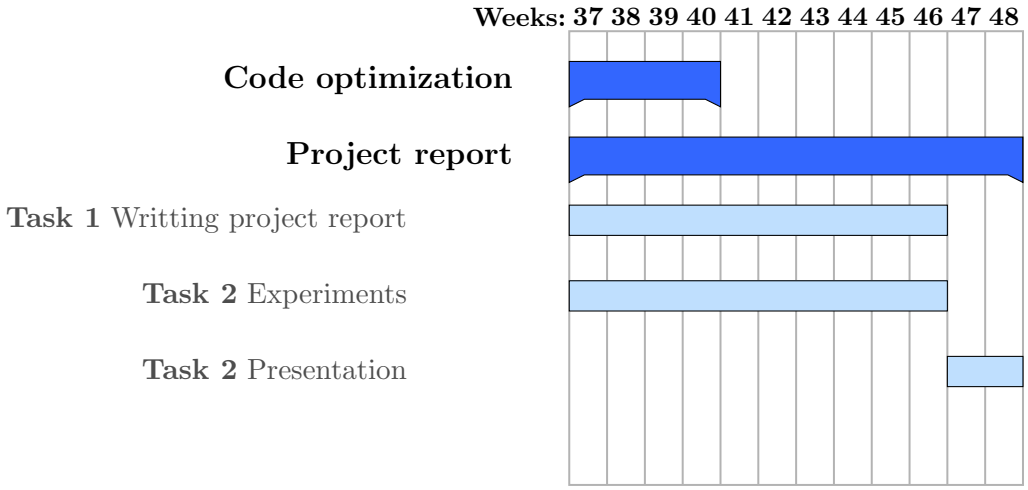



Figure 2.7: Schedule for April 2014-June 2014

## 2.4 Cost study

To calculate the cost of the project we will consider two factors: The tools required for the project and the working hours.

In this part of the cost analysis we consider the tool used for development, the software required and computers used for the calculations of the project. We will consider the two computers and a server that were used for project development and the server where some executions were done. The laptop has a cost of 600€, the PC has a cost of 800€and the server has a cost of 3000€. These machines have a useful life of 4 years. Both computers were used for a year and the server was only used for a month. This means that the cost of the machines will be amortized in 4 years and we will divide the cost taking that into account.

| **Tool** | **Cost** |
|---|---|
| Laptop | 75€ |
| PC | 100€ |
| LARCA Server | 62.50€ |
| Software tools | Free |
| **Total tool cost** | **237.50€** |

Table 2.2: Tool cost study

| Month | Work done | Man-hours |
|---|---|---|
| July 2013 | Knowing problem. Design. Parser. | 40 h. |
| August 2013 | | 80 h. |
| September 2013 | Parser. Implementation of data structure | 75 h. |
| October 2013 | and graphical representation. | 80 h. |
| November 2013 | Data structure. Clustering implementation. | 80 h. |
| December 2013 | Clustering and result representation. | 85 h. |
| January 2014 | Clustering debugging. FSA inference. | 90 h. |
| February 2014 | FSA inference implementation and debug. Code depuration. | 110 h. |
| March 2014 | Debugging and experiments. | 80 h. |
| April 2014 | Code optimization, experiments, project report writing. | 100 h. |
| May 2014 | Experiments and report writting | 115 h. |
| June 2014 | | 50 h. |
| | | **985 h.** |

Table 2.3: Time study

| Month | Analyst | Programmer | Tester | Total hours |
|---|---|---|---|---|
| July 2013 | 40 h. | 0 h. | 0 h. | 40 h. |
| August 2013 | 80 h. | 0 h. | 0 h. | 80 h. |
| September 2013 | 10 h. | 40 h. | 25 h. | 75 h. |
| October 2013 | 20 h. | 40 h. | 20 h. | 80 h. |
| November 2013 | 20 h. | 40 h. | 20 h. | 80 h. |
| December 2013 | 15 h. | 50 h. | 20 h. | 85 h. |
| January 2014 | 20 h. | 40 h. | 30 h. | 90 h. |
| February 2014 | 0 h. | 70 h. | 40 h. | 110 h. |
| March 2014 | 20 h. | 20 h. | 40 h. | 80 h. |
| April 2014 | 20 h. | 50 h. | 30 h. | 100 h. |
| May 2014 | 80 h. | 0 h. | 35 h. | 115 h. |
| June 2014 | 40 h. | 0 h. | 10 h. | 50 h. |
| Total | 365 h. | 350 h. | 270 h. | **985 h.** |

Table 2.4: Time study with roles

For the working hours we have divided the roles of the workers in the following:

- Analyst: The person in charge of the study, analysis and design of the problem. The cost of this role is 75€/hour.

- Programmer: The person in charge of the implementation. The cost of this role is 50€/hour.
- Tester: The person in charge of testing the software and doing the experiments. The cost of this role is 50€/hour.

The cost of each role includes including taxes and costs related to electrical power comsumption, taxes, campus costs and so on. The number of hours would be described as seen in Table 2.3 and the number of hours per role in Table 2.4.

From the total time used for this project 365 hours belong to the analyst, 350 hours to the programmer and 270 hours to the tester as shown in Table 2.4. In euros would be 27375€for the analyst, 17500€for the programmer and 13500€for the tester. This human resources or engineering costs sum a total of 58375€. When considering all costs together, the final cost calculation is 58612.50€, as shown in Table 2.4.

| **Concept** | **Cost** |
|---|---|
| Engineering costs | 58375€ |
| Tool costs | 237.50€ |
| **Total project cost** | **58612.50€** |

Table 2.5: Total costs

# Chapter 3

# Logs and parsing

A log is a file that contains the actions that have occurred in a system. Web servers maintain log files listing every request made to the server. With log files, it is possible to get a good idea of where the visitors are coming from, how often they return, and how they navigate through a site. In this section we will introduce the architecture of web servers in terms of logs, the logs we are going to use and how we are going to parse them.

## 3.1 Web server log architecture

Logs are made up of information gathered by default, in most cases, by the applications themselves. Reviewing the logs is an standard way to gather data as it is included by default in many applications. There are several levels of applications lying under a website. The following diagram of the Figure 3.1 represents the general case.

There are several logs in a website. The web server, databases and the application servers used by the website, each one keeps a log of their activities where the administrator can observe whether everything is alright. Web servers have two types of logs: one that informs of the errors occurred in our system and another that registers how the server is accessed. For this project we use the access log for simplicity, as it provides enough information for this moment, but we can consider in the future using error logs to enhance our model.

The access log is a good way to obtain data. However modern web browsers have built-in caches to save traffic and load to the server in order to give a good user experience, storing the visited pages and contents in the client side. This means that not all the actions that the user perform in a website may be recorded in the server side. If a user visits twice the same

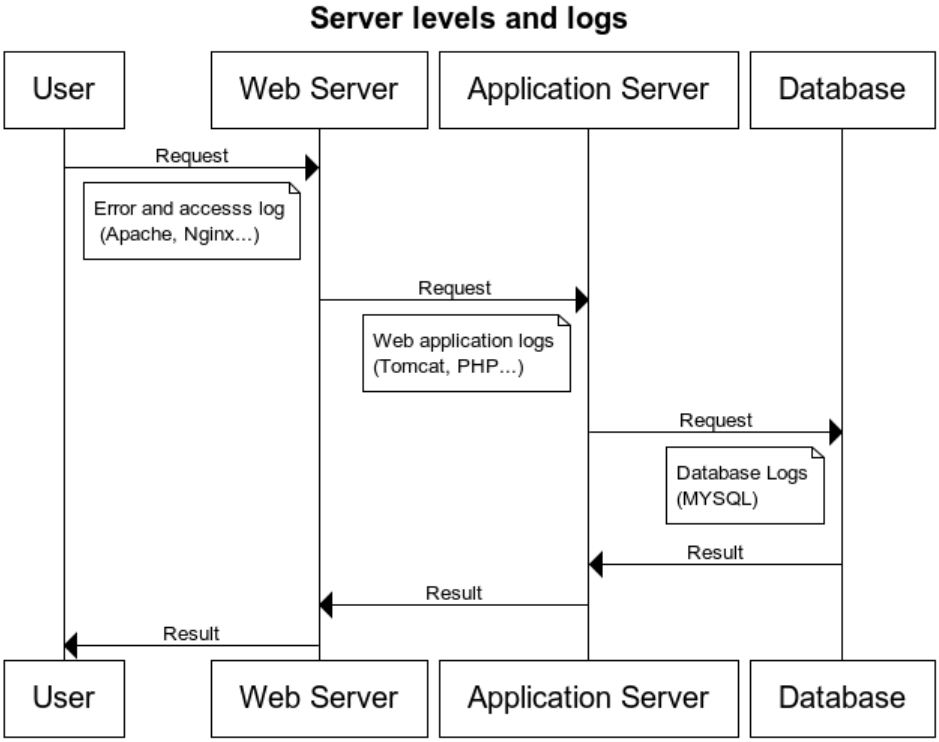



Figure 3.1: Flow of a web request and how it affects to logs.

page in a short period of time, small enough to use the page stored in the cache, the web browser may consider using the cached version, so the request will stay in the user side and never comited to the web server.

In the *top 3* ranking of web servers [11] we can find Apache, Nginx and Microsoft Internet Information Services (IIS). As seen in the ranking, most of the websites around the world are served using Apache WebServer, we will be using this format [12] for this study as it covers more market that the other two. In addition, Nginx's log format [13] is almost equivalent to Apache. So, without having to change anything, we can cover both cases in a general way. In the other hand, the format of IIS [14] differs from the Apache/Nginx format, so we will not cover it in the scope of this project.

## 3.2 Apache logs

The Apache log format is highly customizable. It can be configured using a string pattern where the administrator can select which variables to record in the log, as seen in Apache website [12]. Moreover, there are templates of

possible formats in order to make it easier for the administrator to choose if they do not need anything special. The most frequently used are these three:

- Common Log Format (CLF)
  - Remote host Internet Protocol (IP) address
  - Remote logname
  - Remote user
  - Request timestamp
  - Request (with quotes)
  - Status
  - Size
- Common Log Format with Virtual Host
  - Virtual host
  - CLF parameters
- NCSA extended/combined log format
  - CLF parameters
  - Referrer
  - User-agent

From these three standard formats, only one is valid for the scope of this project. The NCSA extended/combined log format is the only one which has a referrer or reference. This parameter is required for our project because we need to know where each request comes from (from which web site or page), not only the content requested. Moreover, this format is set by default in Apache (depending on the distribution. In other distributions it may be set to common).

**Subdomain problem** If we use combined format, detecting subdomains becomes an issue. More than one website can co-exist together inside a server without having the same domain. For example, we can have *oasi.upc.edu* and *distorsio.upc.edu* (two magazines by UPC students) in the same server. The server receives the request and, based on the domain specified in the received URL, the user is served with one content or another. This concept is called *name-based virtual hosting*.

*Name-based virtual hosting* enables the capability of hosting multiple domain names on a single server. This allows one server to share its resources without requiring all services provided to use the same host name. The server has a configuration file where all the different domain names are registered, also indicating how they have to be treated. This feature is included since HTTP version 1.1 (and may be found in 1.0 protocol with 1.1 extensions).

In terms of logs this have a direct impact. The administrator, at least in Apache and Nginx servers, can set a log for every different virtual host. In this case, it would be like a normal access log. In the other hand, if the access log is shared by more than one virtual host we need a parameter (from the parameter set for the customizable log format) that tells us to which domain belongs each entry. This kind of format is not in the scope of this project but could be considered future work. At this moment, the project will treat the various domains co-existing in the same server as it is only one domain.

Other domains can point to our domain, this can cause the referrer to be external, which is a totally normal situation. Most of the time people come from search engines like Google. They usually come from highly related websites or even other domains from the same server. In order to cover this case, we have introduced the *interesting domain filter*, which is a filter that, when matching, marks the entry referrer as interesting. This way we add the domain name as father of a new traffic branch. This can be useful, for example, if we want to distinguish which users come from Google and which come from other places, and whether there is a correlation between the traffic made and the origin of these users. The domains not covered by the *interesting domain filter* are set to "-", which means "no reference". We have done this in this way in order to treat all users the same way independently from where they come, with the exception of domains and referrers we consider relevant to be know.

## 3.3 How to parse

For the parsing of the document we are using a structure defined in Section 4.3, designed by looking at which information can be obtained from the log. In this section we will define how the parsing is done, independently of the structure.

The structure of the lines of logs can be defined with a grammar expressed in Backus–Naur Form (BNF) as follows:

⟨*line*⟩ ::= ⟨*remote-host*⟩ ⟨*ws*⟩ ⟨*remote-logname*⟩ ⟨*ws*⟩ ⟨*timestamp*⟩ ⟨*ws*⟩ " ⟨*request*⟩ " ⟨*ws*⟩ ⟨*status*⟩ ⟨*ws*⟩ ⟨*size*⟩ ⟨*ws*⟩ " ⟨*referrer*⟩ " ⟨*ws*⟩ " ⟨*user-agent*⟩ "

⟨*request*⟩ ::= ⟨*method*⟩ ⟨*ws*⟩ ⟨*content*⟩ ⟨*ws*⟩ ⟨*protocol*⟩

⟨*referrer*⟩ ::= ⟨*serverURL*⟩ [⟨*content*⟩]

⟨*content*⟩ ::= ⟨*resource*⟩ [? ⟨*param*⟩ = ⟨*value*⟩ {& ⟨*param*⟩ = ⟨*value*⟩}]

In this grammar <ws> represents the whitespace token which we will use in Algorithm 1 to split the line. Moreover, we have used two extensions of BNF syntax for the definition: [] which means that everything in-between is optional and {} which means that what is in-between can be repeated zero or more times. Except the request and the referrer, which is formed by three different parameters, the other tokens are saved directly into the structure. Depending on the token, there is a syntax checking process to avoid malformed requests.

<Referrer> and <content> are almost the same except from the token <serverURL>. This token is left out in <content> as we know on which server we are. As <content> may have parameters, we must parse them. Algorithm 1 describes the parsing process.

**Algorithm 1** Document parsing procedure

**Input:**Access log

1: **for** each line in the document **do**
2: Create a new entry for the structure.
3: Split line in tokens using space as delimiter.
4: Parse first token (IP address) and save it into the structure.
5: **if** Format is incorrect **then**
6: Entry is set as not valid.
7: **end if**
8: Set fourth token (timestamp) in the structure.
9: Parse fifth token (request). Set method and protocol to the structure. Parse content and the parameters it have and set them to the structure.
10: **if** Content is in our filtered list **then**
11: Set entry as not valid.
12: **end if**
13: Parse eight token (referrer).
14: **if** Matches with our interesting site filter **then**
15: Set domain of that site as referrer.
16: **else if** The referrer comes from our domain **then**
17: Parse parameters of the content and set the content as referrer.
18: **else** Set "-" as referrer.
19: **end if**
20: Set the ninth token (user-agent) in the structure
21: **end for**

We will use an example in order to illustrate better the parsing:

```
111.111.111.111 - - [06/Mar/2014:18:24:53 +0100]
"GET /content/image.png HTTP/1.1" 206 1299
"mydomain.edu/index.html" "Mozilla/5.0"}
```

In this case *111.111.111.111* would be the IP address, which we will be using to identify the each client. The following - are the remote logname and remote user. These two take no effect when we are not logged in the system. The following parameter is the timestamp, which we will be using in case we need to reorder entries in our system if needed. After that we have the content, starting with the method, in this case GET, which is a method used in HTTP[1] in order to retrieve whatever is identified by an Uniform Resource Locator (URL). Inside the same quotes of the method, we have the content requested, */content/image.png*. Then, we see that protocol used for this request is Hyper Text Transfer Protocol (HTTP) version 1.1. After that, we have status 206, which is HTTP's returning code. After that we have the size of the returned content which is, in this case, 1299 bytes. Finally, we have the referrer and the web browser or software used to do the request. Here, the content "image.png" was requested from the page "index.html" of the domain "mydomain.edu" using a Mozilla browser.

How the filter works and its implementation is explained in Section 4.2.2

## 3.4 Conclusions

There are different web server applications with their log files; however, with a simple parser we can cover a big portion of the websites. Moreover, logs provide sufficient information to do the study we propose in this project.

In the following chapter we see details of design and implementation. In that chapter we have special attention on traffic shape and the benefits and drawbacks we have in this subject. Some of these problems are introduced because of the usage of logs, leading us to obtain data which may be not 100% precise.

[1] We expect to receive the following methods: GET, OPTIONS, HEAD, POST, PUT, DELETE, TRACE and CONNECT as defined in the standard of HTTP [15].

# Chapter 4

# Design and implementation of the tool

## 4.1 Use cases

As the tool though as a batch query program, it is straightforward. The user only have one, *do analysis*, use case which is described as follows:

- Select a data-set to analyze.
- Select the desired algorithm to use.
- Put the desire parameters.
- Select an output folder.
- Execute and retrieve results.

This use case is used for both analyses or techniques. If we need to implement more algorithms, our framework would not change as we have designed it to benefit from changeability, using classes for each algorithm and adapting the input inside each class.

For each algorithm, a different set of parameters is available. We will explain the different parameters in Chapters 5, for clustering analysis, and in Chapter 6 for FSA inferente analysis.

## 4.2 Architecture

The architecture of this project is strongly focused on how data is represented and how it is transformed as transforming data is our main task and interest.

During the execution of the program, data shape changes on every step of the program. In order to have a modular system to boost changeability of parts of the program, we have defined four different blocks as can be seen in Figure 4.1. Moreover, we will introduce the filtering system.

### 4.2.1 Data architecture

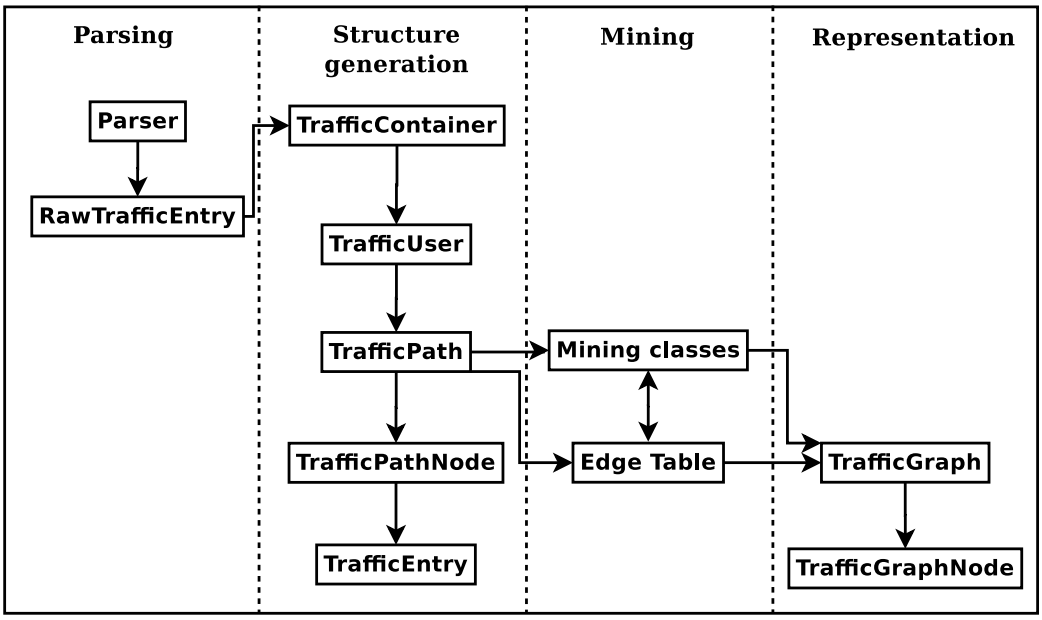



Figure 4.1: Diagram of the stages of data transformation.

The first stage of data is the log. We have to extract data from logs and transform it into a structure that we can use inside our program, the *RawTrafficEntry*. This structure is a simple structure that contains every piece of information that can be obtained from each line of the log using Algorithm 1 explained in Section 3.3. We will not be using every piece of data that the log contains; however, this process is useful to decouple parsing and the specific data structure of our project. Moreover, this way we modify which data goes to the further processes and which not without modifying the parser.

Retrieved all the data from the log in various *RawTrafficEntries*, we build the general structure with only the information used in our application. This structure is formed as a hierarchy of classes. The one that contains everything is the *TrafficContainer*. The *TrafficContainer* can contain every user of the dataset, each represented with the class *TrafficUser*. This *TrafficUser* works

as interface for the *TrafficPath* but also has information to identify the user (the IP address in our case).

Inside *TrafficPath* class, the *NReverseTree* is implemented in the way defined in Section 4.3.1. This structure uses *TrafficPathNode* as node information, in other words, contains the information of a given visited page of the traffic done by a user. This *TrafficPathNode* includes all the information saved in *TrafficEntry*, which is a similar structure to *RawTrafficEntry* but containing only data that is useful for the project.

Given this data structure, we can use a *TrafficContainer*'s method to get, for each user, the *EdgeTable* representation. As seen in Figure 4.2, this representation is meant for doing the mining in the clustering process, as using directly the *TrafficPaths* is less efficient. The clustering process uses *EdgeTables* as input and outputs the representation of the groups as *EdgeTables*. The *EdgeTable* structure is introduced in Section 5.3.1. For the FSA inference we will use directly the *TrafficPaths* as input.

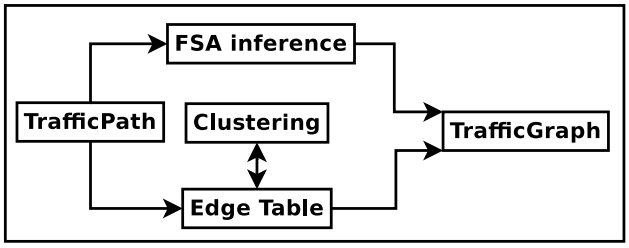



Figure 4.2: Diagram of the mining classes and its relationships.

Finally, the mining results are represented with the class *TrafficGraph* which is, in essence, a graph structure. We use this structure and not the *NReverseTree* as results are not acyclic, thus, they are not trees.

### 4.2.2 Filtering and missing values

*Filtering* is a feature from the project which offers the user control about which pages of the web site are important for the analysis. Pages and content can be ignored at parsing time by using a filter that can be configured by the user.

The filter is a regular expression created in run time using a file from the configuration folder. In this file, the user can write the names of the pages or content in order to ignore them. The file format is simple: for each line of the document a name that identifies pages or content can be specified, either writing the full name or using the wildcard. For example, we may want to ignore every image on our website. If we only have Joint Photographic

Experts Group (JPEG) and Portable Network Graphics (PNG) images on our site, we can accomplish this putting in the file a line containing **.jpeg* and a line with **.png*.

The most common reason to do so is that there is content that does not give information because it is demanded every time a page is accessed. This may be the case of favicon, CSS and images related to page style. If we filter this kind of content, the number entries may decrease, leading to a faster and more accurate analysis. Moreover, we may want to know how the users behave in some part of the web, using the filter to ignore entries not related to that part. Also we can use this filter to refine our analysis using results from previous ones.

Another feature offered for traffic is the treatment of *missing values*. Usually, an user enters our web by two means: "no reference" ("-") or by another web site (Search engines, sites that link to our...). However, there may be cases that users start surfing our web site having our web site as referrer. For example, an user can access to content1 coming from page1 that we own. If everything was correct we would have an entry of the user demanding the page1, however this is not true all the times. A cause we have detected for this kind of problem is that logs are finite. In order to not have very big files and save disk memory, logs are rotated with tools like *logrotate* (Linux). This kind of tools are normally used periodically. They compress the current log and saves it with different name with a sequential identification number. As text can be easily compressed with a good compression rate, a lot of space is saved. In the other hand, as logs are cut, information can be split between two logs. For this case, the start of traffic may be in one and the end in another.

From between ignoring the entry and fixing the missing values take, we the second option. When we find a case like this, we add an entry with requested content the referrer and set as referrer for this "fake" entry "no reference". If this user inserted this way is an outlier, it can be detected by just looking at the resulting model, so it is not a big issue treat missing values this way.

## 4.3 Data structures: User traffic shape and its issues

In this section we focusing in the design in terms of data structures as the data transformation is the most important process of the project, as it is our main goal, but we also introduce the other parts of the project. As described

in Chapter 3, our input will be logs produced by server applications. Given this, we will explain problems found and design decisions we have taken because of the usage of the traffic logs.

At first one may think that sessions follows a linear path because the user starts at one point, in other words a web page, and ends in another; however, we have observed that this is not true for every user. This linearity is one of the shapes that traffic may take, but the structure which defines well all cases is an *n-ary tree*. We show how this happens in three example situations:

- In first instance, we have observed that an user may have more than one traffic session opened. One user may surf our website from 10AM to 10:20AM and, then, at 6PM come back in order to search new information in a different place of the site. For example, we can have a computer shared by a family used by the parents at the morning and the kids at the afternoon. As there is a considerable different amount of time between the two, they may be considered two different sessions. These sessions, which can be produced by the same end-user or not, cause our traffic to have branches.

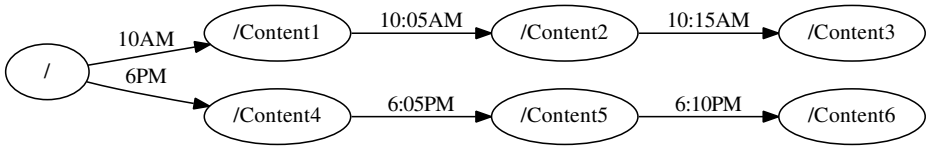



Figure 4.3: Example of traffic of the same user at different time.

- Close to this first problem we have the tab opening problem. Nowadays, when we surf on the Internet we may want to see more than one part of a website at time, hence, opening new tabs for each part of the web. This will create even more branches in our traffic.

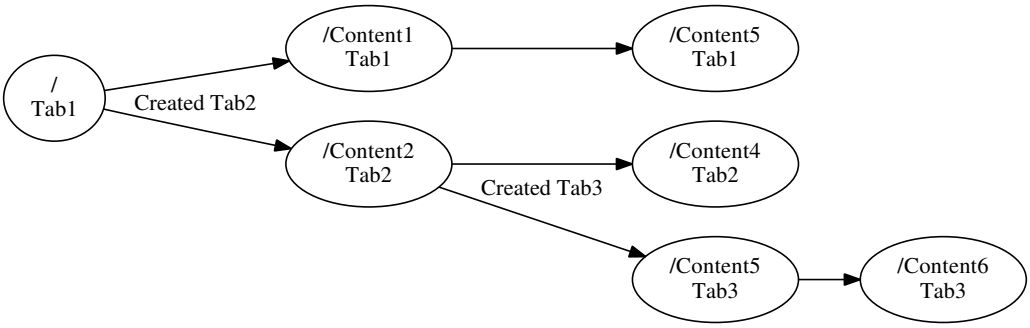



Figure 4.4: Example of an user that uses tabs.

- Last but not least, we have Network Address Translation (NAT). NAT is a mechanism that allows a router to send traffic in and out given an internal IP address and a port. In other terms, the router remaps an IP address into another. This is used, for instance, in home networks. In our homes it is likely to have only one IP address given by our Internet provider but, probabily, we have more than one computer connected. As the mechanism to recognize each user is just the IP, we can not know if there are more than one users under the same IP address. This mechanism will surely create new branches in our sessions as this users may want different things of our website.

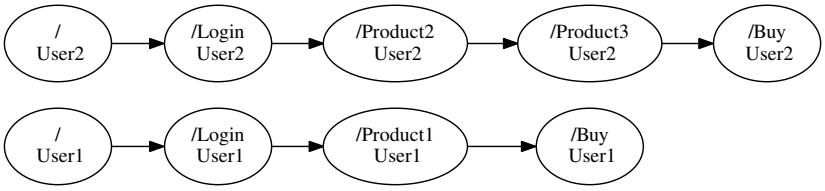



Figure 4.5: Example of two users under the same router.

The two first factors affect only to one computer. However, last factor can affect an entire home network. For each computer in the Local Area Network (LAN) affected by the third factor we may have the two other factors, making our traffic to have even more branches.

There is one more problem to solve in order to have an accurate data structure to represent the traffic, the causality between content petitions. A good example to represent this issue is a computer which is used at 10AM and 6PM. At 10AM the user access to page A, then B and then C in our website. Then, at 6PM, access to page A, then B and, finally D. At the point when we insert D, we will have two different branches which end in the same page, C. We are most likely to think that this new entry produced by the request of D belongs to the last branch added and henceforth we defined the causality criteria that will be introduced in the Section 4.3.1.

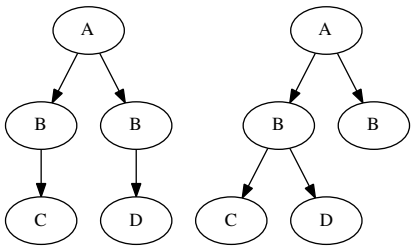



Figure 4.6: Example of two different trees with node D on different B node.

Given these factors on our traffic characteristics, we build a *n-ary tree* structure with some additional features, having a good representation of our traffic. The representation of the traffic can be also done with a Directed Acyclic Graph (DAG), which is a more general type of structure that contains trees. Using this kind of structure, we can fuse page nodes that represent the same page as seen in the Figure 4.7.

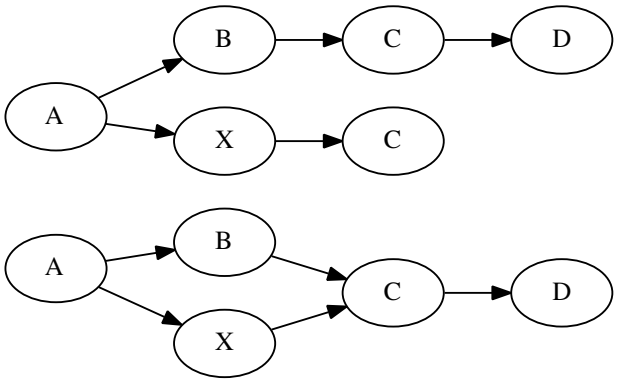



Figure 4.7: Two different representations of a traffic path. A tree versus a more general DAG.

As we can see, in the tree structure we have 6 nodes and in the DAG we have 5. This way we have less nodes, thus, structure is lighter. However, with this compression comes a drawback: the meaning of node D may be different if B or X are visited. For example, we have an on-line shop were we sell Bs and Xs. Our main page is A, which is visited by every user. Then, users interested in buying Bs or Xs goes to B or X pages. C is the page that shows a disclaimer that says the user must accept some terms in order to buy that item. Finally, D is the page were the user does the checkout (purchases the items in the cart).

With the tree model we know that the user has bought only B. However, with the DAG with fused states we can not know if the user has bought B, X or both.

### 4.3.1 NReverseTree class

Seen the problem of non lineal traffic representation and the necessity of a *n-ary tree* structure which allows us to store the traffic information without losing information, we propose a structure that aims to solve these problems.

Our approach implements a specific structure, that we have called NReverseTree, as it is a *n-ary tree* representation maintaining the causal order.

This structure is a graph, implemented as a linked node structure where each node has a father and an arbitrary number of child nodes, building a directed tree structure. The first node inserted on it, which we consider as the landing page of a user in the analysed system, is denoted as the *root node*.

In this class we maintain two different lists: causal order list and leaf node list.

- The causal order list is the reference to know in which order nodes are inserted. For insertion, the program will search, using the *reverse find function*, the node's father in this list in a reverse fashion, starting at the end of the list or, in other words, the last node added to the structure. This is done because we want the "youngest" father in order to deal with causality problem. Here we take the assumption that between two possible referrer nodes, a node is more probable to be child of the last possible parent visited, as is mostly probable to be the last on the users focus of attention. We will illustrate this with a example further in this section.

- On the other hand, the leaf node list makes reference to the nodes that have no child. We need this in order to know every path taken by each user in a reasonable time. Given this list, we can extract every path in a linear way, also called session, by travelling from a node to its father until the root node is reached. We will obtain one path for each leaf node.

A precondition for the building process to build the tree correctly is that entries must be ordered by timestamp. For each child, we need to have the father already inserted as its a tree structure. The building process is explained in the Algorithm 2.

**Algorithm 2** NReverseTree building process

**Input:**Set of traffic entries

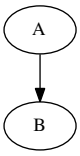
```
1: Set the first entry as root node and add it to causal order and leaf node list
2: for each entry in the set excluding the first do
3:     Search for the father specified in the referrer using reverse find funcion.
4:     Create a node with the entry and link it with the father.
5:     if Father is in leaf node list then
6:         Remove from list.
7:     end if
8:     Push back node to leaf node list and causal order list.
9: end for
10: return Built NReverseTree
```

We will give a simple example based in the one given in the previous Section 4.3. We define the following edges that will be inserted one by one in order to describe how does the process work. The edges will be inserted in the following order:
A→ B, B→ C, A→ C, C→ D, A→ D, C→ D, D→ E

First we insert the first node into the structure and set it as our initial or root node. We will insert node A.

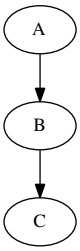

| | |
|---|---|
| Causal order | $A_1$ |
| Leaf node list | $A_1$ |

Then we will insert B node as child of A.



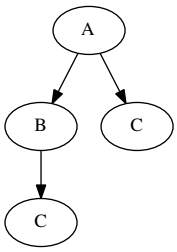

| | |
|---|---|
| Causal order | $A_1$, $B_1$ |
| Leaf node list | $B_1$ |

And then, we add C as child of B.



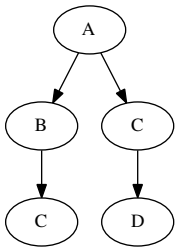

| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$ |
| Leaf node list | $C_1$ |

Now we will insert C.



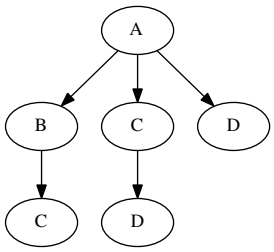

| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$, $C_2$ |
| Leaf node list | $C_1$, $C_2$ |

Up to this point, causal ordering have not really been used as there were no collisions. However, now we have to insert D having C as father. Here we have to take a look at casual ordering list and take the last valid element. In this case is $C_2$, in other words, the C whit father A.



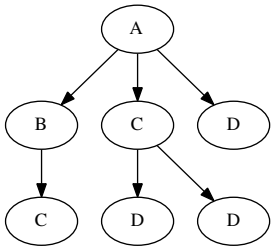

| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$, $C_2$, $D_1$ |
| Leaf node list | $C_1$, $D_1$ |

Then we insert D with father A, so we have 2 D at leaf node list.



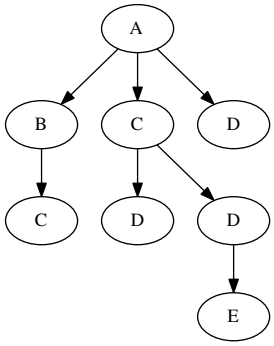

| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$, $C_2$, $D_1$, $D_2$ |
| Leaf node list | $C_1$, $D_1$, $D_2$ |

Finally, we insert another D with father C. Here we have 2 possible fathers, $C_1$ and $C_2$. Being $C_2$ the most recent element, we use that node as father.




| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$, $C_2$, $D_1$, $D_2$, $D_3$ |
| Leaf node list | $C_1$, $D_1$, $D_2$, $D_3$ |

After that we insert the final element an E with father D. At this point we have three possible fathers, being $D_3$ the most recent.




| | |
|---|---|
| Causal order | $A_1$, $B_1$, $C_1$, $C_2$, $D_1$, $D_2$, $D_3$, $E_1$ |
| Leaf node list | $C_1$, $D_1$, $D_2$, $E_1$ |

## 4.4 Implementation

In this section we define some implementation details. As the design and the implementation share most of the class structure, we will explain only what is new from the original design and interesting things about implementation process, like used technologies and optimizations done during the development.

### 4.4.1 Used technologies

The development was done using various Linux distributions as programming in Linux is more convenient than Windows as it is highly customizable and required applications can be downloaded from the repositories easily. Most of the applications can be found there, however it is not the case of one of the softwares used, Treba, which we introduce in this section.

The programs and libraries used for the project are the following:

- **C++** is an object-oriented language which enables us to use object-oriented methodology and is fast. We have chosen this language as we think we will produce a software with high performance, which is one of our goals.

- **QTCreator** [16] is an Integrated Development Environment (IDE) for C++ projects developed by Trolltech. Usually this IDE can be used for projects that include a QT visual interface but it can also be used for command-line software like the project. This IDE provides a reliable and confortable user interface with debuging, depuration integrated and Git repositories.

- **Boost** [17] is a set of libraries programmed by the Boost community which provides tools to perform various different task. In this case we use three features of this libraries: the graph library for graph representation, the regular expressions library XPressive and a *foreach* macro that is useful to iterate over structures like vectors or maps. We learned how to use Boost Graph Library following the work of Jeremy Siek et al. [18] as website documentation is pretty cryptic.

- **Treba** [19] is an application with a set of algorithms for training, decoding, and calculating with FSA and Hidden Markov Model (HMM). This software is used to do the FSA inference analysis.

- **GraphViz** [20] is a toolset is used to represent graphically graphs. This tool has been used through all the project for debuging tree structure,

representing partial and final results and altogether with Valgrind to study were to improve the performance.

- **GDB and Valgrind** are two handy tools for C++ development. GNU Debugger (GDB) [21] is a tool that enables you to find where the program has crashed in case of exceptions or even follow the program execution for the correction of bugs. On the other hand, Valgrind [22] is a tool with two different purposes in our case: code depuration and optimization. Valgrind can be used to detect memory leaks which is memory that has not be correctly freed. These memory leaks may cause the program to store more memory than the memory that is really been used, which may lead to insufficient memory problems (for example, crashing because there is not enough memory). Finally, we have a tool inside Valgrind called Callgrind, which provides information like how many calls have been done for each function of our code and the time spent on them. With this tool unexpected performance issues can be found. Moreover, using the tool Gprof2dot [23], a visual representation of this calls can be generated in GraphViz's dot format.

- **Git** [24] is a version control software. We have used Git along QTCreator to save the project's code on the Internet and have various versions of it.

### 4.4.2 Class implementation

Now we explain details about class implementation that are not shown in the design. On the design we have parsing, data structure and mining blocks, however we will add a new one in this step. For implementation issues we have defined a new block called *template block*.

In the *template block* we have the *NReverseTree* class which is a C++ template class that allows us to have a n-ary tree with the properties mentioned in Section 4.3.1. This class is used by a *TrafficPath* class which is the one which contains some information about the user and the tree. The main reason to use a template class for *NReverseTree* was that we could implement the process trees, proposed by Joos C. A. M. Buijs et al. [25], for clustering but this idea was droped as we found problems in fusing this kind of structures. As the template class already worked and was very convenient, the *NReverseTree* was left as is. After this point we started to use *EdgeTables*, which have given good results for our purposes.

For the FSA inference process we chosed to integrate an already working system, the previously mentioned Treba, in our system. We have a class that,

given the input which are traces of the traffic obtained from the n-ary tree using the leaf node list, processes the information to feed the Treba program with it, waits for the result and transforms it into the output graph and the output lists. Treba's output is text representing the set of nodes and the set of transitions with their probabilities. As representing this directly with a graph can show a large number of edges going from a node to another, we have taken another path which is to sum the probabilities of each transition from a given node to the other and represent that set of edges as one and write, for each node, a set of lists with the full information of each edge. This way the graph can be seen clearly and none of the information is lost on the process.

The rest of the implementation follows the lines of the design. Further details about the analysis algorithms can be found in Chapters 5 and 6.

### 4.4.3 Optimizations and code debugging

During the development of the project we have had several issues, we have divided them in two categories: bugs and optimizations.

**Bugs** We had several bugs regarding to the graph structures, most of them caused by incorrect setting of pointers. Moreover, memory leaks were found with Valgrind and fixed. For further development, we learned that there is a structure used to prevent this kind of problem, smart pointers and shared pointers. This two classes encapsulate the pointers so the memory is reserved at the creation of the class and freed when the delete function of the class is called.

**Optimizations** In order to enhance the functional requisite that is performance, we applied some optimizations during the development. Using callgrind, we saw that there was an anomalous behaviour inside the clustering class. In that class, we assigned an user to a group or another and, if there was a tie between groups, we selected one at random. Improving this process we won about 30% of performance. Moreover, we found that the distance function of the clustering used a lot of CPU. The reason was that one of the two points passed as parameter was passed by value (the other was the implicit parameter) and not by reference, leading to a masive creation of objects. Fixing this we got 50% more of performance.

Finally, in some parts of the parser we used string splits instead of regular expressions. Regular expressions may cover more field than the splits, however they are quite expensive in CPU time. Using splits improved the parsers

performance in a 35% cut of the parsing time. As mentioned in the Chapter 3, we are covering a very fixed format, therefore is not a great inconvenient to lose flexibility upon performance.

## 4.5 Conclusions

We have seen that the tool only has a use case that acts as a main door for every different analysis that is performed in order to have a changeable system. We introduced the architecture of the system making emphasis on that transformation. We proposed an enhanced n-ary tree as a representation of the traffic given the problems mentioned on the chapter like the ambiguity that logs can have when we have to decide how do we connect the requests. In our case, we have chosen, when an entry can have more than one father, to select the youngest, the newest entry. This is done on the insertion of the proposed structure, *NReverseTree*. Also, we presented the filtering feature and how we may use it.

In the implementation section we have presented the tools that have been vital and handy for our project. We have seen that a structure has to be designed in a way it accept changes if what we thought at the start is not correct or fails at reaching our goal. Finally, with optimizations and depuration we have seen that small things can have a big effect in the whole picture if they are used a lot of times. As can be abstracted from Amdahl's law: "Improve what consumes more time, not the what you see more optimizable".

In the following chapter, we will see the first part of the mining block, the clustering approach.

# Chapter 5

# Learning the User Typology

Our aim is to obtain groups of users that have a similar traffic. This will define the typology of the users of a web site. This section strives to define the first technique used to do the traffic analysis. We will cover the basic concepts of clustering technique, $K$-Means algorithm and how this is integrated in the project.

## 5.1 Introduction to clustering

As seen in Pang-Ning Tan et al. [26] and Ian H. Witten et al. [27], cluster analysis or clustering is the process which groups data entries (or objects) using only the available information found in the data that represents the entries. What we want by using this kind of analysis is to have our data distributed in groups (or clusters) where the elements from within a group are more similar among them than to the ones from other groups. If we maximize the similarity between the elements of our cluster and minimize the similarity with the ones from other clusters, we will have well defined clusters.

The concept of clustering can be found in fields apart of Data Mining, like biology, psychology or even in other computational issues, like in Information Retrieval where the categories may be seen as clusters. Moreover, clustering can be useful for simplifying task of other algorithms providing summaries or compressing data.

### 5.1.1 Preliminaries

After seeing the general idea, clustering can go deeper and present different properties or capabilities, depending on the problem:

**Clusterings can be partitional or hierarchical.** The first kind of clustering would be giving a partition of the dataset in clusters. If we admit having partitional clusterings of existing clusters, then we have a hierarchical clustering, in other words, if we have nested clusterings it is said that the process is hierarchical.

**Clusterings can be exclusive, overlapping or fuzzy.** An exclusive clustering would be a partition where the points on each cluster belong only to that cluster and not to others. Overlapping clustering would be the opposite, each point can belong to more than one cluster.

Fuzzy clustering works in the same way as overlapping clustering but with the difference that the membership of a point in a cluster is weighted with a probability. The sum of weights for each point must be 1. For example, a point can have a weight 0 to a cluster so we would say that it does not belong to that cluster and a weight 1 to another cluster, telling us that it surely belongs to that cluster; however, it can have any value in between. If we use only maximum belonging (value 1 in the example) and minimum belonging (value 0 in the example) we would have an exclusive clustering.

**Clusterings can be complete or partial.** Complete clusterings are the ones that assign every point to a cluster. However, we may want to leave outside some points because they are outliers. If we do so, we have a partial clustering. This differences between clusterings are not the only traits that define the whole process. Clusters can also be different from a process to another.

**Clusters can be well-separated or not.** Well-separated clusters are the ones where each point in a cluster is closer to other points in the same cluster than to any one in another cluster. If a point of a cluster is closer to a point of another cluster than to a point of the cluster where it belongs, the clusters are not well-separated.

**Clusters can be prototype-based.** In this kind of cluster we have a point prototype that acts as representative of the entire cluster. The most common prototype is a centroid, which is the mean of all the points that belong to the cluster. Moreover, if the data has categorical attributes, a medoid is used. This medoid is calculated as the most representative point of the cluster. The points belonging to a cluster are closer to the cluster's prototype than to other prototypes.

**Clusters can be graph-based.** In this kind of clustering data is represented as a graph. One way to do the partition could be searching for the strongly connected components of the graph and divide this connected components into groups. There are approaches that search for cliques, in other words, sets of points that are completely connected to each other.

**Clusters can be density-based.** In this clustering model, for example, two clusters are separated by a region where point density is low. The clusters would be the regions where density is high.

Once seen these clustering details, we have a set of different techniques to create partitions of users. For this project we have chosen *K-Means* as it is known to give good results in an affordable time, as well as the number of groups, $K$, can be defined. We will define this algorithm in next section.

## 5.2 $K$-Means

For clustering process we have selected $K$-Means algorithm as is known to be fast and will provide the functionality we require, partitionate users in groups. This algorithm is a prototype-based partitional algorithm which searches for partition in $K$ groups, being $K$ defined by the user. The objective of this algorithm is to minimize the sum of distances between the elements and their assigned centroids. This will provide partitions of elements that are close. This algorithm is used to do clustering in n-dimensional space with continuous attributes. If we need to have categorical attributes another option is to use $K$-Medoid as commented in the previous section.

The basic $K$-Means algorithm, also refereed as Lloyd's algorithm [28], proceed as follows:

**Algorithm 3** glsk-Means (Lloyd's algorithm)

```
1: Initialize K centroids given the criteria by the selected initialization function.
2: while Centroids change do
3:     Assign the closest centroid for each point p of users
4:     For each cluster recompute it's centroid
5: end while
```

To measure distance, typically, euclidean or Manhattan distance are used. For this project we have chosen to use euclidean distance and, in order to measure the quality of the solutions we have chosen to use the sum of the squared distances or sum of squared errors (SSE).

**Initialization function.** As seen in Algorithm 3, an important step of $K$-Means is initialization. It is a heuristic algorithm so it might converge to the global optimum or not depending on the initial solution given.

We propose two different approaches for initialization: Forgy initialization and $K$-Means ++.

**Forgy intialization.** This function, introduced by Charles Forgy [29], is a way to initializate by setting $K$ random points of the dataset as centroids. It is defined as follows:

**Algorithm 4** Forgy initialization function

1: Set every point from data into possible centroid list
2: **for** $i = 1 \rightarrow k$ **do**
3: Set a random point from possible centroid list as centroid $i$
4: Remove selected point from possible centroid list
5: **end for**
6: **return** K centroids

We have chosen to implement this function without repetition of centroids in order to have less conflicts when calculating centroids.

**$K$-Means++ initialization.** $K$-Means ++, proposed by David Arthur and Sergei Vassilvitskii [30], is an initialization function which strives to get an initial solution that makes the process faster and with better solutions. The function is defined as follows:

**Algorithm 5** $K$-Means++ initialization function

1: Set a random point from data as first centroid
2: Calculate a vector to store distances for each point to their nearest centroid
3: **for** $i = 2 \rightarrow k$ **do**
4: Set a random point from the point list as centroid $i$ using the distance vector as the probability distribution
5: Recalculate distance vector values
6: **end for**
7: **return** K centroids

This function provides an heuristic initial solution that converges faster than other initialization methods like Algorithm 4 and the probability of obtaining a good solution fast is higher than with the other function. Here we show, in the appendix Section A a comparision of both methods with our data.

**Obtaining a good solution.** As mentioned, $K$-Means is suboptimal, in other words, it may lead to local optimum, which may be different from the global optimum. Having a good initialization function is not everything we need as the random component of the process may affect us. Apart of the initialization, we need a method that provides us with a good solution, departing from the initialization. Our proposal for this problem is to repeat $K$-Means algorithm with different initializations $N$ times, where $N$ is an carefully chosen number, so we can get the best solution among the $N$ solutions generated by the repetitions. We chose the $N$ value from a study shown in appendix Section A).

## 5.3 Clustering in our problem

Now we will explain how to apply the concepts described in this chapter to our problem. As explained in Section 5.2, $K$-Means is a partitional, exclusive clustering algorithm. What we want to do in this project is to assign users to an exclusive group. However, an user may have more than one behaviour as commented in Section 4.3 thus leading to a problem of having multiple behaviours in one user. For this reason we will do clustering on the representation of each user and not on their sessions. In chapter 6 we will introduce another approach that will use sessions as input.

### 5.3.1 The n-ary tree clustering problem

Our representation of users traffic up to this point is an n-ary tree. In order to do clustering with this structure, we need to transform it into a vector that represents a point in a multidimensional space or, at least, have a distance function and a way to calculate the mean of various trees. For this reason we introduce the Edge Table structure.

**Edge tables** We require comparing trees and a parameter to adjust the error margin of similarity in order to not have over too general nor overfitted models. Comparing directly n-ary trees can be an exhausting task for a CPU as it may require a complex algorithm with backtracking to check which branches fit more. For this reason we created the *edge table* structure approach.

The basis of this structure is to have parts of the tree saved and compare directly those parts. In the table we will save in each entry an edge or a set of edges depending on a memory parameter. The *memory parameter* determines how many past edges does the table remember. With memory 1

the EdgeTable will only remember one edge, the one that leads to the node that represents the entry from its father. However if the memory is 2, it will also remember the node that leads to the father, hence the father's father. This parameter allows us to vary which grade of similarity we want to apply on our analysis. We will illustrate this with the table building algorithm pseudocode (Algorithm 6) and some examples.

Imagine that we have a set of nodes $n_1$,$n_2$,...$n_N$ of a tree connected with edges. From this tree we build an structure that may be seen as a matrix with a number of dimensions equal to the memory parameter. Such as:

$$M[n_1, n_2, n_3] = num\_ocurrences(n_1 \rightarrow n_2 \rightarrow n_3)$$

$$M[n_1, n_2, .., n_{k+1}] = num\_ocurrences(n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_{k+1})$$

where memory is equal to k. As also need to cover sets of edges with length less than k, we need to have a value in each dimension which doesn't refer to any node. The first example would be as follows:

$$M[n_1, n_2, n_3, noNode, ..., noNode] = num\_ocurrences(n_1 \rightarrow n_2 \rightarrow n_3)$$

Therefore, matrix has to be of dimension $(N+1)^k$. As there will not be edges like $n_1 \rightarrow n_2 \rightarrow noNode \rightarrow n_3$, we will have positions of the matrix that will never be used. Moreover, there will be combinations of pages that will not be possible edges as there is no link between them.

Because of this problem we propose to use a list instead of a matrix. The list identificators work in the way matrix parameters do so we can have directly the following:

$$L[n_1 \rightarrow n_2 \rightarrow n_3] = num\_ocurrences(n_1 \rightarrow n_2 \rightarrow n_3)$$

$$L[n_1 \rightarrow n_2 \rightarrow .. \rightarrow n_{k+1}] = num\_ocurrences(n_1 \rightarrow n_2 \rightarrow ... \rightarrow n_{k+1})$$

As this structure is designed to identify an edge or a set of edges with an identificator, the comparison of similarity between two of these structures is lineal in terms of the different edges that both structures have.

The *memory parameter* of edge tables and the insertion order that will be key in Section 5.4.

The algorithm works in a recursive and depth-first fashion. In every call we insert an edge with the exception of the first case (root node) as we do not have any edge to insert.

**Algorithm 6** Edge table building algorithm.

**Input:** Tree node and path done until the moment

1: If the input node is not the first node (root node), append the current node to the path done. If the length was equal to memory value, we delete first element and append the new.
2: Insert current path into the table. If it already exists, we add 1 to the count value.
3: For each child we call this function recursively with each child as the input node and the current path as path done.

**Normalization.** At the end of this process, the resulting table is normalized by the number of edges (the sum of the count variables). The reason to do this process is to measure as similar two users that visit the same page but one visits it 5 times and the other one 50. Without normalization distance between them would be 45, but normalizing distance it would be 0 and they would be considered as equals.

**Building example.** To introduce this concept we will explain it with an n-ary tree using the Algorithm 6, the same that we obtained in Section 4.3.1 where we introduced the NReverseTree structure but with one more node in order to see differences using different memory parameters. Note that memory parameter in this algorithm equal to 0 works the same as 1. The reason to do so is that we need this information in order to do the graph reconstruction in Section 5.4

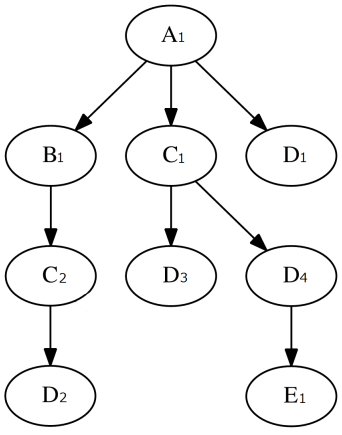



Figure 5.1: An example of the n-ary tree.

The first node, the root node of the example of Figure 5.1, would be node

A. As at this point we don't have any, we just call the function for it's childs, starting by node B.

| Child node list | $B_1$, $C_1$, $D_1$ |
|---|---|

Table 5.1: Child nodes of $A_1$ to visit.

Now that we are in node $B_1$, we have as path $A_1 \rightarrow B_1$, therefore we can insert it in the table.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |

Table 5.2: Edge table on step 1.

Next node is $C_2$. Here we see the first difference between two executions as the depth of the path is now 3. In this case the example with memory 1 we will have a path equal to B and in the example with memory 2 we will have A → B. This leads to two different entries on Table 5.3.1.

| Child node list | $C_2$ |
|---|---|

Table 5.3: Child nodes of $B_1$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |

Table 5.4: Edge table on step 2.

Finally, we are at the end of the branch reaching $D_2$. Here we will just add the node to the table and then backtrack to $A_1$

| Child node list | $D_2$ |
|---|---|

Table 5.5: Child nodes of $C_2$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |

Table 5.6: Edge table on step 3.

Up to this point, the following node to explore is $C_1$.

| Child node list | $C_1$, $D_1$ |
|---|---|

Table 5.7: Child nodes of $A_1$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 1 |
| A → C | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |
| A → C | 1 |

Table 5.8: Edge table on step 4.

C has two D children. Here we will see the main difference between having enough memory or not. We already have added an edge C → D which in memory 2 example comes from B, however we can not differ this case from the one given by $A_1 \rightarrow B_1 \rightarrow C_2$ because of memory shortage.

| Child node list | $D_3$, $D_4$ |
|---|---|

Table 5.9: Child nodes of $C_1$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 2 |
| A → C | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |
| A → B | 1 |
| A → C → D | 1 |

Table 5.10: Edge table on step 5.

Now we will insert $D_4$ node.

| Child node list | $D_4$ |
|---|---|

Table 5.11: Child nodes of $C_1$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 3 |
| A → C | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |
| A → B | 1 |
| A → C → D | 2 |

Table 5.12: Edge table on step 6.

And then $E_1$ which has a totally new edge as E has not appeared before in the system.

| Child node list | $E_1$ |
|---|---|

Table 5.13: Child nodes of $D_4$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 3 |
| A → C | 1 |
| D → E | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |
| A → B | 1 |
| A → C → D | 2 |
| C → D → E | 1 |

Table 5.14: Edge table on step 5.

Finally we insert the last element.

| Child node list | $C_1$, $D_1$ |
|---|---|

Table 5.15: Child nodes of $A_1$ to visit.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| B → C | 1 |
| C → D | 3 |
| A → C | 1 |
| D → E | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → B → C | 1 |
| B → C → D | 1 |
| A → B | 1 |
| A → C → D | 2 |
| C → D → E | 1 |

Table 5.16: Edge table on step 6.

At this point we have the complete edge tables that describes the tree with memory 1 and 2. The final step would be to normalization.

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1/7 |
| B → C | 1/7 |
| C → D | 3/7 |
| A → C | 1/7 |
| D → E | 1/7 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1/7 |
| A → B → C | 1/7 |
| B → C → D | 1/7 |
| A → B | 1/7 |
| A → C → D | 2/7 |
| C → D → E | 1/7 |

Table 5.17: Edge table on step 7.

**Distance between two edge tables and mean.** Now we will describe how to measure distance between tables. As explained in Section 5.2, as distance function we will use the classic euclidean distance. For this task we will take each entry of the table as a dimension. Due to compression of the structure not every dimension of the space is directly represented. For example, if we have a table with A → B and we want to compare with another table which has C → D, both tables could be seen as in the table 5.18.

| Table 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| C → D | 0 |

| Table 2 | |
|---|---|
| Path | Count |
| A → B | 0 |
| C → D | 1 |

Table 5.18: Representation of omitted values on edge tables.

The comparison function of two tables is the euclidean distance calculated as follows:

$$d(t1,t2) = \sqrt{(t1[1]-t2[1])^2 + (t1[2]-t2[2])^2 + ... + (t1[M]-t2[M])^2}$$

$$= \sqrt{\sum_{i=1}^{M}(t1[i]-t2[i])^2}$$

Where M is the number of different edges, or dimensions, that both tables have and t1[1] reefers to the first dimension of t1, including omitted ones that appear in the other table.

The mean is calculated in a similar way. For each dimension, we sum every value and divide by the number of tables we are using to calculate the mean, as we would do normally with, for example, a 2 dimensional point.

**Effects of memory parameter on clustering.** Seen this example, it is easier to understand how memory affects to the whole process. Memory affects the speed of the execution as we may add new dimensions and, what is more important, may have over-fitting in our model. If we use a memory number equal to the deepest path on users' trees, we might be overfitting the model; Two users that are almost similar could be taken as different this way. This parameter is important also to find loops in our traffic, but will introduce this in Section 5.4, where we explain how the graph is reconstructed from an edge table. In appendix A we will see how memory affects in the result variables.

**Performance.** In terms of performance, edge tables are not the best representation to use to do $K$-Means calculations. For this reason we have created alternative structure that uses a map to set an unique ID to every set of edges. We build a vector for each users with size the size of the map and setting the number of occurrences of each edge set. For example, if we did this process in Table 5.18 it would look like Table 5.19.

| Table 1 | |
|---|---|
| ID | Count |
| 0 | 1 |
| 1 | 0 |

| Table 2 | |
|---|---|
| ID | Count |
| 0 | 0 |
| 1 | 1 |

Table 5.19: Representation of omitted values on edge tables.

In this case A → B would be represented as the identifier 0 and C → D with 1. We can represent this structure directly as a vector. The examples

can be represented as (1,0) and (0,1), as the ID can be the position on the vector. In this structure we don't have compression at all as every table has the same size, the number of different dimensions.

As we said, we lose compression with this table as we enlarge the table with 0s; however, as the operations access the values of the table in order, exploiting the spatial locality of memory in computers. Moreover, GNU C++ compiler provides automatic optimizations for kind of vector usages and this structure could easily be modified to take advantage of vectorial instructions in order to make calculations even faster.

## 5.4 Graph reconstruction

After doing the clustering we will obtain $K$ edge tables that represents each group. As these tables are extremely hard to analyze and we want them to be viewed easily we will do the inverse process of table construction, graph reconstruction. This process will build a graph that represents the traffic represented in an edge table. The reason to do so is to provide an easy way to read the edge tables, that represent each group found, obtained from clustering process.

For each element in the list, we will obtain the *memory path*, which is the set of nodes remembered in a successions of edges. For example, in a succession $n_1 \rightarrow n_2 \rightarrow ...\rightarrow n_{k-1} \rightarrow n_k$, memory path would be $n_1 \rightarrow n_2 \rightarrow ...\rightarrow n_{k-1} \rightarrow n_k$. This memory path represents the succession of fathers the node we are inserting have. After obtaining this *memory path* we search for fathers that accomplish that accomplish that succession of edges and add them as fathers of the node we are inserting. We have established a special case with memory equal to 0 to have an output graph were there are not two nodes representing the same content. This is interesting if we want to have a more simplified view of the output. The algorithm can be seen in Algorithm 7.

A thing that may catch our attention about this process is that it works different when memory is equal to 0. This is given by the fact that there will be only one father as there will not be equivalent nodes (with the same content, for example $C_1$ and $C_2$). Therefore, we do not have to search for potential fathers as there will be only one unique father.

Memory 0 tables are built as memory 1 tables because we need to know their father in order to reconstruct the graph, however, the nodes with memory 0 can be seen as if they do not know their father; Only their father know its children. When we add edges to the potential fathers of a node, we are generalizating our model in a way that may be under-fitting. In order to

**Algorithm 7** Graph reconstruction

**Input:** An EdgeTable

```
1: Insert root node
2: Order elements by the insertion order.
3: if Memory > 0 then
4:     for each element in list do
5:         Extract the memory path
6:         Search fathers that match this memory path
7:         for each node that matches (possible father) do
8:             Add edge between the node and the possible father.
9:         end for
10:    end for
11: else
12:    for each element in list do
13:        Add edge between the node from the list and the father
14:    end for
15: end if
16: return Graph representation of the list
```

fix this we must do a simulation of the model with the users that belong to the cluster represented by the current centroid. This simulation is called paternity test and is described in Algorith 8.

**Algorithm 8** Paternity test (Graph Simulation)

**Input:** A graph of centroid with possible fathers and n-ary tree sessions.

```
1: for each user that belongs to the cluster represented by the centroid do
2:     for each session of the user n-ary tree do
3:         Simulate each step of the session in the graph adding weight 1 to the
   used edge in the graph.
4:         Add 1 to the end count of the node on the last element that we have
   travelled to.
5:     end for
6: end for
7: Remove edges that have weight 0.
8: return Graph with weights and without extra edges
```

The main reason to do the transformation from edge tables and then this simulation instead of fusing n-ary trees of the users is that this way we can observe loops in our result depending on how we set memory parameter. Depending on the tree fusion algorithm we would not have this loops that may be interesting to analyze.

To understand better this process, we will give an example of how algorithms 7 and 8 work. Initially we have an user with the following traffic:

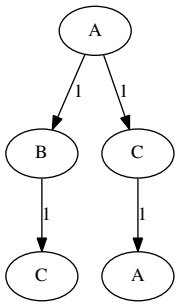



Figure 5.2: User's traffic for the example.

This traffic would produce tables for memory 1 and 2 like this:

| Memory 1 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → C | 1 |
| B → C | 1 |
| C → A | 1 |

| Memory 2 | |
|---|---|
| Path | Count |
| A → B | 1 |
| A → C | 1 |
| A → B → C | 1 |
| A → C → A | 1 |

Table 5.20: Edge tables produced by the example of the Figure 5.2

Now we proceed to reconstruct the graph from the given tables, starting with memory 0, that is the same as the memory 1 table but will be used differently as seen in algorithm 7. Now we will add the root node and then the first edge.


Figure 5.3: Root node insertion.

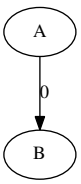



Figure 5.4: First edge insertion.

Then we will add the second edge and the node C:

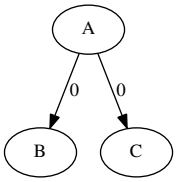



Figure 5.5: Second edge insertion.

With memory 0 nodes with same tag (A, B or C in this example) are the same node, independently from where they come. Given this, we don't need to insert any nodes at this point, just edges. So we will insert the edge B $\rightarrow$ C.

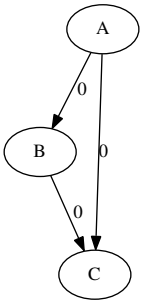



Figure 5.6: Third edge insertion.

Now we will insert edge C $\rightarrow$ A. This edge creates a loop in our graph, therefore we can not say this is a tree anymore. If we want to obtain a tree, we need a memory parameter bigger or equal to tree's deepest path.

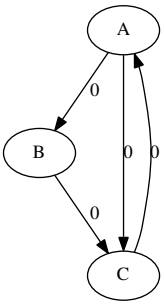



Figure 5.7: Last edge insertion.

Now we almost have the final graph, we just need to do the process shown in algorithm 8. After the graph simulation we obtain the following graph:

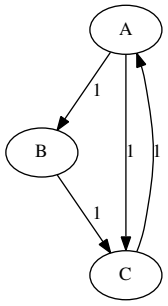



Figure 5.8: Final graph with weights. Memory 0.

Now, for memory 1 we will start from step of Figure 5.5. Up to this point the process was equal to the one made with memory 0. In this point the process starts to differ because we use the knowledge we have about father nodes. The insertion of B $\rightarrow$ C would look like this:

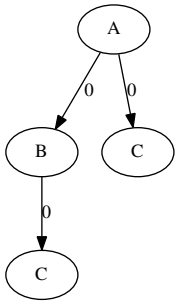



Figure 5.9: Third edge insertion with memory 1.

The following edge, C $\rightarrow$ A is where algorithm 8 is more useful. In this case we have to insert A with a father with tag C. Given this condition we have two possible fathers, so we add an edge between the new node and them.

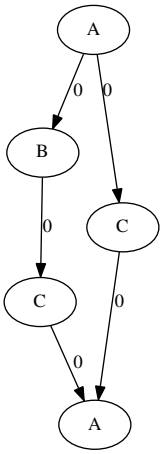



Figure 5.10: Fourth edge(s) insetion with memory 1. More than one possible father case.

And then we use algorithm 8.

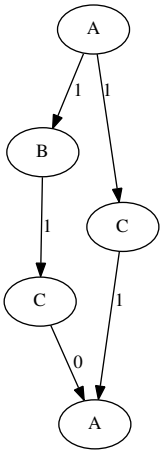



Figure 5.11: Graph simulation of the Figure 5.10

Here we see that there is an edge with 0 transitions on it, so we just delete it as that C node was not the real father. Finally we obtain the following graph:

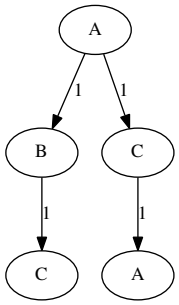



Figure 5.12: Final graph with weights. Memory 1.

In this case we would obtain the same graph with memory 2 (deepest path) because of the algorithm 8. In the case of memory 2 process we would not have any problem inserting edge C $\rightarrow$ A as it comes with an extra memory entry that makes it be A $\rightarrow$ C $\rightarrow$ A denotating that the other C node is not even a possible father.

## 5.5 Conclusions

In this chapter we have seen what clustering is and its properties and how is our problem in terms of clustering. We have chosen an algorithm that seems to adjust well to our constrains and requirements and proposed the implementation in terms of the algorithm calculation and the final output for the user. This process will provide us a better view of our traffic providing groups of users, giving us the opportunity to analyze data easier and deduce what kind of behaviour lies on each group.

As for the implementation of the algorithm and data structures, we have found that the initial representation, the n-ary tree, needs several transformations in order to make the clustering feasible and fast.

Results of the application of this knowledge will be seen in Section 7. In Section A we will see how some parameters are related and which values we will select for our study.

# Chapter 6

# Learning how the user behaviour changes

In this section we explain another point of view for analyzing behaviours. Users do not have to belong only to one behaviour at all, they can move from one to another given their circumstances. We aim to cover this issue using FSA inference algorithms. This approach allows us to find behaviours and transitions between them.

## 6.1 Introduction to FSA inference algorithms

On the FSA inference algorithms set we find various techniques to solve the problem of inferring FSA from a given sample set. The basis of this set of techniques is having a starting graph made with the sample set where given a criteria, nodes will be merged. Some algorithm may require a positive sample set (a sample set that represents what belongs to what we want to infer) and a negative sample set (what does not belong to it).

The result of this process is a model that represents the sample set given in a compact way. A typical example of usage of this kind of algorithm is inferring grammars of regular languages.

For this project we have selected the Red-Blue algorithm which is, in essence, the ALERGIA algorithm proposed by Rafael C. Carrasco and José Oncina [31], using the Blue-Fringe framework proposed by Kevin J. Lang et al. [32]. In particular, we will describe the one that can be seen in the survey done by Jorge Castro and Ricard Gavaldà [33] and in the Master Thesis of Toni Cebrián [34].

Before presenting the Red-Blue algorithm, we give some preliminaries that need to be known in order to understand some parts of the algorithm.

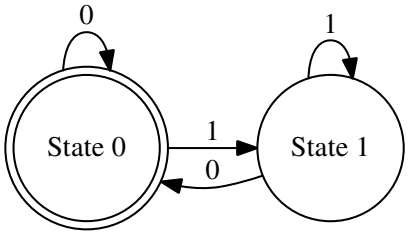



Figure 6.1: An example of a DFA

### 6.1.1 Preliminaries

Previous to the algorithm presentation we must know other things that are applied in the algorithms. We present the concept of FSA and derivated models that are used in our project.

**Finite State Automaton** A FSA is a mathematical model used to design computational programs and circuits. It is a model represented with a graph where we have a finite number of nodes that are the states of the automaton and edges are associated with a symbol of the alphabet of a given language, denoting the transition between states.

Of all the types of FSA we will cover Deterministic Finite Automaton (DFA), Non-deterministic Finite Automaton (NFA) and Probabilistic Finite Automaton (PFA) as these are the ones we need for the scope of this project.

**Deterministic Finite Automaton** A DFA or Deterministic Finite State Machine (DFSM) is an FSA that can be defined by a tuple of 5 elements:

- A finite set of states (Q).
- A finite set of input symbols called the alphabet ($\Sigma$).
- A transition function ($\delta$ : Q x $\Sigma \rightarrow$ Q).
- A starting state ($q_0 \in$ Q).
- A set of accept or final or acceptor states (F $\subseteq$ Q).

Given a string $w$ formed by elements of $\Sigma$, the task for the automaton will be to process this string and tell us if it is accepted in the model.

A simple example can be the automaton defined in Figure 6.1. This automaton has only two states and only one of them, the state 0, is a final state. The alphabet is $\Sigma$=$\{0,1\}$ and the starting state is state 0. The $\delta$ would be the transitions denotated in the figure.

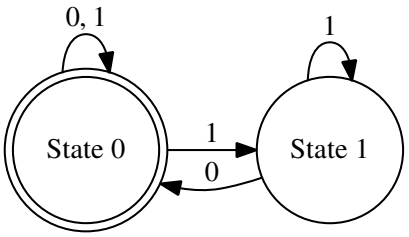



Figure 6.2: An example of a NFA

Being the final states denoted with a double circle, this automaton recognizes strings that end in 0. For example, if $w$ is the string 010, the transition between the states would be this: State 0 (as starting state) $\rightarrow$ state 0 $\rightarrow$ state 1 $\rightarrow$ state 0. As the transitions end in state 0, which is final, the word is recognized as defined by the automaton.

**Non-deterministic Finite Automaton** The DFA model can be extended changing the function $\delta$ for a transition relation $\Delta \subseteq Q \times \Sigma \times Q$. This transition relation represents that a state can have a relation to more than one state using the same symbol. We will illustrate this with a example using the figure 6.2.

This example is the same automaton as the previously seen DFA with the addition of a 1 in the edge that connects the state 0 with itself. This means that when the automaton receives a 1 in state 0 it may go to state 0 or state 1. This example, in essence, would recognize every possible combination of the alphabet, commonly denoted as $\Sigma^*$, because of the non-deterministic behaviour.

**Stochastic Finite Automaton** In NFA we cannot predict which behaviour would the automaton take when the transition function for a symbol gives more than one state. However, this model can be extended setting probabilities and, thus, producing a Probabilistic Finite State Machine (PFSM) or Stochastic Finite Automaton (SFA).

The definition of a SFA differs form a NFA in the terms of transition an final or acceptor states. For each element of the set of transitions defined in $\Delta$ we set a probability. Moreover, we set a probability in the set F, final states, for each entry. This two probabilities must follow this property:

$$p(f(q)) + \sum_{q'}^{q' \in Q} \sum_{s}^{s \in \Sigma} p(\delta(q, q', s)) = 1$$

Being $q$ and $q'$ two states of $Q$, $\delta$ the transition function between $q$ and $q'$ given a symbol of $\Sigma$, $p$ the probability and $f$ the final function. The first term of the addition refers to the probability of ending in a given state $q$ and the second refers to the probability of transition between a state $q$ and a $q'$. This means that the probability must be normalized to 1 for each node. In the example shown in Figure 6.3 we can see that there are 3 different node transitions for state 3, pondered with a probability of 0.25 each and an ending probability of 0.25, being 1 their sum as the property says. This is also accomplished in state 0.

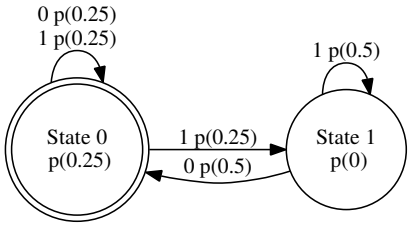



Figure 6.3: An example of a PFA

There is a special case of automaton that we are going to use as starting point in the algorithm, the Prefix Tree Acceptor (PTA).

**Prefix Tree Acceptor** A PTA is a DFA with the shape and properties of a tree having on every state a prefix. It is build taking all the prefixes of the input data set. In the example of the Figure 6.4, we use as input of the building process the set of three words: Home, house and hair.

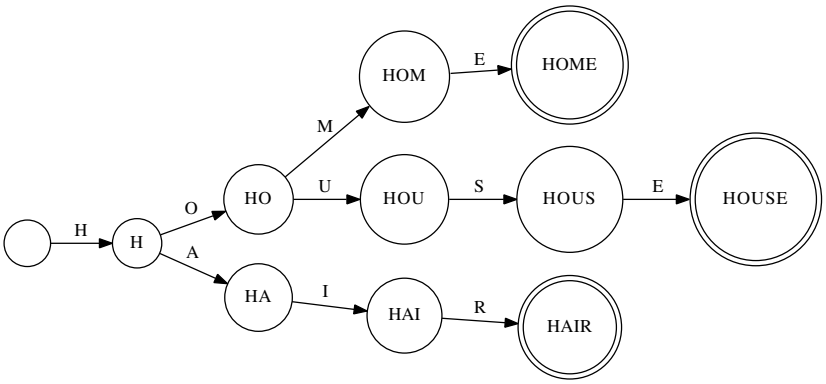



Figure 6.4: An example of a PTA

As we see, this automaton only accepts these three words. The acceptor

name is given by the property of only accepting the language defined by the input.

## 6.2 Red-Blue algorithm

Inside the category of FSA inference algorithms there are two types:

- Acceptor inference: This kind of algorithm uses a positive and a negative sets in order to infer a FSA which can accept every sequence from the positive set and reject every sequence of the negative set.
- Generator inference: This kind of algorithm uses only a positive set to infer a FSA which can generate all the sequences from positive set with a given probability distribution.

From these two the second is the approach we need for our project as we want a model that represents the traffic generated on our website. We propose the usage of Red-Blue algorithm to infer a stochastic model that we will use after the process to see how every user behaves and estimate how the population behaves.

Our starting point for this algorithm is building a PTA with the data sets we have available. The Red-Blue algorithm defines the order each node is visited and how it is treated, by setting colors to each node. There can be three different types of nodes:

- Red nodes: Nodes that have been visited.
- Blue nodes: Nodes that are children of red nodes and have not been analyzed yet.
- White or colorless nodes: Nodes that are neither of the previous types.

The blue nodes will be colored as red after they are analyzed. If a blue node is compatible with a red node, both nodes are merged. Else, the blue node is promoted to red node. After all blue nodes are merged or promoted, the children of the red nodes are promoted to blue node. If a red node is the child of another red node, this node is not promoted to blue node.

As compatibility of two different nodes can only have two possible results, compatible or not, it can be seen as a Bernoulli variable. The Hoeffding bound, proposed by Wassily Hoeffding [35], provides a confidence range for Bernoulli variables given a confidence parameter $\alpha$.

The function defined in algorithm 9 will return true if the difference of the frequencies are in the confidence range, meaning that it is believable

**Algorithm 9** Hoeffding Bound Test (HBT)

**Input:** $f_1, n_1, f_2, n_2, \alpha > 0$

1: **return** $|\frac{f_1}{n_1} - \frac{f_2}{n_2}| < (\sqrt{\frac{1}{n_1}} + \sqrt{\frac{1}{n_2}})\sqrt{\frac{1}{2} ln \frac{2}{\alpha}})$

that $f_1$ and $f_2$ come from Bernoulli variables having the same probability distribution with probability at least $1 - \alpha$. The input variables used for the calculation are defined in algorithm 10.

The function described in algorithm 9 is used several times in the compatibility function, the one that measures if two states are compatible for merge or not. This function, the one described in 10, use for the Hoeffding bound test function the following functions:

- $F_{end}$(SFA A, Node $q_u$): returns the frecuency of ending in the node $q_u$ of the SFA A.

- $F_{total}$(SFA A, Node $q_u$): returns the frecuency of passing by or ending in the node $q_u$ of the SFA A. This is the total frecuency of a node.

- $F_{out}$(SFA A, Node $q_u$, Symbol a): returns the frecuency using the outgoing node associated with the symbol a in the node $q_u$ of the SFA A.

- $\delta$(Node $q_u$, Symbol a): returns the node that is accessed by following the arc represented by a (a child node of $q_u$).

The defined compatibility function is used in the algorithm 11 to check the compatibility of two different nodes. Only if they are compatible, or is believable that these two states are actually the same state in the hidden FSA generating the sample, they are merged.

In this algorithm we have a new parameter, $t_0$. This parameter allows us to ensure statistical significance for the study. If a node $q_u$ of a SFA A has a $F_{total}$(A, $q_u$) $\geq t_0$ it will be candidate of merging. Otherwise, it will be ignored.

The merging process of the algorithm 11 works in a recursive fashion, like the compatibility function, to get rid of non-deterministic choices in our automaton.

**Algorithm 10** State compatibility function (Compatible)

**Input:**an SFA A,states $q_u$ and $q_v$ and $\alpha > 0$

Compatible = true
**if** HBT($F_{end}$(A, $q_u$), $F_{total}$(A, $q_u$), $F_{end}$(A, $q_v$), $F_{total}$(A, $q_v$), $\alpha$) returns false **then**
    Compatible = false
**else**
    **for** a $\in \Sigma$ **do**
        **if** HBT($F_{out}$(A, $q_u$, a), $F_{total}$(A, $q_u$), $F_{out}$(A, $q_v$, a), $F_{total}$(A, $q_v$), $\alpha$) returns false **then**
            Compatible = false
        **end if**
        **if** compatible(A, $\delta(q_u, a)$, $\delta(q_v, a)$, $\alpha$ ) returns false **then**
            Compatible = false
        **end if**
    **end for**
**end if**
**return** Compatible

**Algorithm 11** Red-Blue Algorithm

**Input:**a data set S, $\alpha > 0$ and $t_0$

We build a PTA A with S
Set $q_\lambda$, the root node, into the list of red nodes.
Set the children of $q_\lambda$ into the list of blue nodes.
**for** each qb from blue node list such that $F_{total}(A, qb) \geq t_0$ **do**
    **if** $\exists qr \in$ red node list such that Compatible(A, qb, qr, $\alpha$) **then**
        Merge(A, qb, qr).
    **else**
        Add qb to red node list.
    **end if**
    Remake the blue node list setting the child nodes of the nodes from the red node list as blue only if they are not in the red list.
**end for**
**return** A

## 6.3 FSA inference in our problem

For the project we will use the Red-Blue algorithm for the inference of transitions between users behaviours. The input of this algorithm is a confidence parameter $\alpha$, the $t_0$ and the traffic sessions of the users.

These traffic sessions are obtained using the NReverseTree class mentioned in Section 4.3.1 by following the path from the leaf nodes to the root. Each user have a number of sessions equal to the number of leaf nodes. Using this we will obtain a set of lineal sessions for each user.

Each session is a string of one or more symbols of an alphabet $\Sigma$. $\Sigma$ will be the list of all different pages represented by an identifier from 0 to the number of the pages minus 1. Therefore, each node will represent a different page at the start of the algorithm when we construct a PTA from the set of sessions.

Pages will merge when they are compatible, creating nodes that represent more than one page, which are similar in how they are used. This will generate partitions in the initial PTA. We consider each partition as a behaviour and each edge from a partition to another is a transition between behaviours.

### 6.3.1 Treba

For this task we have chosen to use an already existing software, Treba [19], developed by Mans Hulden. This software provides a implementation of the Red-Blue algorithm as described along with other algorithms. Treba takes as input the parameters previously mentioned and, in addition, takes the selected algorithm as parameter as Treba includes a set of algorithms and different techniques to choose from. For Treba to perform right, we need to create a map between page identifiers an integers as Treba works with an $\Sigma = \{0,1,...,N\}$, where N is the number of pages minus 1.

Treba's call is implemented with the popen system call that creates a fork to execute the application and opens a pipe to retrieve the result. Treba will output the text representation of a graph and, for each transition between nodes, a probability and a symbol (in our case a page). We must use a map to transform the integers back to page identifiers.

Given that there may be a lot of different edges from a node to another, we decided to mix all of them in the graph summing their probability and then, in order to not lose content, make various lists with the symbols that lead to a node, the symbols that exits from a node and the symbols that makes a cycle in a node.

## 6.4 Conclusions

In this chapter we have seen FSA inference and the required background. We have seen that FSA inference algorithms are applicable to our problem and provide a model. By using this kind of model we can obtain knowledge

about how the user behaviour changes in basis of what he or she sees and the path that has been taken. This actionable knowledge can be used in many ways, like, for example, grouping more related pages or finding issues with website like users falling in error or users acceding to a content that was not desired because the description of the link was badly written.

Results of the application of this knowledge will be seen in Section 7.

# Chapter 7

# Experiments

In this section we will comment the results of the program and we will analyze them in detail. We will use three different kinds of dataset: A wiki, a game's news website and the same game's forum. Each dataset will have a section where the results will be explained.

The experiment's methodology will be iterative and it will have the following steps:

1. Do experiments with the different available parameters.

2. Do a summary of what have been seen.

3. Show the results to the stakeholder. If the stakeholder gives more information to refine the process (by confirming that some pages can be noise to filter or not interesting, for example), we return to step 1.

Each dataset will have a subsection for each iteration of the experiments. In each subsection both algorithms, clustering and FSA inference, will be considered. For clustering we will be using the memory parameter, that describes how many steps we will remember from each access, and the $K$ parameter, which describes the number of different groups we want to have. Both parameters were defined in Section 5.1. For FSA inference we will be using the alpha parameter, which describes the confidence level of state merging, and the $t_0$ parameter, which is the one that allows to not merge things that have been seen under $t_0$ times. Both parameters were defined in Section 6.

As a starting point, the first iteration of each dataset will have no filtering of any kind given that nothing is known about the website structure, or should not be known at the start. Moreover, the first iterations will show how an analyst would do the work step by step in different ways. In further

iterations, only conclusions of what has been seen are written along with what is needed to improve for the next iteration.

As there are pages that are visited only once in the website but may fall into a category of the web site (for example, articles of a wiki), we do not want to ignore these pages. If we ignore less frequent pages, we may be ignoring a category of our website. Therefore, our hypothesis is that the parameter $t_0$ will give clearer results being set at 1, as bigger values will not merge some of the less frequent pages.

## 7.1 Wiki dataset

The wiki dataset we are going to study represents a month of use of the website. It has 197 different users and 103 different pages. The traces depth is in a range from 1 to 5 with a mean of 1'17 of depth. Most representative depths are 1 with 644 occurrences and 2 with 114. This website is not frequently used by humans. Our first hypothesis is that we will find that most of the users, if not all, are bots.

### Iteration 1

**Clustering** At first glance, with the clustering results with a $K = 2$, we can't conclude anything because both clusters seem quite heterogeneous given any memory parameter, so a bigger $K$ will be used.

With $K = 5$, a more homogeneous behavior is seen. In every test done with different memory parameter at least one of the five clusters is only formed by users that access to images. This is due to hotlinks of external websites to that image. In this case, the hotlinking comes from Google images.

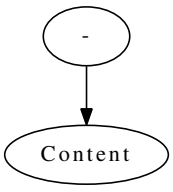



Figure 7.1: An example of the traffic generated by hotlinking.

Another observed tendency is the "one time, one visit" user type, which we will use further in the experiments as a behaviour that encloses others. This kind of user visits the website one time to see the desired content and then ends its surfing on the website. This tendency may be also due to

web crawlers, as they usually access a web, search for links, stack them and reiterate until there is no more website to visit. This produces a kind of traffic that only have one level in depth, as they always come from a no reference.

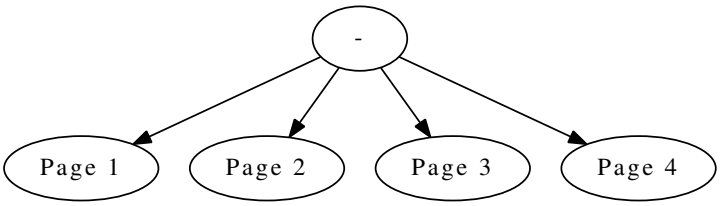



Figure 7.2: An example of the traffic generated by a bot.

Up to this moment, the rest of the traffic remains heterogeneous. Covered the hotlink user, the stakeholder has no more interest in images. Henceforth, images will be filtered in the following iterations.

**FSA inference** For this FSA inference experiments an $\alpha$ range between 0.01 and 0.51 in steps of 0.1 and a $t_0$ set to 1 and 2 have been chosen. The first interesting result that have been observed is that with $t_0 = 2$ the mean of the number of different states obtained is 157 nodes. As number of different pages in this dataset is 194, this parameter value will not be used in the futher iterations if it keeps producing poor mergings.

With a $\alpha$ parameter set at 0.01 a 3 state SFA have been learned. This SFA shows that we have two kind of behavior: "surfing users" and "ending users". The first behavior is seen in states 0 and 2. As state 0 is the starting point for everyone, a surfing user will return to that state or travel to state 2. The returning users tend to visit wiki pages and images, however the ones who travel to state 2 only visit wiki pages. It have been observed that the difference between the pages of one state and the other is the popularity. The wiki pages visited on state 2 are more popular (or demanded) than those visited on state 0.

Up to this point, only state 1 is left unknown. The most probable page that leads State 0 to this node is "/w2/index.php". This page is a special Mediawiki page usable for different chores like viewing the difference between two revisions of one page or adding a page to the user's watch-list. This page should be normally seen on long traffic traces because it is used as a support for maintaining the content, yet is visited a lot in withing the "one time, one visit" behavior. Nonetheless, these users are robots crawling the review system.

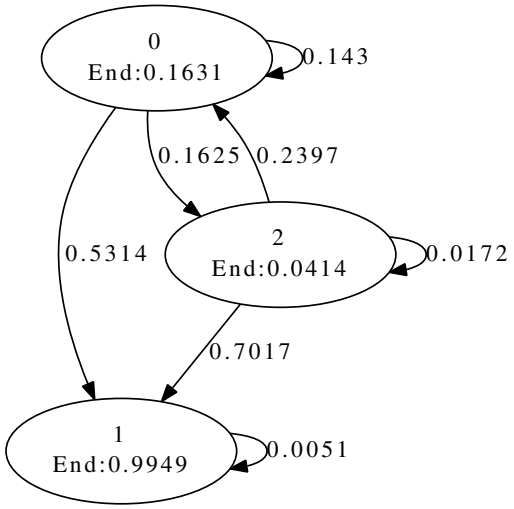



Figure 7.3: Wiki dataset's FSA inference graph with parameter $\alpha$=0.01 and $t_0$=1.

The main page coming from the state 2 is "/w2/load.php". This page is a Mediawiki special page used to load resources in another page. It is normal for this page to be the ending page if the page from where the user comes is the last visited.

With $\alpha$ set at 0.51 the previous graph can be observed in a less merged way. Now, the previous state 2, known as popular website state, is now divided in state 3 and 4, being the pages of state 4 more visited than the ones from state 3. The new state 2 seems to be special traffic. This special traffic is formed only by favicon and robotstxt. We observe that this traffic is made by users and bots but favicon is most likely negligible as it is demanded every time a website is open. Given that, only robotstxt is left. Seen that in order to enter the state 2 we need to request robotstxt, it is obvious that this behavior is a web crawler behavior. The probability to reenter the state is due to visits of pages and images, crawled by the web crawler. This crawling is also reflected in the "less popular websites" state, or state 1, as the input of FSA inference, due to implementation, has a problem with sessions as seen in section 6.

**Conclusions** Due to the function of "/w2/load.php", it really does not give too much information to differentiate traffic, so we will filter it. Also, images and favicon will be filtered because this files are requested almost every time, therefore they can be considered noise.

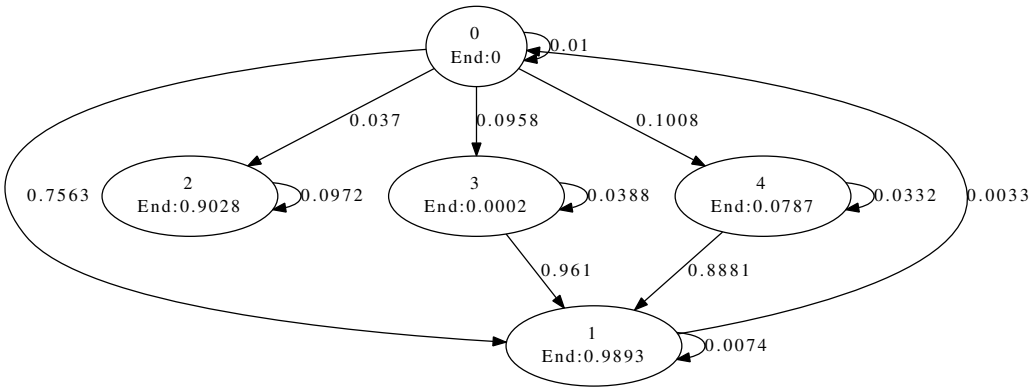



Figure 7.4: Wiki dataset's FSA inference graph with parameter $\alpha$=0.51 and $t_0$=1.

### Iteration 2

**Clustering** Having filtered images and "/w2/load.php", clusters are more clear. We observe a very small group of users whose behaviour can be denominated as "real reviewers". This kind of user view an article and then edit them or watch the revisions, not directly the revisions. Crawlers, on the other hand request special pages directly, coming from no reference.

We expected to find random article readers as mediawiki has a random article feature, however we have not found any.

**FSA inference** This time we have once tried $t_0$ set to 2 but the content of this wiki is so diverse that a big part of the articles are only visited 1 time. With $t_0$ set to 1 we have found that with a big confidence, $\alpha$ set to 0.01, we obtain a two state automata. Reducing this confidence to the maximum possible, $\alpha$ set to 1, we obtain the automata that is equal to the one shown in Figure 7.5.

With this model we see that our hypothesis about popular pages and non-popular pages was not the only factor affecting the traffic. The "/w2/load.php" page filtered as noise really had a function, only normal users request this page. By our experiments, seems that the crawlers we have in the website download directly the php file and then search links in the local copy. Therefore, popular pages and human users have some kind of correlation. We have studied one of the specific popular pages that makes reference to a person and the requests comes from a spanish forum on where there is a post refering to that person and a link to this web page.

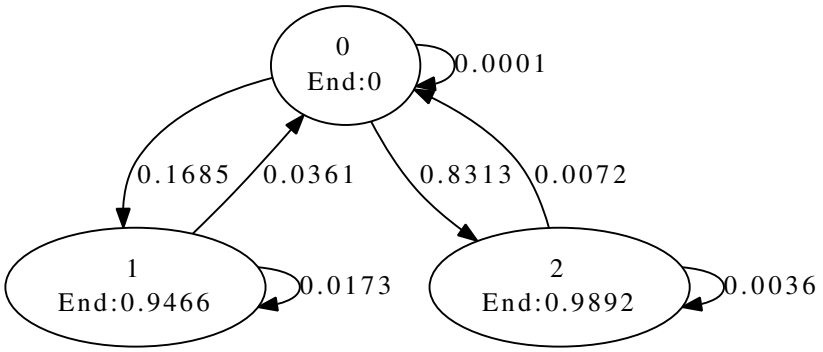



Figure 7.5: Wiki dataset's FSA inference graph iteration 2 with parameter $\alpha$=0.51 and $t_0$=1.

**Conclusions** In this website around a 80% of the traffic is done by bots. Yet, we can find different two kinds of human users: readers and reviewers. Moreover, there are two kind of readers: the ones who come directly for a specific article from a link in a website and the ones who access to the main page. The "/w2/load.php" page, which at first we thought it did not give any information surprised us as it helps in the task of discern bots from users. Finally, a correlation with popular pages and user pages was found. Most of the users that access to a specific page comes from a page where the link to the article is written, as we explained before with the spanish forum example.

## 7.2 Game's news website dataset

The game's news dataset we are going has been gathered during a day. It has 175 different users and 96 different pages. The traces depth is in a range from 1 to 5 with a mean of 1'32 of depth. Most representative depths are 1 with 228 occurrences followed by 2 with 158. Our hypothesis here is that most of the traffic will go around the main page, as the recent news can be seen there.

### Iteration 1

**Clustering** The study starts with a $K$=10 and memory equal to 5. It is likely that this model will be over-fitted, however it is a point of view that may offer ideas of cluster merging by managing the parameters. Unexpectedly, all clusters except one represent media traffic. This includes images, CSS and favicon. As the stakeholder is not interested in this nodes, they will be filtered for the next iteration.

Given that interesting information was found, a study with the same $K$ and a memory equal to 1 is done. At this point it is seen that there are more clusters that contains useful traffic, not just media files. One of the clusters presents an interesting resource usage that proceed as follows: from no reference makes a request to "/mw/index.php" which is, in fact, the main page where news can be seen. This is represented, once more, by the "one time, one visit" behavior, previously seen in Section 7.1. This behavior comes in the most pure "pass by" flavour. Up to this point we may want to define a behavior less general than the "one time, one visit" one as we have observed that is quite frequent to have this tendency. This new behavior will be, henceforth, called "news reader".

We have detected more clusters that also presents the this behavior. In this case the root page and the website's front page are involved as follows: First, the user enters the site from no reference or from another external website (like a search engine) requesting the root page. As this website is a portal for various games, the root is different from the game's front page so people interested in the concrete game's website must choose it in that root page, then the game's front page is accessed. Then the traffic ends. The front page of the game is a news board and news can be read directly on it. Given that, it is quite normal to have this kind of behavior.

Also another interesting behavior can be found in relation to some pages of "articulos" folder of the website but will be left for further study in the following iterations as noise made by images seems too big.

**FSA inference** Once more, with a parameter $t_0$ set to 2, the graph has too many nodes. A good model may be found after filtering, but in this iteration only models with $t_0$ set to 1 will be studied. On the experiment procedure, models with $\alpha$ equal to 0.41 and 0.51 were made along others, but this two are identical. Therefore, from this pair, we will study the one with $\alpha$ equal to 0.41.

In Figure 7.6 is seen that around 73% of the total traffic go directly to nodes 1 and 2. Node 1 makes reference to a set of pages that seems, like in the wiki dataset, more popular than others. In this case, users in this state visit the root page or other popular pages. Those popular pages include information related to the game's installation, downloads and tutorials of the game. These users seem to follow the "one time, one visit" behavior or pattern, but the probability of finishing the traffic in that state is almost 0. This is given by the fact that state 2 is a state of image requesting, that is seen almost in every page. Node 0 seems to contain the rest of the traffic. To see patterns in this node, we must do further studies with a higher $\alpha$ and

images filtered.

If the merging confidence is a little more restrictive giving an $\alpha$ equal to

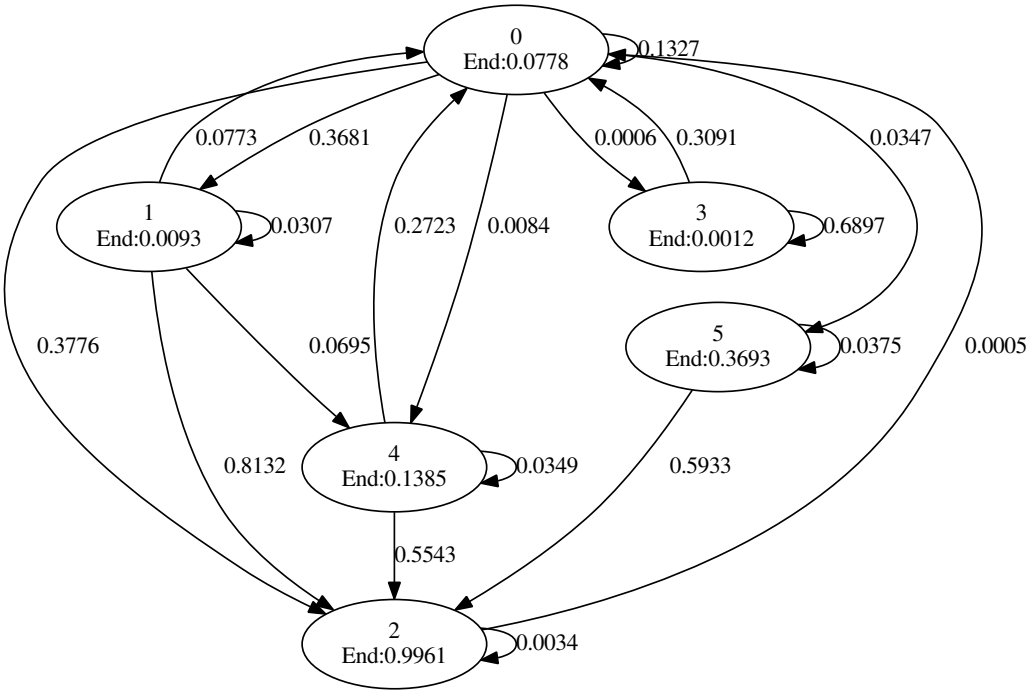



Figure 7.6: Game's news dataset's FSA inference graph with parameter $\alpha$=0.01 and $t_0$=1.

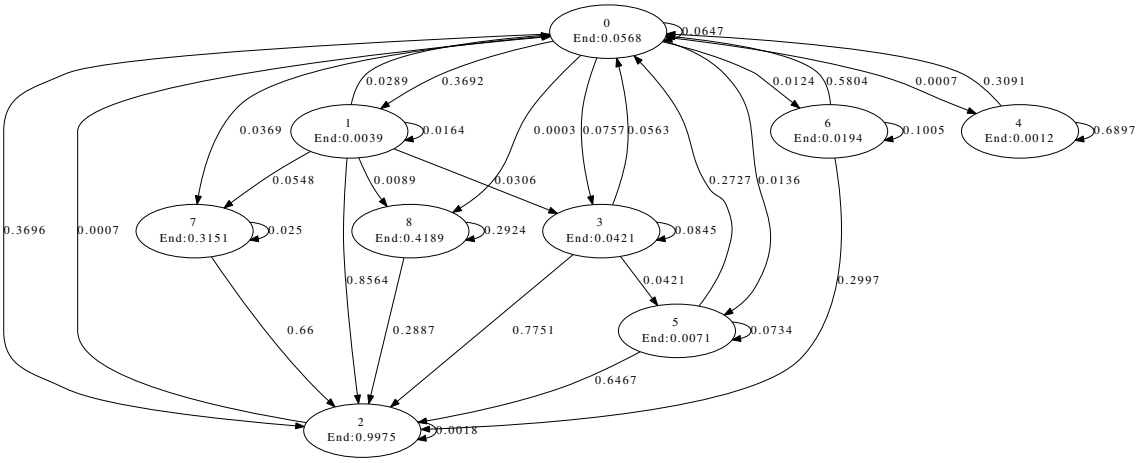



Figure 7.7: Game's news dataset's FSA inference graph with parameter $\alpha$=0.11 and $t_0$=1.

0.11, as shown in Figure 7.7, node 0 seems to be unmerged, but the noise is still too strong. For example, node 7 and 8 only contains CSS files. In order to have a better analysis, this kind of files must be filtered. State 5 seems to be a subset of the previously seen popular pages' set as it only contains details of game's installation and information about the third game. The other part of the popular pages' set is located in state 3, which is a previous step of state 5 in some occasions. This leads to the thought that maybe those popular sites are linked in some particular way.

**Conclusions** For further iterations of the analysis, images and CSS must be filtered. There seems to be a tendency to visit some particular pages, the previously called "popular pages", along other pages that, at the moment, are seen as random traffic.

### Iteration 2

**Clustering** With a $K$ set around 5 we see three different groups. The first are bots that ask for robotstxt and do not crawl. The bots present on this site are MSNbot-media, Bingbot, Attentio and R6_commentReader. First bot asks for robotstxt and then gathers an image. The other three just ask for robotstxt as web page administrator asks kindly in that file for bots to not crawl the website. Then we confirm the there is a set of "popular pages" from "articulos" folder visited in a "one time, one visit". These users come from search engines looking for installation guide of the game and interesting downloads for the game. Finally, we have users that mainly access to the news portal and after that some of them surf on the web.

**FSA inference** In the Figure 7.8 we can see the inferred FSA with an $\alpha$ set to 0.51. Node 0 simply acts as a fork between two paths. Important nodes in this graph are state 1 and state 3. The first state is the state where root and portal pages are most probable. Therefore, we can say that the behaviour found in this node is most likely to be "news reader". There is a transition with a big probability, 28%, which is produced by the "portada-destacados.php". This php file is a script loaded on root and portal pages to manage the news highlights bar found at the top of both web pages.

Node 3 is a node were the "popular pages", as said in clustering, are visited. This set of pages belong to the "one time, one visit" behaviour, as most of them end in this node and most of this websites are accessed from search engines or direct links from other websites. We have denominated this kind of behaviour "information seekers", as they come to the website in

order to find something very specific they are looking for, like the installation order of the game.

Finally, a very small portion of the traffic is concentrated in the node 2. This node represents the minority that seems to be using the web site's searcher. Although this behaviour seems to be a minority, we have considered it and denote as "inside-web searchers".

**Conclusions** Is clear that in this dataset we have two big tendencies "news readers" and "information seekers" behaviours. Both have a strong "one time, one visit" behaviour component, however, as "/w2/load.php" in wiki dataset, "portada-destacados.php" helps us to discern both behaviours.

Finally, there are also bots and "inside-web searchers" behaviour but they have less weight than the other two.

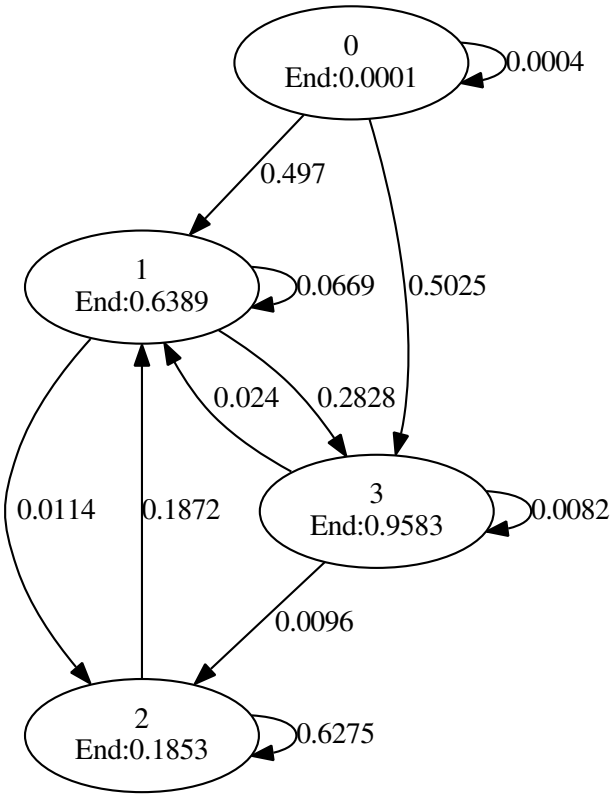



Figure 7.8: Game's news dataset's FSA inference graph in second iteration with parameter $\alpha$=0.51 and $t_0$=1.

## 7.3 Game's forum dataset

The game's forum dataset we are going to study has been gathered during a day. It has 417 different users and 51 different pages. The traces depth is in a range from 1 to 25 with a mean of 1'69 of depth. Most representative depth is 1 with 2322 occurrences. The number of occurrences with the depth follows a power law. Our hypothesis here is that most of the traffic will first access a category and then several threads to read.

### Iteration 1

**Clustering** For this analysis we will start taking a look at results with a $K$=2 and memory=0. Here, both clusters seems to be mixed but there is a "one time, one visit" behavior tendency in one of the groups that access to a "viewtopic.php" a remarkable number of times. From this we can conclude we have more than 2 groups; maybe we can find 4 different groups as there are 2 mixed clusters, so we will augment the $K$ by one and we will see if our hypothesis is correct or not. This time, with $K$=3, we can see that the "one time, one visit" behaviour cluster is there. The other two clusters are a mixed cluster and a cluster where a behaviour seems to be seen. This behaviour has a big component of "viewtopic.php". This kind of user seems to navigate from topic to topic as it was a "reader user" that comes to the forum periodically.

Now we have two clusters with a marked tendency, but one remains quite mixed. Therefore, we will try to use a $K$=4. With this $K$, the last found behaviour, "reader user", is split. Our hypothesis of the noisy cluster splitting with $K$=4 turns out being not true.

We will try to augment the memory by one and keep using 4 as $K$ in order to reward the similarity of longer traces. This time, only one cluster stays untouched, the "one time, one visit". Other clusters are redefined, showing interesting behaviours. The "reader user" gets sightly refined showing an interesting behaviour on some users who access to "/download/file.php". Our hypothesis is that some of these users comes from search engines looking for a specific file to download. From the search engine they go directly to "viewtopic.php" and then download. Therefore two behaviours may be mixed in this cluster. Furthermore, on one of the mixed clusters now we can find a more defined traffic where we see a strange behaviour arond page "/ucp.php". This page is in charge of register, login and logout of the users. In those nodes we observe a *looping ending* as shown in Figure 7.9

In this peculiar pattern, the value of sessions going to that page, $x$, is between 9 to 12 times smaller than $y$. This means that the same page is used

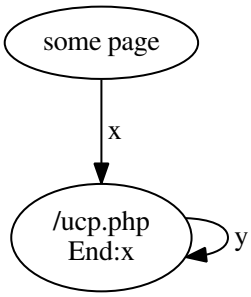



Figure 7.9: The ucp.php loop behaviour.

consequently for 9 to 12 times per user. Our hypothesis for this situation is that bots are trying to register this forum but they can't as the forum is protected with a question related to the game combining that with calculus. To pass that filter, a bot must have a powerful natural language processing mechanism as it implies know what information is being asked for and know how to do the calculus with that information. At first we thought that the cause was the registering process, that calls "ucp.php" 3 times; however, the ratio given by the division of $y$ and $x$ is, as mentioned before, over 9.

Images, as in the other two dataset, seems to be a source of noise. In this dataset noise was not too strong, but we think that we can obtain better results filtering images. Hence, we will filter them for the next iteration.

**FSA inference** This time taking a look at the results of the algorithm with a $\alpha$ parameter set to high confidence, $\alpha$=0.01, and $t_0$=2, we see that there are some nodes that do not get merged because they are not representative statistically. Most of these nodes are images so, as long as we still have images, we will continue with parameter $t_0$ set to 1.

In the previous datasets we observed a reduced number of nodes with these parameters but now we have 9 states, as seen in Figure 7.10.

Here we can see something that may catch our atention: from initial state 0, there is a 80% of probability to go to state 3 with the page "viewtopic.php". Users ending their traffic in state 3 are the same that in the clustering analysis fell into bot cluster, as this state can only be accessed from state 0. This tells us that we have an 80% of the traffic made by crawlers and, maybe, an user that access the topic from a search engine or a hotlink. We will define state 3 as suspected bot state, as not all traffic ends there, but traffic that ends there is more likely to be a bot.

After this, the biggest portion of traffic that exists state 0 is the one leading to state 6 requesting page "viewforum.php". There is only one page

that can lead to enter state 6, "viewforum.php", and it is used from the 3 entering points, states 0, 2, 3 and even the state 6 itself. Therefore we will associate this state directly with the page and hypothesize that a "forum overview behaviour" is represented here. This seems to be a normal and desirable behaviour in a forum, as we always start at forum categories and then go to the topics. The returning edge can be explained adding the concept of subcategory. As subcategories are possible and the same php file is used for them, it is totally normal for the traffic to be defined as it is.

What catches our eye next is the end probability of 1 and the number of edges that fall into that state. The pages with most probability to lead to this state are "posting.php" with a probability of 43% coming from state 4, "ucp.php" with a probability of 26% coming from state 7, "viewtopic.php" with a probability of 24% coming from state 6 and "download/file.php" with a probability of 7% coming from state 8. Our hypothesis here is that state 1 is a sink state as a 57% of the traffic end there, sharing a close connection with state 0, which also is a state were traffic tend to be terminated.

Finally, the last thing that catches our eye is the returning probability of state 5. This state is related to "ucp.php" and represents the same as mentioned in clustering: we may have bots on our website trying to register accounts. We may call this behaviour "frustrated malicious bot", as it seems that the robot protection on register works.

**Conclusions** We have discovered that we have users that seem to be regular readers, people that navigate through the category menus and two kinds of bots: crawlers and malicious bots.

In this dataset we have discovered a lot of behaviours and we have not been affected severely by images as in the other two datasets; however, we may see new things filtering them.

### Iteration 2

**Clustering** As we suspected, behaviour groups have not varied really much. We confirm the behaviours found previously plus the "file download" behaviour which, in fact, is divided in two: one that follows our hypothesis of coming from outside to download a file and the other forum readers that download files.

**Conclusions** No new information about behaviours was found in further FSA inference analysis. As the model did not vary much, we confirm our hypothesis that images does not make noise in this dataset. We confirmed

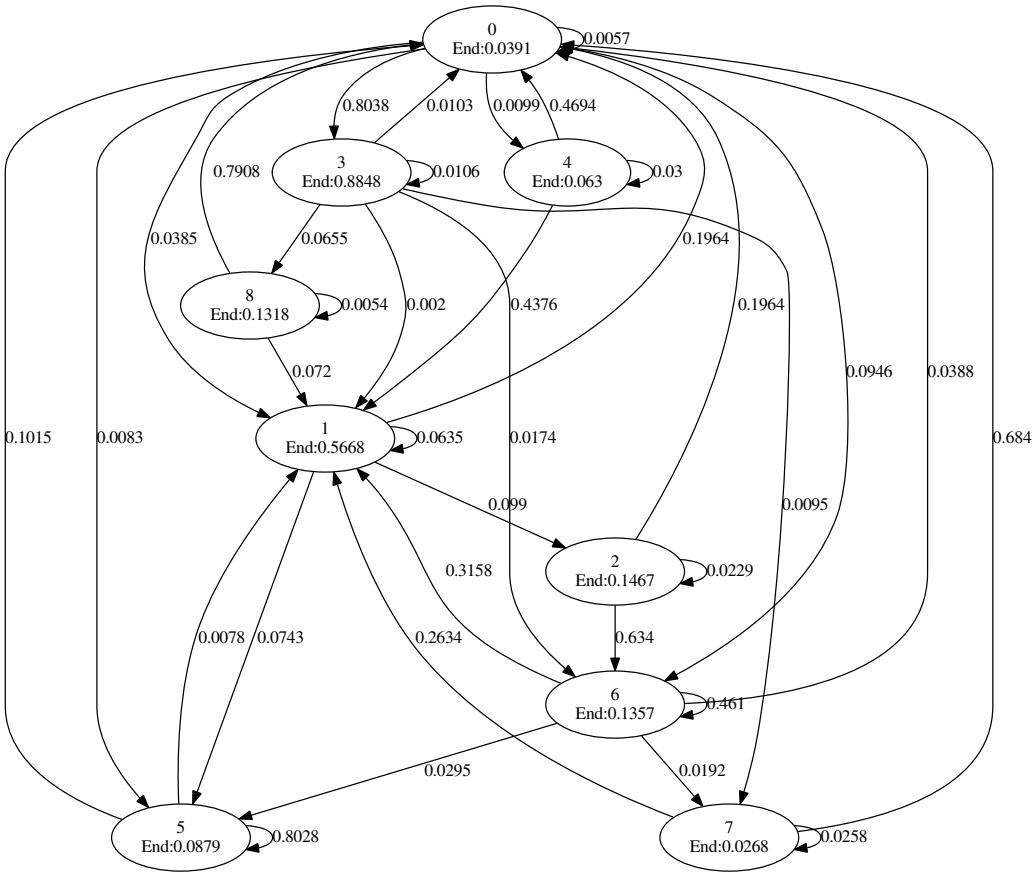



Figure 7.10: Game's forum dataset's FSA inference graph with parameter $\alpha$=0.01 and $t_0$=1.

that there are two kind of downloader behaviour "extern file downloader" and "reader downloader".

## 7.4 Conclusions

We have found behaviours that we expected to find and behaviours totally new. Now we summarize the found behaviours in our study and their relationship:

- One time, one visit behaviour: This behaviour is found in every dataset. A trace of an user with this behaviour is just requesting a page of the website and nothing more. This pattern is seen in several behaviours, therefore we say this is a behaviour that encloses other behaviours, in other words a pattern.

- Crawler: This behaviour is found in wiki and forum datasets. A trace of an user with this behaviour is like the "one time, one visit" behaviour but doing it for more than one page.

- Login/Register problem behaviour: This behaviour is found in news dataset. It is defined as many loop on the same page and then end. In this case, this behaviour is related to "ucp.php" page, which is used for registering, login and logout on the website. Repeated failures on the captcha of the register process produced this behaviour. This behaviour is also known as "frustrated malicious bot", as makes reference to an account registering bot that has failed in its task.

- Popular pages behaviour: This behaviour is found in both, wiki and news dataset. In the wiki, this behaviour is related to visiting wiki articles that are more popular than the others. We discovered that popular pages were associated with human users and unpopular pages were associated to bots that crawled the wiki. The main reason to do so is that users may come from the same web page through a link, as they did in this case, making some articles to be more visited than the others that have 1 visit from the crawler.

  In game's news dataset, this behaviour was seen in some pages accessed in a "one time, one visit" behaviour fashion. The pages referred to content like how to install and download interesting stuff for the game. As in the wiki, these pages were referred from a website from outside or a search engine.

- News reader behaviour: This behaviour is found in news dataset. Users with this behaviour comes to the website to take a look at the news of the game and, in most of the cases, close the site. This behaviour may change into "inside-web searcher" behaviour.

- Forum overview behaviour: This behaviour is found in forum dataset. This user iterates over page "viewforum.php". We supose the user takes a look at the forum to search for any interesting new thing. From this behaviour there is a tencende in changing to "regular reader behaviour".

- Regular reader behaviour: This behaviour is found in forum dataset. This user iterates over page "viewtopic.php" reading messages from topics. This user may change to "forum overview" behaviour or "reader downloader" behaviour.

- Real reviewers behaviours: This behaviour is found in wiki dataset. These users looks at articles and then to revisions to acknowledge changes or edit. The "real" from the names comes from bots crawling revision pages without doing nothing.

- Information seekers behaviour: This behaviour is found in news dataset. For the definition of this behaviour, refer to "popular pages" behaviour. Follows the "one time, one visit" behaviour style.

- Inside-web searchers behaviour: This behaviour is found in news dataset. This behaviour refer to users that use the web page searcher to look for something.

- Downloader behaviours: This behaviour is found in forum dataset. These users downloads files uploaded on the forum. This behaviour is divided in the two following.

- Extern downloader behaviour: This behaviour is found in forum dataset. These users downloads files uploaded on the forum coming from outside of our page.

- Reader downloader behaviour: This behaviour is found in forum dataset. These users downloads files uploaded on the forum after surfing on the forum.

# Chapter 8

# Conclusions

## 8.1 Final conclusion of the project

After almost a year of work, we have achieved our goals. The main goal, developing a tool to create models for behaviour inference, was accomplished with good results. Now we can generate models from an access log of a web site that allows us to have more actionable knowledge about our web site. We have shown in Section 7 how an analysis can be performed with the resulting models.

As the application only uses traffic shape and the page name, we have developed a tool that can be applied to every kind of website. On the other hand, this versatility produces that the results must be analyzed by a human to conclude anything from them but with a no time-consuming process.

Applying clustering in clustering with this kind of data was not new at all. However, application of FSA inference was somehow new. We think that this kind of algorithm based on grammar recognition can be used on this data, being the possible ways to use the web site *the grammar* and each of trace of usage a word. This is only possible if we convert the traffic in linear traces, in other words we split traffic as done for the FSA inference in this project. By this means, we could use the theory already developed for grammars on traffic.

In terms of my education as engineer, this is the first real project that I lay my hands on. Some projects during the studies are intended to be this way but the feeling is quite different. This project has taught me to work harder, more constantly and research by myself in a topic. This autonomy is a trait desired on an engineer as engineers must know what to do and research how to do so.

With this project I have learned about Data Mining field, which I only

heard about it during the Machine Learning subject. I have found it useful and, somehow, really interesting. Moreover, the study of how the traffic works has made me learn the different parts of a website and how does the interaction works, from the user to the server that hosts the site.

Finally, FSA inference and grammars has been the most interesting part to learn. Even though the mathematical background was somehow hard to grasp, when everything became clear also became amazing. With this part came back to a long lost friend, base of several things and tool for some others, mathematics.

Because of these reasons I feel that I need to research more this way, finding new things and applying them to world's known problems, in other words, researching.

The tool developed is far from being ready for commercialization in the current version as there is a lot of work to be done and things to be improved. Yet, I consider this project a huge success both in the tool developed and personal growth. I think that if we keep working on it, we will get a really useful tool and easy to use, this last being part the biggest drawback of the current tool. We have set a list of points to attack in the future work list that can be found in Section 8.2.

## 8.2 Future work

Finally, in this section we define our lines of future research for this project.

- We would like to test fuzzy clustering so we can assign a user to two different kinds of behaviour and see what happens, as we have observed several clusters mixed that are hard to break. This study can tell us how many of our users have hybrid (more than one at time) behaviours.

- Another good idea can be to apply hierarchical clustering may be a good idea when we do a clustering and some of the groups seems to remain heterogeneous. We could implement a threshold for the sum of the distances of every point to its centroid in order to know if that cluster can be divided or not.

- Web parameters in PHP web sites can have a lot of importance as they can determine which content is loaded. Given this, a future extension of this project could be to do an automatic analysis of parameters to decide if that parameter is important to make nodes with the same .php file distinct or not.

- Parsing module can be improved using a model like some compilers do. First, you translate the code into an "interlingua" language and then you translate this "interlingua" to our data structures. This way we only would need to do translator for "log languages" instead of change everything. There are translators from different kinds of logs like the Apache's log to a standard called XES [36], introduced for process mining. XES is a customizable XML format that provides the structure needed for our purposes.

- Filtering system can be improved adding flexibility supporting more wildcards. Moreover, a categorizator could be implemented to add more semantics to the analysis. This categorizator would change the identificator of a web page for a string given. This way we can group web pages that we consider similar or that we have seen with an analysis that they form a block, for example.

- Improve user interface is a big point to have into account for this project. At the moment, the software is just a terminal client which does the analysis and prints the outputs in files. Future interface must have better user interface elements like a bar to vary the parameters of the algorithms, hide edges and so on.

- Improving the system to do faster analysis is mandatory. We had a big dataset that we could not analyze because of the time consumption of the process. We think that using databases may be a good thing for really big datasets. Moreover, we want to study the usage of Apache Spark GraphX [37].

- For the future, we would like to experiment more with the software. Obtaining datasets for this project is difficult as we need to get in touch with the administrator of a website and, then, be allowed to use them (something difficult because of privacy issues).

# Appendix A

# Evaluation of K-Means algorithm

## A.1 Introduction

Our goal in this appendix is to know even more our data and our tools in clustering. We will be studying how the different parameters of the K-Means algorithm affect to the result.

**Study environment.** In our environment will be managing the following input parameters: $K$, memory, number of K-Means repetitions (iterations) and the initialization function for K-Means. Given this input, the resulting output variables will be: time, solution quality and K-Means round mean.

We will do a combinations of K in a range from 2 to 10, memory from 0 to 5 and number of repetitions from 1 to 100 in order to see how the output variables get modified. We will repeat this for the two available initialization functions: Forgy and K-Means++. Finally, we will use three different datasets for this task: the game's news site, the game's forum and the wiki. Moreover, as the executions are not idempotent because of the random component of the initialization, we must do repetitions of these experiments in order to not fall into outliers. Because of the CPU time cost of each experiment is high, we have chosen to do 10 executions of this experiments with all the possible input parameters combination. They will be done with the Llenguatges i Sistemes Informàtics (LSI)'s computational cluster as doing it in a commodity computer would be too expensive in time. Thanks to the cluster's architecture time will be reduced dramatically. Given that the cluster is used by more users, the times resulting on executions may not have maximum precision, however this will be checked by doing the previously

mentioned 10 executions and using means for the plots.

**Quality** We measure quality using the sum of the distance between all points and its respective centroids. Therefore, the representation of quality will be inverted. This means that the lower the value is in the plot, the better the quality is.

## A.2 Study

As seen in Section 5.2, K-Means is an algorithm that will fall into a local optimum. In order to fix this issue we proposed to use more than once the algorithm and compare the results to get the best available.

The first hypothesis we want to confirm with this study is if 100 repetitions of the algorithm is enough to obtain a solution that can be considered good. For this issue we will be comparing the mean of quality with how many repetitions we do.

As seen in Figure A.1, we see that around 20 iterations the quality of the solution tends to get better in a less pronounced way solution. Solutions with greater iteration numbers seems to stabilize with a logarithmic tendence. This means that quality tend to grow more in the first steps than in further steps. Our conclusion is that after 20 iterations we get a considerable good solution, but more iterations are encouraged to arrive to the solution stability.

In A.2 we can see that this logarithmic shape is also there. Some of the datasets tend to give better solutions than others. One of the known factors involved in this is the size of the dataset, being the news site dataset the biggest one. If we have more users, we have more probability of having different users in base of the number of different pages we have because of combinatorial factor.

As we can observe in Figure A.3 the standard deviation stabilizes after 10 iterations, meaning that after that point the range of quality in the different solutions of the N repetitions of the algorithm grows in a linear fashion given that every solution grows with the mean as the standard deviation is almost constant. On the other hand, this variation means that, given the random component of the algorithm, the global optimum solution is not reached every time, we get a lot of solutions that are local optimums. In order to obtain the global optimum always, we would have to repeat this algorithm a number of times that would be really big, which is not interesting for our project as what we want is a fast solution that is good enough.

Finally, in order to see which number of repetitions would be good for the three datasets we will take a look to Figure A.4. This plot shows that, after

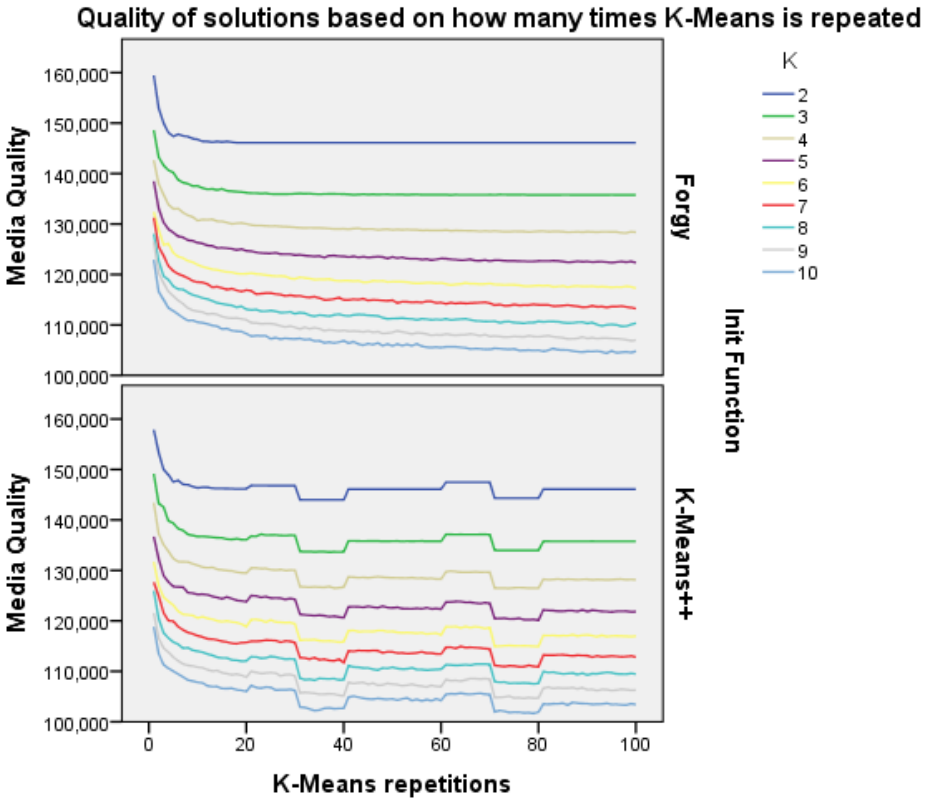



Figure A.1: Quality mean vs K-Means repetitions with different functions and Ks.

20 repetitions, the solution seems good. However, a good number would be around 80 because the pendent seems to be less than in previous points like 60 and 40. For the K-Means++ function we see some irregularities that we have observed before but we have not commented yet. Due to the randomness of the experiments, it seems that sometimes the K-Means++ function can find a better solution than the counterpart, Forgy, but then get worse. Overall, K-Means++ function seems better than Forgy in terms of quality.

After seeing the effect of number of repetitions and which function seems to be better in quality, we want to see if the memory affects to the solution quality. In Figure A.5 we can see how the memory affects the quality in examples that use 100 K-Means. As opposite of what we though about memory, seems to not affect at all to solution quality.

After we have seen the effects of parameters in quality, we will see how they affect to time. First we will see in Figure A.6 how does the dataset affect to time. Here we see that the slower dataset to analyze is the news site, then

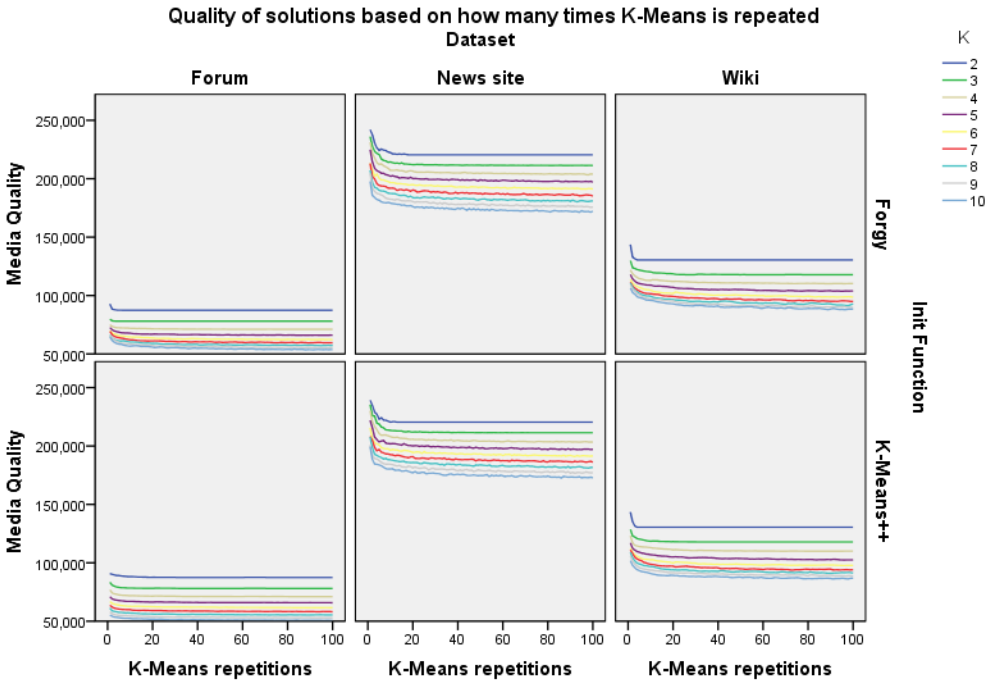



Figure A.2: Quality mean vs K-Means repetitions with different functions, Ks and dividing the datasets.

the Forum and then the wiki. This is due to the size of the datasets and, maybe, due to the difficulty of doing the clustering itself with each dataset, which is correlated in some sense with the size. We will see this in further plots that involves the K-Means round mean.

K-Means++ function has more process inside than Forgy, however times are not dramatically bigger. In Figure A.7 we can observe that the times' mean are really close, however K-Means++ tend to have outlier cases that may be given to the overload of the testing machines or the algorithm itself but the times obtained are still good.

In Figure A.8 we can observe the differences on time based on the memory. Memory may affect the time because the probability to add new dimensions to our points as we difference more nodes given their fathers and ancestors and stablish them as new dimensions.

As this process includes the graph reconstruction, memory 0 and memory 1 differ a lot. However, after that point, the first and third quartile seems to stablish. After that point what varies is the mean. This confirms that memory has an effect on time because of the adition of new dimensions.

Finally we will see how K-Means round mean change with some parame-

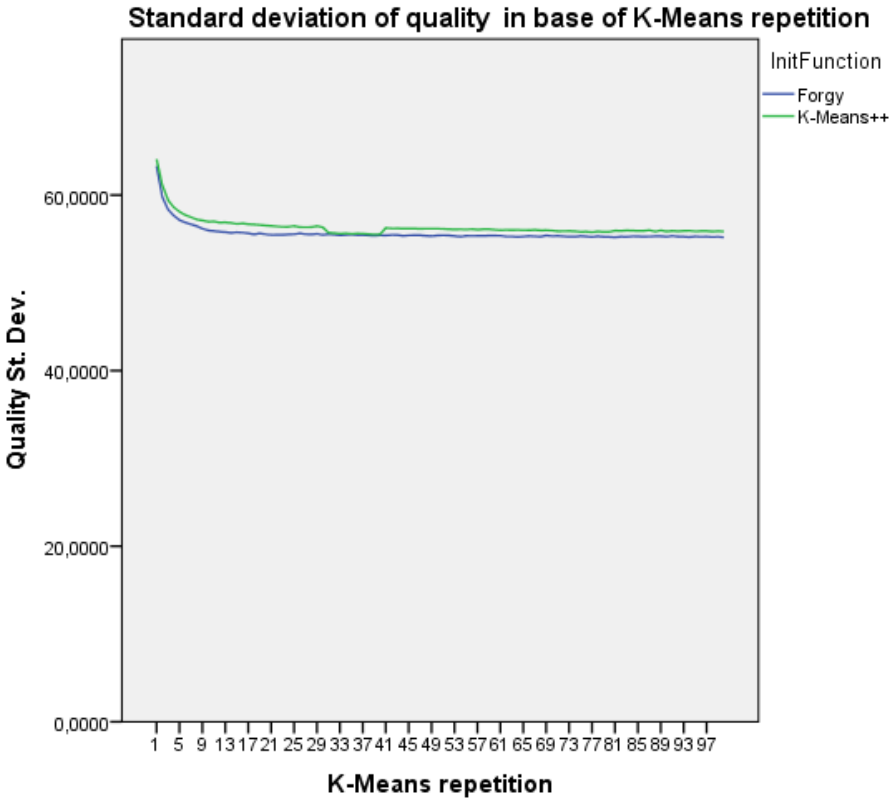



Figure A.3: Quality standard deviation vs K-Means repetition with different functions.

ters. First we will compare round mean with the datasets in Figure A.9. In this figure we see that our hypothesis of round mean affecting the time seems to be erroneous in terms of complexity. The dataset which more rounds takes is the forum followed by the wiki and, then, the news site. Ironically, the news site dataset is the one which takes more time for the calculation. There are two reasons for this: the size of the dataset in the K-Means distance calculations and the size in graph reconstruction. This size seems to affect more than the complexity of dividing the dataset itself.

Now we will take a look to the round mean in terms of memory. As seen in A.10, round mean takes a logarithmic shape. This is due to the tree's shape as we have more elements that of depth 1 than depth 2, etc.

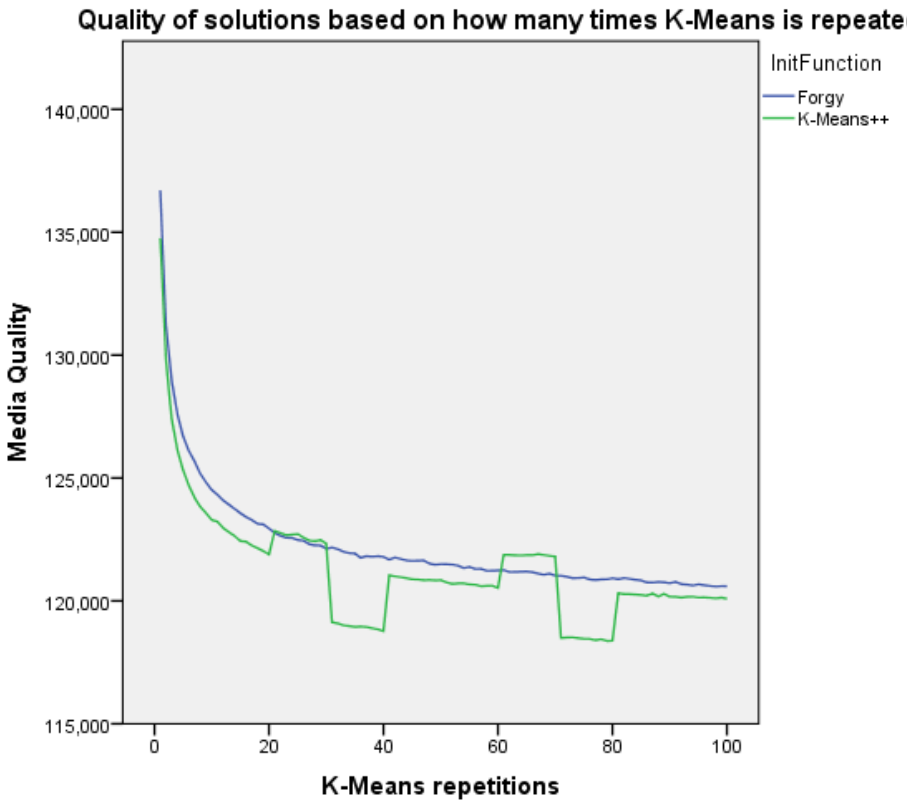



Figure A.4: Quality VS Algorithm iterations with different functions.

## A.3 Conclusions

We have confirmed that we need the K-Means re-iteration process in order to obtain a good solution. We have also determined that a repetition number over 20 is requiered in order to obtain a well adjusted solution. A number of iterations near 80 will be sufficient to match our requierements. We have also seen that K-Means++ may have a greater overhead than Forgy but the variation is acceptable, the solutions given are better and the number of rounds, interesting in case we have a bigger dataset, are less.

We have determined that the datasets clustering time is mostly biased by size, confirming that the algorithm works with a polynomial cost in base of the size. Finally, contrary to our hypothesis, memory does not seem to affect too much to solution's quality.

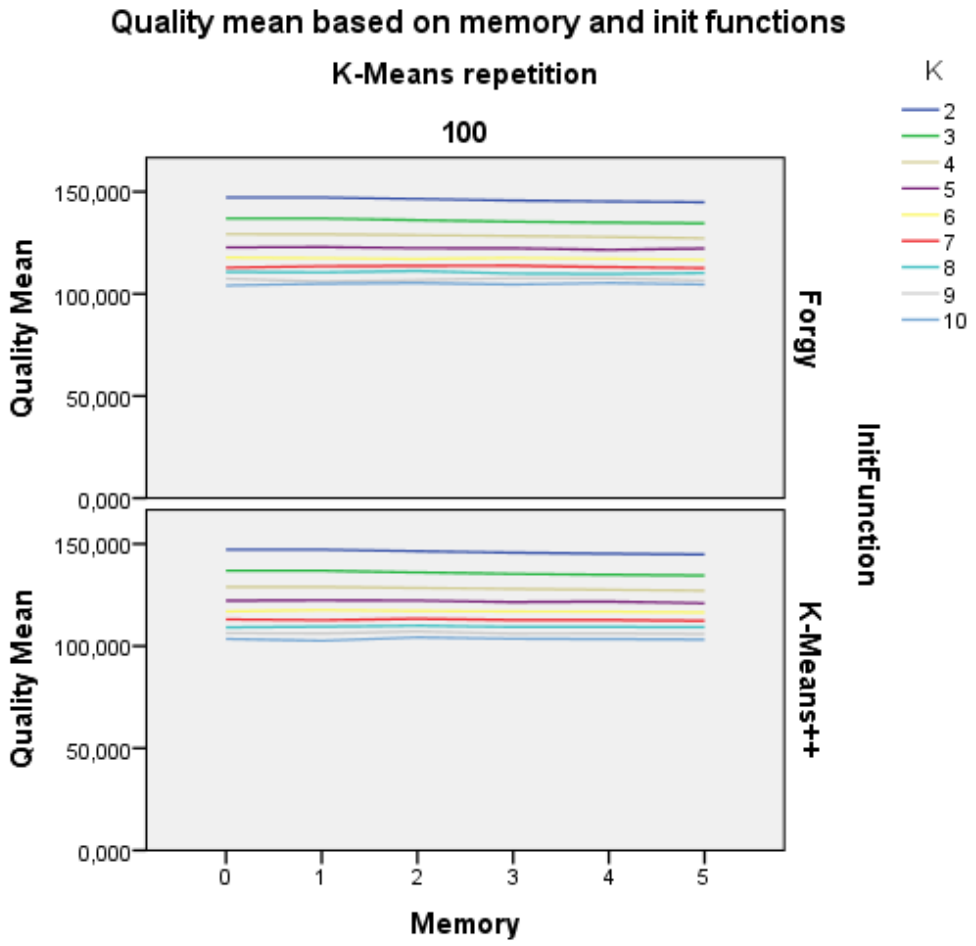



Figure A.5: Quality VS Memory with functions.

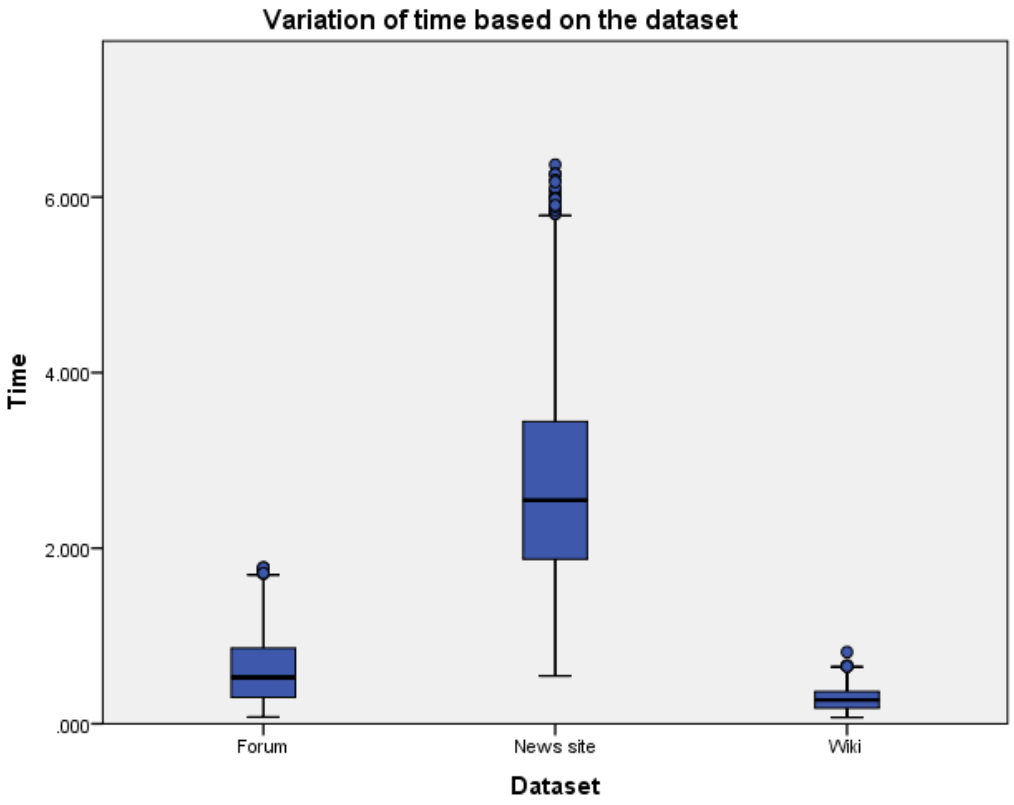



Figure A.6: Time VS Dataset

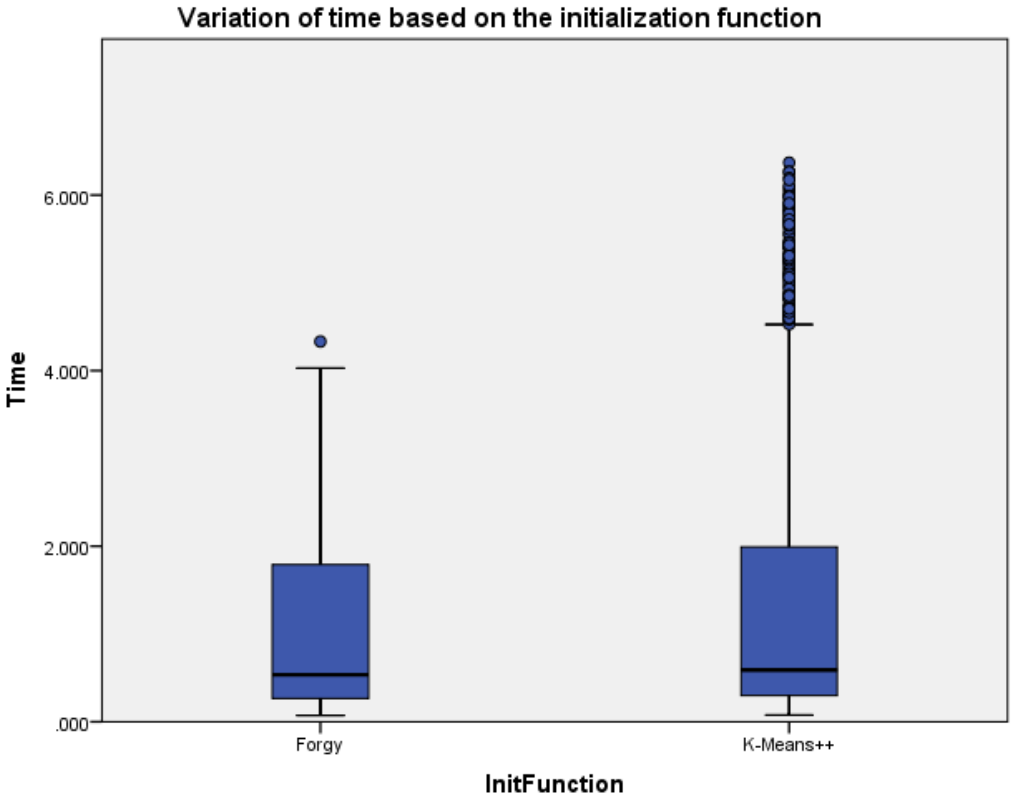



Figure A.7: Time Vs Initialization function

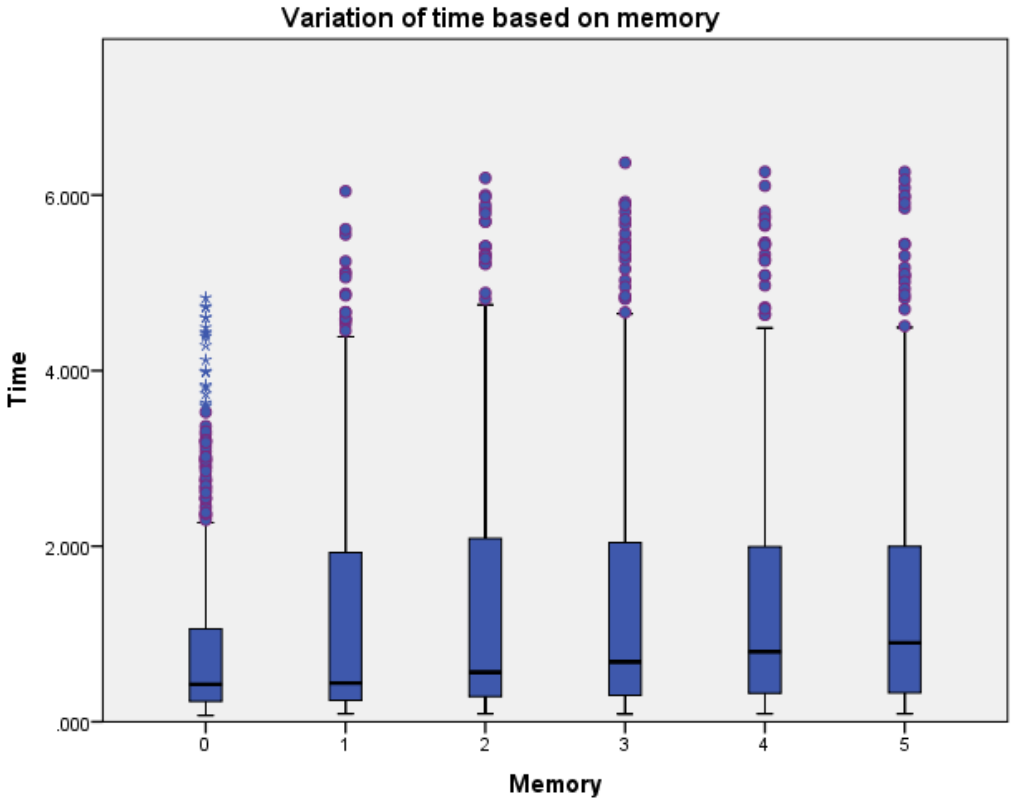



Figure A.8: Time VS Memory

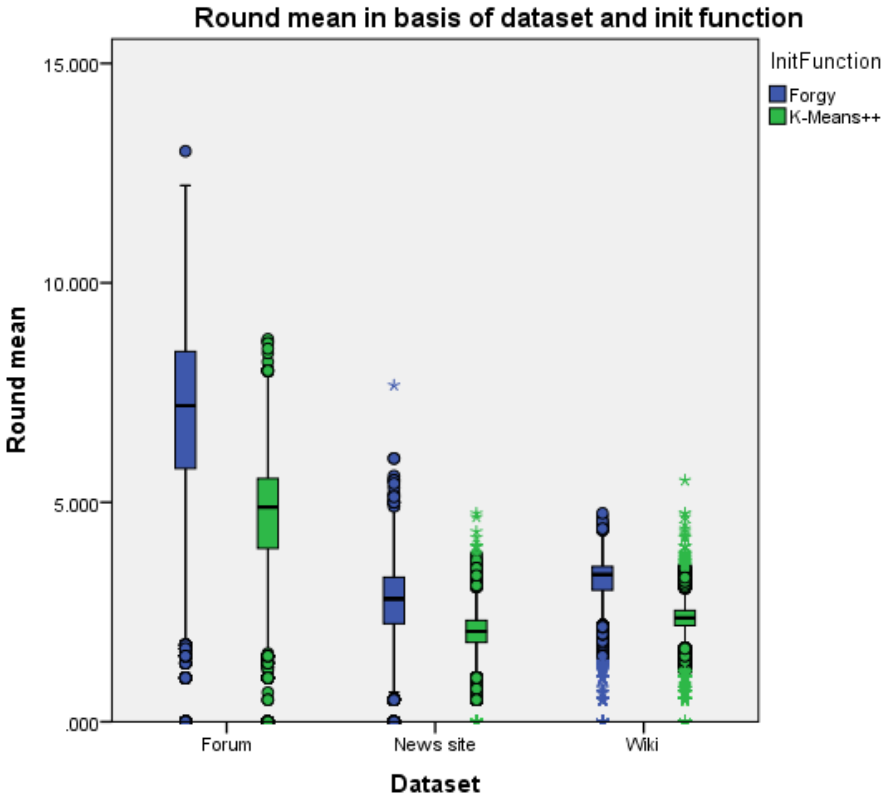



Figure A.9: Round mean vs Dataset with function

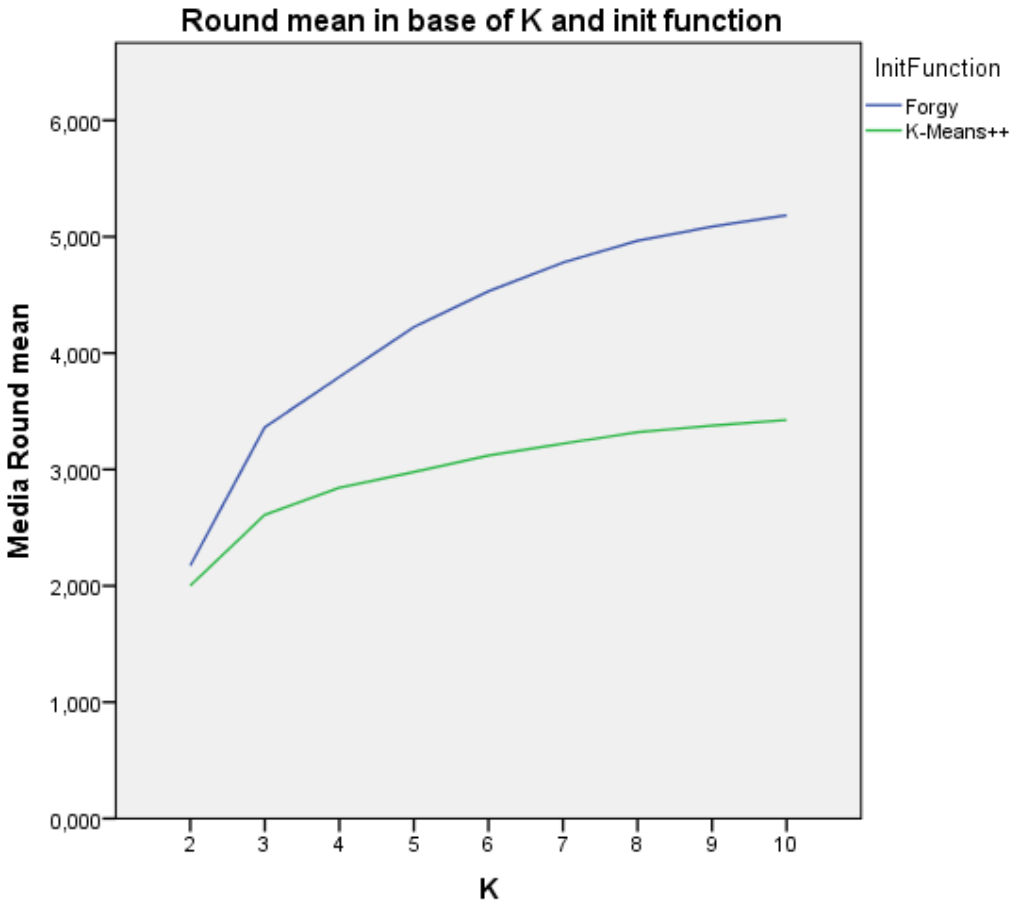



Figure A.10: Round mean vs K with function

# Glossary

$K$ $K$ parameter used for K-Means algorithm. This parameter defines which is the number of different clusters that the resulting model will have. 51, 52, 61, 79, 80, 84, 85, 87, 89, 99

$\alpha$ In the scope of this project, $\alpha$ is a parameter of Red-Blue algorithm. This parameter is a confidence level given for merging in the algorithm. 81–88, 90, 92

$t_0$ t0 is a parameter of Red-Blue algorithm.This parameter is the minimum number of attested strings passing a state in the prefix tree to consider merging. 75, 79–86, 88, 90, 92

**cache** A cache is a memory element designed for fast access in an already demanded content. 29, 30

**CSS** Cascading Style Sheets (CSS) is a style sheet language used for describing the look and formatting of a document written in a markup language [1]. 38, 84, 87

**favicon** Favicon.ico is the icon of a website that's shown in bookmarks, top of the web browser's window, etc. 38, 82, 84

**hotlink** Hotlinking is an hyperlink directly to an image of a website embedded in another website without going through any of the pages of that website.. 80, 81

**mediawiki** Mediawiki is a software used to power wiki sites, like Wikipedia. 81–83

**memory leaks** Memory leaks are memory non-freed that can be produced when memory is reserved by using malloc or new C++ instructions. When the memory is reserved and not freed, this memory remains occupied and this may lead to memory problems if we need to reserve more memory or other processes needs it.. 47

**no reference** This phenomenon happends when a user demand any of our available content with a direct petition, without coming from another webpage. This can be done, for example, entering the full URL directly in the webbrowser.. 81, 85

**regular expression** A regular expression is a string which forms a search pattern. This is mainly used to do pattern matching with strings. 37

**robotstxt** Robots.txt is the a text file that every polite web crawler looks before crawling a website. In this file folders can be marked to tell the crawlers to not look on them. 82, 87

**stakeholder** A corporate stakeholder is that which can affect or be affected by the actions of the business as a whole.[1] In this project the stakeholder is usually the administrator of the website that provided the dataset to study. 81

**URL** A uniform resource locator, abbreviated as URL, is a specific character string that constitutes a reference to a resource.[1]. 112

**vectorial instructions** CPU instructions that can process more than one item at time. 61

**web crawler** A Web crawler is an Internet bot that systematically browses the World Wide Web, typically for the purpose of Web indexing. A Web crawler may also be called a Web spider, an ant, an automatic indexer, or (in the FOAF software context) a Web scutter.[1]. 81, 82

**wiki** A wiki is usually a web application which allows people to add, modify, or delete content in collaboration with others. Text is usually written using a simplified markup language or a rich-text editor. While a wiki is a type of content management system, it differs from a blog or most other such systems in that the content is created without any defined owner or leader, and wikis have little implicit structure, allowing structure to emerge according to the needs of the users.[1]. 79

**wildcard** A wildcard character is a special character that has a function in the language. For example, * can mean every possible combination of characters in bash. In regular expressions, * means repetitions of a symbol or a set of symbols. For example (ab)*, which means "ab" repeated from 0 to n times ($\lambda$,ab,abab,abababab...). 37

[1]Extracted from wikipedia

# Bibliography


[1] Wil M. P. van der Aalst. *Process Mining - Discovery, Conformance and Enhancement of Business Processes*. Springer, 2011.

[2] IEEE Task Force on Process Mining. Process mining manifesto. In Florian Daniel, Kamel Barkaoui, and Schahram Dustdar, editors, *Business Process Management Workshops (1)*, volume 99 of *Lecture Notes in Business Information Processing*, pages 169–194. Springer, 2011.

[3] Nicolás Poggi, Vinod Muthusamy, David Carrera, and Rania Khalaf. Business process mining from e-commerce web logs. In Florian Daniel, Jianmin Wang, and Barbara Weber, editors, *BPM*, volume 8094 of *Lecture Notes in Computer Science*, pages 65–80. Springer, 2013.

[4] Nicolás Poggi, Toni Moreno, Josep Lluis Berral, Ricard Gavaldà, and Jordi Torres. Web customer modeling for automated session prioritization on high traffic sites. In Cristina Conati, Kathleen F. McCoy, and Georgios Paliouras, editors, *User Modeling*, volume 4511 of *Lecture Notes in Computer Science*, pages 450–454. Springer, 2007.

[5] Nicolás Poggi. *AUGURES, Profit-Aware Web Infrastructure Management*. PhD thesis, Polytechnic University of Catalonia, Barcelona, May 2014.

[6] Prom 6. `http://www.promtools.org/prom6/`. [Online; last access on 13-May-2014].

[7] Bing Liu. *Web Data Mining: Exploring Hyperlinks, Contents, and Usage Data (Data-Centric Systems and Applications)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

[8] Myra Spiliopoulou and Lukas Faulstich. Wum - a tool for www ulitization analysis. In Paolo Atzeni, Alberto O. Mendelzon, and Giansalvatore Mecca, editors, *WebDB*, volume 1590 of *Lecture Notes in Computer Science*, pages 184–103. Springer, 1998.

[9] HTTPerf homepage. http://www.hpl.hp.com/research/linux/httperf/. [Online; last access on 18-May-2014].

[10] Wikipedia. Extreme Programming. http://en.wikipedia.org/wiki/Extreme_programming. [Online; last access on 01-June-2014].

[11] NetCraft. April 2014 Web Server Survey. http://news.netcraft.com/archives/2014/04/02/april-2014-web-server-survey.html. [Online; last access on 18-April-2014].

[12] Apache Software Foundation. Apache HTTP Server - mod_log_config documentation. http://httpd.apache.org/docs/2.2/mod/mod_log_config.html. [Online; last access on 18-April-2014].

[13] Nginx. Module ngx_http_log module documentation. http://nginx.org/en/docs/http/ngx_http_log_module.html. [Online; last access on 18-April-2014].

[14] W3C Extended Log File Examples (IIS 6.0). http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/ffdd7079-47be-4277-921f-7a3a6e610dcb.mspx?mfr=true. [Online; last access on 18-April-2014].

[15] HTTP/1.1: Method Definitions. http://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html. [Online; last access on 18-May-2014].

[16] QTCreator - QT Project. http://qt-project.org/wiki/category:tools::qtcreator. [Online; last access on 08-Jun-2014].

[17] Boost C++ Libraries. http://www.boost.org/. [Online; last access on 08-Jun-2014].

[18] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library - User Guide and Reference Manual*. C++ in-depth series. Pearson / Prentice Hall, 2002.

[19] Treba - Command-line tool for training, decoding and calculating Probabilistic Finite State Automata (PFSA) and HMM. https://code.google.com/p/treba/. [Online; last access on 08-Jun-2014].

[20] Graphviz - Graph Visualization Software. http://www.graphviz.org/. [Online; last access on 08-Jun-2014].

[21] GDB: The GNU Project Debugger. http://www.gnu.org/software/gdb/. [Online; last access on 08-Jun-2014].

[22] Valgrind Home. `http://valgrind.org/`. [Online; last access on 08-Jun-2014].

[23] Gprof2Dot - Conver profiling output to a dot graph. `https://code.google.com/p/jrfonseca/wiki/Gprof2Dot`. [Online; last access on 08-Jun-2014].

[24] Git. `http://git-scm.com/`. [Online; last access on 08-Jun-2014].

[25] Joos C. A. M. Buijs, Boudewijn F. van Dongen, and Wil M. P. van der Aalst. A genetic algorithm for discovering process trees. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2012.

[26] Pang-Ning Tan, Michael Steinbach, and Vipin Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[27] Ian H. Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques, Second Edition (Morgan Kaufmann Series in Data Management Systems)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2005.

[28] Lloyd's algorithm. `http://en.wikipedia.org/wiki/K-means_clustering`. [Online; last access on 4-May-2014].

[29] E. Forgy. Cluster analysis of multivariate data: Efficiency versus interpretability of classification. *Biometrics*, 21(3):768–769, 1965.

[30] David Arthur and Sergei Vassilvitskii. k-means++: the advantages of careful seeding. In Nikhil Bansal, Kirk Pruhs, and Clifford Stein, editors, *SODA*, pages 1027–1035. SIAM, 2007.

[31] Rafael C. Carrasco and José Oncina. Learning stochastic regular grammars by means of a state merging method. In Rafael C. Carrasco and José Oncina, editors, *ICGI*, volume 862 of *Lecture Notes in Computer Science*, pages 139–152. Springer, 1994.

[32] Kevin J. Lang, Barak A. Pearlmutter, and Rodney A. Price. Results of the abbadingo one dfa learning competition and a new evidence-driven state merging algorithm. In Vasant Honavar and Giora Slutzki, editors, *ICGI*, volume 1433 of *Lecture Notes in Computer Science*, pages 1–12. Springer, 1998.

[33] Jorge Castro and Ricard Gavaldà. Learning probability distributions generated by finite-state machines. 2012.

[34] Toni Cebrián, René Alquézar, and Alberto Sanfeliu. Learning probabilistic finite state automata for opponent modelling. Master's thesis, 2011.

[35] W Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American Statistical Association*, 1963.

[36] XES - Process Mining. `http://www.processmining.org/logs/xes`. [Online; last access on 09-Jun-2014].

[37] Apache Spark GraphX. `https://amplab.cs.berkeley.edu/publication/graphx-grades/`. [Online; last access on 12-Jun-2014].