A Data Mining Model
to
Capture User Web Navigation Patterns

José Luís Cabral de Moura Borges

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Doctor of Philosophy
of the
University of London.

Department of Computer Science
University College London

July 2000
Abstract

This thesis proposes a formal data mining model to capture user web navigation patterns. Information characterising the user interaction with the web is obtained from log files which provide the necessary data to infer navigation sessions. We model a collection of sessions as a hypertext probabilistic grammar (HPG) whose higher probability strings correspond to the navigation trails preferred by the user. A breadth-first search algorithm (BFS) is provided to find the set of strings with probability above a given cut-point; we call this set of strings the maximal set. The BFS algorithm is shown to be, on average, linear in the variation of the number of iterations performed with the grammar's number of states. By making use of results in the field of probabilistic regular grammars and Markov chains, the model is provided with a sound foundation which we use to study its properties. We also propose the use of entropy to measure the statistical properties of a HPG.

Two heuristics are provided to enhance the model’s analysis capabilities. The first heuristic implements an iterative deepening search wherein the set of rules is incrementally augmented by first exploring the trails with higher probability. A stopping parameter measures the distance between the current rule-set and its corresponding maximal set providing the analyst with control over the number of induced rules. The second heuristic aims at finding a small set of longer rules composed of links with high probability on average. A dynamic threshold is provided whose value is set in such a way that it can be kept proportional to the length of the trail being evaluated.

Finally, a set of binary operations on HPGs is defined, giving us the ability to compare the structure of two grammars. The operations defined are: intersection, difference, union, and sum.
I dedicate this thesis to Inês.
Acknowledgements

This thesis was funded by the University of Porto and by a grant from JNICT. I also acknowledge the Department of Computer Science at UCL and the UCL Graduate School for the financial support given to attend conferences.

A very special word goes to Mark Levene, my supervisor, for the advice, encouragement, and friendship he was always available to give. I am also grateful to Tony Hunter, my second supervisor.

My thanks to all the people of GEIN for their support. A special thanks to João Falcão e Cunha whose help was crucial in my decision of doing the PhD abroad.

My thanks to the fellow students at UCL whose daily presence and sharing of experiences were very important to make my stay in London an invaluable experience. I would like to name Rafael Bordini, Ethan Collopy, Antonio Liotta, Hugh Mallinson, João Oliveira, Jorge Ortega-Arjona, Alberto Souza, Manlio Valdivieso-Casique, Kanta Vekaria and Nadav Zin. I also thank all the friends from Portugal whose frequent visits were much appreciated.

I would like to thank all my family for their support. A very special thanks to my mother and (in memoriam) to my father who made possible for me to achieve this. Finally, my thanks to Inês, Afonso, and Gonçalo for their unconditional love and support.
Contents

1 Introduction ................................................................. 13
   1.1 Problem Definition ..................................................... 13
   1.2 Contribution of the Thesis ............................................. 15
   1.3 Structure of the Thesis ............................................... 17

2 Background ................................................................. 19
   2.1 Hypertext and Hypermedia ........................................... 19
      2.1.1 The World-Wide-Web and Search Engines ............... 20
      2.1.2 The Navigation Problem ...................................... 23
      2.1.3 Hypertext Modelling ............................................ 24
      2.1.4 Hypertext Visualisation Tools .............................. 25
      2.1.5 Adaptive Hypertext ............................................ 26
      2.1.6 Web Navigation Assistants and Agents .................. 27
      2.1.7 HTML and XML ................................................. 29
   2.2 Data Mining and Knowledge Discovery ....................... 30
      2.2.1 Data Mining Techniques ..................................... 31
      2.2.2 Mining Association Rules and Sequential Patterns .... 32
   2.3 Web Data Mining ...................................................... 33
      2.3.1 Web Content Mining ......................................... 35
      2.3.2 Web Structure Mining ....................................... 36
      2.3.3 Web Usage Mining ......................................... 37
         2.3.3.1 Log Files and Data Preparation Issues ............ 38
         2.3.3.2 Web Usage Mining Techniques ..................... 40
### Contents

2.3.3.3 This Thesis Approach ................................................... 46
2.3.4 Privacy Issues in Web Data Mining ................................. 49

2.4 Probabilistic Languages .......................................................... 50
2.4.1 Context-Free and Regular Grammars .............................. 51
2.4.2 Probabilistic Regular Grammars ....................................... 53
2.4.3 Finite Automata ................................................................. 55
2.4.4 Markov Chains to Model Probabilistic Regular Grammars .. 57

2.5 Hypertext Probabilistic Automata .......................................... 59

2.6 Information Theory ............................................................... 60

2.7 Summary ............................................................................. 62

3 Hypertext Probabilistic Grammars ............................................ 64
3.1 Formal Definition ...................................................................... 64
3.2 HPG as a Markov Chain .......................................................... 76
3.3 The $N$-gram Model ............................................................... 79
3.4 Incremental Model .................................................................. 86
3.5 Exhaustive Computation of All Rules ................................. 88
3.6 The Entropy of a HPG .............................................................. 92
3.7 Summary ............................................................................. 94

4 Experimental Evaluation .......................................................... 96
4.1 Synthetic Data Experiments .................................................. 96
4.1.1 Synthetic Data Generation Method ................................. 96
4.1.2 Experimental Results .......................................................... 98
4.2 Real Data Experiments ........................................................... 101
4.2.1 Real Log Files ................................................................. 101
4.2.2 Experimental Results .......................................................... 102
4.3 $N$-gram Analysis ................................................................. 112
4.4 Entropy Analysis ................................................................. 114
4.5 Summary ............................................................................. 116
## Contents

5 Heuristics for Mining High Quality Navigation Patterns 117

5.1 Motivation ................................................................. 117

5.2 Iterative Deepening Heuristic .................................... 119

5.2.1 Synthetic Data Experiments ................................. 122

5.3 Fine-Grained Iterative Deepening Heuristic ............... 123

5.3.1 Experimental Evaluation .................................... 130

5.4 Inverse Fisheye Heuristic .......................................... 134

5.4.1 Experimental Evaluation .................................... 140

5.5 Summary ................................................................. 146

6 Operations on Hypertext Grammars 147

6.1 Motivation ................................................................. 147

6.2 Inferring a HWG from a HPG ................................. 149

6.3 Grammar Equality and Sub-grammars ...................... 154

6.4 Primitive Methods .................................................. 155

6.5 Binary Operations on Grammars .............................. 159

6.6 Summary ................................................................. 172

7 Summary and Conclusions 174

7.1 Summary of the Thesis ............................................. 174

7.2 Directions for Future Research ................................. 176

7.2.1 Incorporating Relevance Measures in HPG ......... 178

7.3 Applications of the HPG model ............................... 179

7.4 Final Remarks ......................................................... 181

A Details of the HPG Implementation 182

Bibliography 187

Index 202
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of a regular grammar.</td>
<td>52</td>
</tr>
<tr>
<td>2.2</td>
<td>An example of a probabilistic regular grammar.</td>
<td>55</td>
</tr>
<tr>
<td>2.3</td>
<td>The finite automaton corresponding to the grammar in Figure 2.1.</td>
<td>56</td>
</tr>
<tr>
<td>2.4</td>
<td>The transition diagram for the finite automaton in Figure 2.3.</td>
<td>56</td>
</tr>
<tr>
<td>2.5</td>
<td>Examples of Markov chains.</td>
<td>58</td>
</tr>
<tr>
<td>2.6</td>
<td>The Markov chain equivalent to the grammar given in Figure 2.2.</td>
<td>59</td>
</tr>
<tr>
<td>2.7</td>
<td>Entropy decomposition property.</td>
<td>62</td>
</tr>
<tr>
<td>3.1</td>
<td>An example of a web topology.</td>
<td>65</td>
</tr>
<tr>
<td>3.2</td>
<td>An example of a collection of user navigation trails.</td>
<td>65</td>
</tr>
<tr>
<td>3.3</td>
<td>The methodology of the hypertext probabilistic grammar model.</td>
<td>66</td>
</tr>
<tr>
<td>3.4</td>
<td>The production rules of the hypertext grammar corresponding to the web topology in the running example.</td>
<td>67</td>
</tr>
<tr>
<td>3.5</td>
<td>The transition diagram for the hypertext weighted grammar corresponding to the collection of trails in the running example.</td>
<td>71</td>
</tr>
<tr>
<td>3.6</td>
<td>The production rules of the hypertext probabilistic grammar corresponding to the collection of trails in the running example.</td>
<td>72</td>
</tr>
<tr>
<td>3.7</td>
<td>The strings in the language generated by the grammar in the running example for several values of the cut-point and of the $\alpha$ parameter.</td>
<td>75</td>
</tr>
<tr>
<td>3.8</td>
<td>An example of a HPG.</td>
<td>77</td>
</tr>
<tr>
<td>3.9</td>
<td>The absorbing Markov chain corresponding to the HPG in Figure 3.8.</td>
<td>77</td>
</tr>
<tr>
<td>3.10</td>
<td>The irreducible Markov chain corresponding to the HPG in Figure 3.8.</td>
<td>78</td>
</tr>
<tr>
<td>3.11</td>
<td>An example of a HPG whose equivalent irreducible Markov chain (for $\alpha = 0$) is periodic.</td>
<td>79</td>
</tr>
</tbody>
</table>
List of Figures

3.12 The trigram HWG for the collection of trails in the running example. 81
3.13 An example of the computation of the $\chi^2$ test. 84
3.14 An example of a conditional entropy plot. 86
3.15 The BFS exploration tree for $\alpha = 0$ and $\lambda = 0.6/6 = 0.1$. 89
3.16 Two HPGs to exemplify the entropy rate definition. 94

4.1 The variation of the number of iterations and the running time with the number of states. 98
4.2 The variation of the number of iterations with both the branching factor and the confidence. 99
4.3 The variation of the number of induced rules with the number of states. 100
4.4 The effect of the distribution parameter for the links weights in the number of iterations and the variation of the average rule length with the number of states. 101
4.5 The variation of the number of iterations and the number of rules with the number of states for the real data. 103
4.6 The variation of the number of induced rules and their length with the confidence threshold. 103
4.7 The top-ten pages for several values of the $\alpha$ parameter. 105
4.8 Characteristics of the rule-sets induced with $N = 2$ and $\delta = 0.5$ for several values of $\alpha$. 106
4.9 The percentage of pages in the web site covered by the best $X\%$ of the induced rules. 107
4.10 URL codification. 108
4.11 Example of real data rules mined with $\delta = 0.5, \theta = 1/n, $ and $\alpha = 1$. 109
4.12 Variation of the grammar size with the history depth and an histogram for the sessions’ length. 113
4.13 The $\chi^2$ test for some of the weeks in the real data. 114
4.14 Variation of the conditional entropy with the $N$ of the $N$ gram model. 114
4.15 Correlation between the grammar entropy and the number of rules. 115
5.1 Characteristics of the rule-set induced by the BFS algorithm. 118
5.2 Example of an exploration tree obtained with the ID heuristic for $k = 2$ and $\lambda = 0.15$. 121
5.3 Variation of the stopping parameter and of the percentage of rules mined with the exploration depth. 123
5.4 An example of an exploration tree obtained with the fine-grained heuristic when using the greedy criterion for $k = 5$ and $\lambda = 0.15$. 126
5.5 The contents of the sets of trails induced by the FG heuristic when using the greedy criterion for $\lambda = 0.15$. 129
5.6 The contents of the sets of trails induced by the FG heuristic when using the best-trail criterion for $\lambda = 0.15$. 129
5.7 The variation of the number of rules and the length of the rules with the fine-grained stopping criterion when the heuristic is applied to the synthetic data. 131
5.8 The performance of the fine-grained heuristic with the synthetic data. 132
5.9 The variation of the number of rules and the size of the rules with the fine-grained stopping criterion when applied to the real data. 133
5.10 The evolution of both the set of candidate trails and the rule-set with the stopping criterion for the real data. 134
5.11 An exploration tree for the inverse fisheye heuristic with $\theta = 0.49$ and $\delta = 0.6$. 138
5.12 The characteristics of the rule-set induced by the Inverse fisheye heuristic with synthetic data. 141
5.13 The performance of the inverse fisheye heuristic with synthetic data and comparison between the rule-sets obtained with the inverse fisheye and the BFS algorithm for real data. 142
5.14 Evolution of the characteristics of the rule-set obtained with the geometric version of the inverse fisheye heuristic with real data. 143
5.15 Distribution of the rules' length with synthetic data. 144
5.16 Distribution of the rules' length for real data with $\theta = 1/n$. 145
List of Figures

5.17 Distribution of the rules’ length for real data with $\theta = 2/n$. .... 145
6.1 An example of a generic HPG. ............................................. 150
6.2 An example of a HPG and its corresponding HWG. ............ 151
6.3 The relationship between HPGs, HWGs, and uHWGs. .......... 153
6.4 Examples of HWGs. ............................................................. 155
6.5 Two alternative interpretations when adding a link traversal to a HWG. 156
6.6 Two alternative interpretations when removing a link traversal from a HWG. ................................................................. 158
6.7 Example of two HWGs to illustrate the operations. .............. 162
6.8 Example of the computation of the sum of two HWGs. .......... 162
6.9 Example of the computation of the union of two HWGs. ....... 165
6.10 Two examples of the computation of the difference between two HWGs. 167
6.11 An example of the drawbacks of adopting a strict definition for the computation of the difference between two HWGs. .... 168
6.12 An example of the consequences of adopting a strict definition for the computation of the intersection between two HWGs. .... 170
6.13 An example of the computation of the intersection of two HWGs. ... 171
A.1 A list of the classes in the HPG model implementation. ....... 182
A.2 The more important methods in the HPG class. ................. 183
A.3 The methods in the RuleSet class. ....................................... 184
A.4 The methods in the Rule class. ............................................ 184
A.5 The Entity-Relationship diagram for the database used in the experiments. ................................................................. 186
List of Algorithms

3.1 The pseudo-code for the incremental algorithm to infer a HWG from a collection of trails. ........................................... 87
3.2 The pseudo-code for the Breadth-first search algorithm. .......... 90
5.1 The pseudo-code for the fine-grained heuristic with the best-trail criterion. 128
5.2 The pseudo-code for the geometric version of the inverse fisheye heuristic......................................................... 139
6.1 The pseudo-code for the method to increment the weight of a link in a HWG............................................................. 157
6.2 The pseudo-code for the method to decrement the weight of a link in a HWG............................................................. 159
6.3 The pseudo-code for the method to compute the sum of two HWGs. ................................................................. 163
6.4 The pseudo-code for the method to compute the union of two HWGs. ................................................................. 165
6.5 The pseudo-code for the method to compute the difference between two HWGs........................................................ 167
6.6 The pseudo-code for the method to compute the intersection of two HWGs............................................................ 172
Chapter 1

Introduction

1.1 Problem Definition

The explosive growth of the world-wide-web (known as the web) in recent years has turned it into the largest source of available online data. Recent studies estimate the web as having over one billion pages (Inktomi, 2000), and pages are being added and deleted every day. The web is an unstructured collection of pages and hyperlinks, and pages are accessed and provided by a wide variety of people with different backgrounds and interests. Moreover, the web doesn't provide its users with a standard coherent page design across web sites (or even within a site). The web is heterogeneous in its contents both in relation to the information available and to its quality. Currently, web pages are mainly designed using HTML. With HTML the definition of the structure of a web page is severely limited, therefore, the analysis of the content of web pages by automated tools is a challenging and difficult task. All these facts together imply that it is difficult for a user to find relevant information on the web.

Data Mining and Knowledge Discovery (Fayyad et al., 1996b) is a research discipline involving the study of techniques to search for patterns in large collections of data. The application of data mining techniques to the web, called web data mining (Cooley et al., 1997), was a natural step, and it is now the focus of an increasing number of researchers. Web data mining has been defined as the study of data mining techniques to automatically discover and extract information from the web. Such techniques are aimed at providing the content provider with methods to improve the quality of service.
of a web site, or at providing the individual user with navigation assistant tools that help to manage web data. In web data mining there are mainly three research directions being followed:

(i) web content mining,

(ii) web structure mining, and

(iii) web usage mining.

Web content mining is a research field focused on the development of techniques to assist a user in finding web documents that meet a certain criterion, see for example (Craven et al., 1998). Mining the link structure aims at developing techniques to take advantage of the collective judgement of web pages' quality which is available in the form of hyperlinks, see (Chakrabarti et al., 1999a), where links on the web can be viewed as a mechanism of implicit endorsement. Finally, web usage mining focuses on techniques to search for patterns in the user behaviour when navigating the web, see (Borges and Levene, 2000a).

Web usage mining techniques are useful both to the web master and to the individual user. When understanding the users preferences, characterised by the set of most popular trails, the web master can improve the site topology according to the business objectives. Such objectives can be, for example: to personalise web pages, to increase the average time a user spends in the site, or to introduce new pages in places which make them highly visible. Personalised web pages can be made available using techniques from adaptive hypertext (Brusilovsky, 1998), for which it is essential to understand the user navigation behaviour. By creating links between pages of popular trails we can increase the average time a user spends on a site and a new product can be given good exposure by placing links to its pages from a popular trail which includes related products. Knowing the popular trails also allows the web master to identify the pages where users frequently terminate their sessions so that the contents and layout of such pages can be improved. On the other hand, if the browser is set-up to record the user's navigation history, web usage mining techniques can be useful to the individual user. In fact, by using such techniques it is possible to infer the user's preferred trails. The user's individual trails can be seen as a representation of his knowledge of the web and
be used as a prediction technique to prefetch in advance pages the user may be interested in. In addition, the user would benefit from being able to compare and/or exchange his navigation patterns with those of his peers and, for example, identify the preferred trails which are unknown to him but are among the preferences of his peers.

When web users interact with a site, data recording their behaviour is stored in web server logs, which in a medium sized site can amount to several megabytes per day. Moreover, since the log data is collected in a raw format it is an ideal target for being analysed by automated tools. Currently, several commercial log analysis tools are available (Stout, 1997; Mena, 1999). However, these tools have limited analysis capabilities producing only results such as summary statistics and frequency counts of page visits. In the meantime the research community has been studying web usage mining techniques to take full advantage of information available in the log files. There have so far been two main approaches to mining for user navigation patterns from log data. In the first approach log data is mapped onto relational tables and adapted versions of standard data mining techniques, such as mining for association rules, are invoked, see for example (Chen et al., 1998). In the second approach techniques are developed which can be invoked directly on the log data, see for example (Borges and Levene, 2000a) or (Spiliopoulou et al., 1999).

In this thesis we propose a new model for handling the problem of mining navigation patterns from log data. We model the user navigation records, inferred from log data, as a hypertext probabilistic grammar whose higher probability generated strings correspond to the user’s preferred trails.

1.2 Contribution of the Thesis

In this thesis we propose a novel data mining model to capture user web navigation patterns. The aim of this work is to devise a model to represent the information contained in a collection of user navigation sessions and techniques to analyse such information. We aim at developing a model with a solid statistical foundation in order for it to be stable and capable of coping with the fast moving field of web technologies. We have therefore built on results from the theory of probabilistic regular grammars and Markov
1.2. Contribution of the Thesis

chains in order to provide the model with a sound theoretical foundation.

We model a collection of user navigation sessions as a hypertext probabilistic grammar (HPG) whose higher probability strings correspond to the user preferred navigation trails. A HPG is within the class of regular grammars and can alternatively be viewed as a finite Markov chain. We make use of the \(N\)gram concept to model the assumed user's memory when browsing the web. A chi-square test is used as the method to assess the order, \(N\), of the \(N\)gram model that gives the required trade-off between the model size and its accuracy in representing the user navigation sessions.

A breadth-first search (BFS) algorithm is provided to compute the set of grammar strings having probability above a given cut-point; we call such strings rules. We prove that the BFS algorithm is, on average, linear in the variation of the number of iterations with the grammar's number of states. The results of extensive experiments with synthetic and real data are reported. The cut-point enables the analyst to determine the minimum probability that a string must have in order to be considered a rule and consequently the size of the rule-set. However, a high value of the cut-point results in a small set of very short rules and a small value of the cut-point results in a very large set of rules. The manipulation of large rule-sets limits the model's performance; therefore, two heuristics are proposed which enhance the HPG capabilities by providing flexibility in the way a set of rules is induced from a HPG. Using these heuristics it is possible to analyse the contents of a HPG from different perspectives.

The first heuristic, called the fine-grained heuristic, implements an iterative deepening search that incrementally computes the set of strings with probability above a cut-point. A stopping parameter is provided that gives control over the number of mined rules. With this heuristic the analyst is able to compute small subsets of rules containing the best strings from a larger set of strings with probability above the cut-point. The second heuristic, called the inverse fisheye heuristic, makes use of a dynamic cut-point that is very strict when evaluating short strings and becomes more permissible as the strings get longer. This heuristic enables the analyst to induce relatively small sets of longer rules composed of links having high probability on average. Extensive experiments with both synthetic and real data were conducted to assess the heuristics' utility.
1.3. Structure of the Thesis

A set of binary operations on HPGs is defined, which gives the ability to compare the contents of two distinct HPGs. The operations defined are union, sum, intersection, difference and complement. Given two HPGs, each characterising the interaction of a user with the web, their intersection gives the subset of the web that was visited by both users. Such a grammar can be further analysed and the high probability strings identified by using the algorithms we have provided. The grammar resulting from the union or the sum of two grammars enables to identify the high probability strings in a grammar that represents the overall users' navigation history. Finally, the difference between two grammars is useful to discover strings that are among the favourites of one user but are unknown to the other user.

A HPG provides a compact way to incrementally store log data and, in addition, algorithms to extract useful patterns from a HPG are provided. The HPG model can be useful to the web site owner, since understanding the users navigation patterns can provide insight on how to improve the site’s topology. The grammar’s operations can be useful to assess the evolution of the users access patterns over time. When used as a browser plug-in, a HPG can be useful to the individual user, where the user’s log data can be collected via a local proxy. In this scenario the HPG works as a compact representation of the user’s knowledge of the web, making it possible to share that knowledge with others. In fact, by using the HPG operations and the provided algorithms a user is able to compare his navigation preferences with those of his peers.

1.3 Structure of the Thesis

After this introduction, in Chapter 2 we discuss related work in the field of web technologies which aim at enhancing the user navigation experience. We briefly review search engines technology, web visualisation techniques, adaptive hypertext and web navigation assistants. Then, we focus on the field of data mining and we pay particular attention to techniques developed in the field of web usage mining. Finally, we present the relevant concepts on formal language theory, discrete Markov chains and information theory which are used throughout the thesis.

Chapter 3 presents the formal definition of the hypertext probabilistic grammar
1.3. Structure of the Thesis

(HPG) model and the analysis of its basic properties. An algorithm to infer a HPG from a collection of sessions is given as well as an algorithm to find its strings with probability above a cut-point. In addition, the $N$-gram extension to the model is given together with a statistical test to determine the order of the model.

In Chapter 4 we present the results of the extensive experiments conducted in order to assess the performance and effectiveness of the HPG model.

Two novel heuristics are presented in Chapter 5 which extend the model's analysis capabilities. The first heuristic implements an iterative deepening search wherein the set of rules is incrementally augmented by first expanding the trails with high probability. The second heuristic makes use of a dynamic threshold to find longer rules composed of links with above the average probability. Results of the experiments conducted with both real data and synthetic data are presented and analysed.

Chapter 6 presents the formal definitions of several binary operations on hypertext probabilistic grammars, whose aim is to provide the analyst with techniques for comparing the contents of two grammars. The operations defined are: sum, union, difference, intersection, and complement.

In Chapter 7 we give our concluding remarks, present directions for future research, and discuss a range of applications for the hypertext probabilistic grammar model.
Chapter 2

Background

In this chapter we discuss related work and present the relevant background concepts for the thesis. We start by presenting the basic concepts common in research in hypertext and the world-wide-web. We give examples of models proposed for hypertext systems and of different types of navigation assistants, which aim at helping the user overcome the navigation problem (see Section 2.1.2). We then present relevant research in the field of data mining and knowledge discovery and its application to web technologies. We pay particular attention to techniques developed in the field of web usage mining. Finally, we present basic relevant concepts on formal language theory, discrete Markov chains, and information theory which are used throughout the thesis.

2.1 Hypertext and Hypermedia

A hypertext system, (Nielsen, 1990b), is an information management system which has the information stored in a network of nodes and links. The nodes contain the information and links connect related concepts. The user accesses the contents of a hypertext system by means of an interface that provides the ability to browse the information in a non-sequential way. A hypertext system whose nodes also contain audio, video, or images is often called a hypermedia system.

The concept of hypertext was first introduced by Bush in (Bush, 1945) when he proposed the Memex machine. Bush's machine was a device allowing an individual to store his books and publication materials in a microfilm format. A mechanism to access the stored information with speed and flexibility was specified. The essential feature
of Memex was a process which enabled to tie together two pieces of information, in a manner that emulates the associative way the mind works. This feature enabled its owner to construct trails of pieces of information containing related concepts, and store them for future reference. In a following paper, (Bush, 1991) written in 1959, the author revisited his Memex machine and new features were described. The improved Memex was able to identify trails that were followed frequently. In addition, the system could build new trails by searching the stored information for items containing a given set of keywords; the items found were tied together to form a new trail. A collection of Bush’s publications on his Memex machine is available in (Nyce and Kahn, 1991).

The word hypertext was coined by Nelson in (Nelson, 1965) to mean a body of written and pictorial material interconnected in such a way that could not conveniently be presented or represented on paper. He proposed a system (later named Xanadu) which aims at being a repository of everything anybody has ever written. An implementation of the system is being developed whose source code is about to become an open source.

Since this early research the interest in the hypertext concept has grown steadily and several hypertext systems were implemented; a survey of the hypertext history can be found in (Conklin, 1987). The research into hypertext systems provided the foundations for the world-wide-web.

2.1.1 The World-Wide-Web and Search Engines

The world-wide-web (or simply the web), (Berners-Lee et al., 1994), is the largest and most wide spread hypertext system available. What distinguishes the web as a hypertext system is the facility with which anyone can access its contents or place information on it. As such, the web is becoming a powerful tool and growing in importance in the life of many people. As opposed to stand-alone hypertext systems, which in general have focused contents, the web is heterogeneous in: its content and its quality. Moreover, the web doesn’t provide its users with a standard page design across web sites (or even within a site) and it is accessed by a variety of users with different backgrounds and expectations. For example, a site providing information about a programming language can be accessed by an expert, looking for a solution of a very specific problem,
and at the same time by a young student taking the first steps in learning a programming language.

The web is the largest information resource available. According to recent estimates, in February 1999 the web contained about 800 million pages on about 3 million different servers, see (Lawrence and Giles, 1999). The number of web servers was estimated by sampling and testing random IP address numbers and determining the fraction of such tests that successfully located a web server. The estimate of the average number of pages per server was obtained by crawling a sample of the servers identified in the first experiment. A recent update (Inktomi, 2000) verified the existence of over one billion unique indexable pages on the web. In addition, a previous study aimed at evaluating the usefulness of caching techniques on the web, (Douglis et al., 1997), verified that 16.5% of the pages accessed were modified every time they were accessed. This last results were obtained from the analysis of the log traces from two large corporate networks.

A research which studied the structure of the graph defined by the pages and their links on the web is presented in (Broder et al., 2000). The study was performed on a graph inferred from two large Altavista crawls. The analysis of the graph confirmed previous studies which stated the hypothesis that the number of in-links to a page follows a Zipf distribution; see (Wyllys, 1981) for details on the Zipf distribution. The number of out-links from a page exhibits a similar distribution, although there is some evidence that pages with low number of out-links follow a different distribution. In addition, the analysis showed that when the web graph has its links treated as undirected edges 90% of the pages form a single connected component. The analysis of the directed graph revealed the existence of four distinct components. The first component is called the strongly connected component (SCC) and is composed by the set of pages on the web that can be reached by one another. The second component is called the IN component, and is composed by the set of pages from which is possible to reach the SCC but cannot be reached from it. The OUT component contains those pages which can be reached from SCC but do not have a path back to it. Finally, the fourth component is called Tendrils and contains those pages that cannot reach the SCC component.
and cannot be reached from it. The four components present a similar size, measured by the number of pages.

One problem that users face when navigating the web is that of finding relevant information from the enormous pool of web documents. There are essentially two types of service to help a user deal with this problem, which are: directories of resources and search engines. The most well known example of a directory of resources is the Yahoo web site (www.yahoo.com). A directory of resources is basically a classification of web sites into a hierarchy of categories. A user can browse the hierarchy of concepts and, once the required category is found, he is supplied with a set of links to sites containing relevant information. The advantage of this type of service is that the pages are classified into categories under human supervision. Therefore, only pages with high quality and focused contents are included. On the other hand, since human intervention is required these type of sites struggle to cope with the fast pace with which new web sites become available.

A search engine consists essentially of a repository of pages, each indexed by keywords characterising its contents, and an interface to allow users to query the repository, (Brin and Page, 1998). The repository of pages is collected by web crawlers that periodically traverse the web in search for new or modified pages, (Chakrabarti et al., 1999b; Cho et al., 1998). Each page is downloaded and has its contents parsed and analysed in order to identify the keywords that characterise it. When a user submits a query to a search engine references to the relevant pages are selected from the repository and ordered according to a ranking criterion.

There are several techniques used to rank a set of pages returned as the answer to a query. For example, the Google search engine (www.google.com) makes use of an algorithm that measures the relevance of a page by assessing which pages provide a link to that page and how important those pages are, see (Page et al., 1998); the importance of a page in measured by the number of links to it. The value given by this algorithm provides a query independent measure of page relevance. This measure is used in conjunction with a content dependent ranking. Another example of a search engine ranking criterion is the one used by Hotbot (hotbot.lycos.com). Hotbot
monitors users’ queries and the links they subsequently choose to follow. The set of pages containing the submitted keywords are ranked according to their popularity, the pages on top are the more popular among the users, (Hotbot, 1998).

Several studies were performed to assess to what extent the repository indexed by a search engine covers the web, see (Lawrence and Giles, 1999; Lawrence and Giles, 1998; Bharat and Broder, 1998). In the more recent published study (Lawrence and Giles, 1999) the authors conclude that a single search engine indexes between 2.2% and 16.0% of the estimated web size. The combined coverage of all search engines considered in the study is no more then 42% of the estimated web size.

To take advantage of the combined coverage of several search engines (Selberg and Etzioni, 1995) proposed the MetaCrawler, which makes use of the meta search engine concept. The MetaCrawler provides an interface for specifying a query which, once submitted, is posted to several search engines in parallel. The results obtained from the search engines used are merged and presented to the user. More recently, (Lawrence and Giles, 1998) proposed the Inquirus meta-search engine. This tool posts a query to several search engines and each returned page is downloaded and analysed. The quality of each page is re-evaluated by the Inquirus own relevance criterion and the results are shown progressively, so that the user doesn’t have to wait for all the pages to be analysed. Each link to a page is given with the local context around the query search terms displayed in order to enable the user to assess the relevance of the pages by himself.

2.1.2 The Navigation Problem

Search engines and directories of resources are invaluable for the provision of high quality entry points for web navigation. However, no indication is given on what further pages lie ahead to assist the user in choosing which link to follow. In addition, current web browsers provide little support for dealing with the huge amount of choices a user faces while navigating the web. This often frustrates the user and leads to the feeling of getting lost in hyperspace, (Conklin, 1987).

The phenomenon of getting lost in a large multiple display network was first defined in (Elm and Woods, 1985). In their work the authors define getting lost as the
feeling the user experiences when unable to identify how his current position relates to the overall display network. This feeling leads to difficulties in deciding where to go next and it results in a decrease in performance in the completion of the task at hand. In the web the problem of getting lost is more evident due to its huge size and to the variety of both the design and contents of pages. The problem of getting lost in a hypertext system was identified by Nielsen in (Nielsen, 1990a) as one of the most important usability problems in hypertext, and is commonly referred to as the navigation problem.

2.1.3 Hypertext Modelling

In order to formally study the characteristics of hypertext systems the research community has proposed formal models. In (Frisse and Cousins, 1992) the authors stress the need for formal models for hypertext structure, for the semantics of such a system, and for the browsing semantics that govern the hypertext use. Furthermore, in (Halasz, 1998) the author noted the need for mechanisms to query both the structure and information in a hypertext system.

A hypertext system is often referred as a hypertext database or a database of text, (Tompa, 1989). The definition of a hypertext database according to (Levene and Loizou, 1999c) now follows.

**Definition 2.1 (Hypertext database).** A hypertext database \( \mathcal{H} \) is an ordered pair, \( < \mathcal{S}, \mathcal{R} > \), where \( \mathcal{S} \) is a set of pages and \( \mathcal{R} \) is a set of links between pages. The set of pages and links corresponds to a directed graph in which the links define the reachability relation, \( \mathcal{R} \), between states. The collection \( \mathcal{S} = \{ s_1, \ldots, s_N \} \) is a finite set of states, each corresponding to a page in the hypertext system, and \( \mathcal{R} \) is a binary relation in \( \mathcal{S} \) such that whenever \( (s_i, s_j) \in \mathcal{R} \) there is a link in the hypertext system from \( s_i \), the anchor node, to \( s_j \), the destination node.

Botafogo *et al.* modelled a hypertext system as a directed graph, see (Botafogo et al., 1992; Rivlin et al., 1994), and both global and node metrics were devised to reflect the properties of the hypertext system structure. Such metrics aim to help the author cope with the implementation and organisation of a large hypertext system. An example of a global metric is compactness, which takes the value zero if the network is
disconnected and the value one if the network is completely connected. The stratum is another global metric which measures the linearity of the system's structure. Two node metrics are: the depth, which measures the distance of a node to the root; and the imbalance, which measures the variation in the number of out-links per page on the sub-tree of which the node is the root.

In his work, Park (Park, 1998) uses formal language theory to investigate the hypertext system structure and, for example, to recognise regular sub-structures that are embedded in the overall hypertext system structure. In (Stotts et al., 1998) a mechanism for answering queries concerning a document's dynamic properties is proposed. The browsing semantics of a hypertext system is modelled as an automaton (called the link-automaton) that allows one to verify whether or not the link structure of a system will exhibit the desired properties when browsed. The properties of a hypertext system are formalised with a temporal logic notation, which enables one to specify properties. For example, the property that all long browsing sessions must visit page $X$ at some time after page $Y$ was visited.

In (Levene and Loizou, 1999c) the authors propose a hypertext query language to enable the user to specify trail queries. The output of a trail query is the set of trails that are relevant to a given set of keywords. The set of all admissible trails in a hypertext system is modelled as a hypertext finite automaton. The problem of finding a trail that answers a given query is proven to be, in general, NP-complete. Therefore, in a subsequent work the authors enrich the model with probabilities which assign a degree of importance to each trail. The probabilistic model consists in a hypertext probabilistic automaton (see Section 2.5), which reduces the search space by using a cut-point for the trail probability.

### 2.1.4 Hypertext Visualisation Tools

Visualisation techniques aim to help user orientation in a hypertext system. In stand-alone systems methods such as the Furnas’ fisheye views, see (Furnas, 1986), have been proposed. In a fisheye view a measure of interest is assigned to each document. The measure is a function of both the document contents and its distance to the current document being displayed. An overview diagram of the overall document structure is dis-
played in which only documents whose score passes a given threshold are included. The concept was later enhanced in (Tochtermann and Dittrich, 1992) to take into account the particularities of a hypertext system.

More recently, visualisation tools to build overview diagrams of the web have been proposed. Building such diagrams for the web is a challenging task due to its complex underlying structure. In (Mukherjea and Foley, 1995) the authors propose a Navigational View Builder for reducing the information space by, for example, clustering pages by contents and/or by structure, or by filtering the network by contents and/or structure. Using such techniques, an abstract view of the web can be built by displaying documents which are representative of the clusters.

In (Kreutz et al., 1999) a platform independent toolbox of Java applets is described which enables the automatic generation of maps of the web site topology. The toolbox also provides a link preview function to assess the contents of a page without visiting it. Moreover, the toolbox enables the characterisation of several link types and selective compilation; therefore, it is possible to disable or enable a specific type of links before the compilation, in order to setup a hypertext system focused on a given audience.

### 2.1.5 Adaptive Hypertext

Adaptive hypertext, (Brusilovsky, 1998) is a research field focused on improving the user navigation experience. The development of a hypertext system to suit users with different goals and preferences is a challenging task. Adaptive hypertext systems provide a personalised interface for each user. In such systems, a model of the user’s goals, his previous knowledge of the system, and his preferences are built to provide the necessary knowledge to adapt the page design to the individual user.

There are essentially two ways of adapting a hypertext system presentation to a given user. The first consists in adapting the contents of the pages by, for example, showing advanced concepts only to the users which have already visited the pages containing the relevant background concepts. The second type of adaptation is at link level. In this case, a measure of relevance is assigned to each link, according to the user model, and the links are sorted accordingly. Alternatively, links considered more relevant to the user can be annotated and irrelevant links hidden.
2.1. Hypertext and Hypermedia

In (Perkowitz and Etzioni, 1997) the authors challenge the AI community to make use of artificial intelligence techniques in the creation of adaptive web sites. Adaptive web sites are defined to be sites which automatically improve their organisation and presentation by learning from user access patterns. The authors suggest the use of web server log files as suppliers of the information characterising the users access patterns.

One example of a system which focuses on the implementation of an adaptive presentation is proposed in the AVANTI project, see (Fink et al., 1996). In this system a user model is built from information explicitly provided by the user and from observing the user interaction with the system. Based on the user model, the system predicts what the user wants to see next and adapts the displayed information accordingly. In (Bollen and Heylighen, 1997) a system is presented which uses navigation paths to dynamically update the structure of its hypertext network. The system consists of a hypertext network in which each connection between pages is associated with a measure of its strength. The strength of a link is inferred from the navigation patterns and is updated as new paths are traversed. When a page is presented its links are ordered according to the connection strength, the strongest at the top.

2.1.6 Web Navigation Assistants and Agents

Web agents, (Etzioni and Weld, 1995), are software tools which assist users by performing tasks aimed at enhancing the navigation experience. In (Armstrong et al., 1995) the authors present WebWatcher, an agent which helps the user to locate the desired information. The user fills a form stating what information he is looking for and, as the user navigates the web, the agent uses the knowledge learned from previous users to recommend links to be followed; the links thought to be relevant are highlighted. At the end of the navigation the user indicates whether or not the search was successful, and the model is updated accordingly. Another example is Letizia, which is a personal agent that uses the idle processing time available when the user is reading a document to explore links from the current position, see (Lieberman, 1995). The agent observes the user while he navigates and continuously learns a profile of his interests. Upon request Letizia gives recommendations of which links may lead to relevant information. The knowledge about the user is automatically acquired and does not require any user input.
Ngu and Wu propose the SiteHelper as a local agent that acts as the housekeeper of a web server, in order to help a user to locate relevant information within the site, (Ngu and Wu, 1997). The agent makes use of log data to identify the pages viewed by a given user in previous visits to the site. The keywords characterising the contents of such pages are incorporated into the user profile. When that user returns to the site the agent is able, for example, to show the changes that took place in pages that are known to be of interest and also to recommend any new pages.

In (Chen and Sycara, 1998) the authors describe a tool named WebMate; a proxy agent that monitors the user web navigation while building his profile. Each time the user finds an interesting page he points the page to the agent. The agent analyses the contents of the page and classifies it into one of a predefined set of classes. In this way, the user profile is inferred from a set of positive training examples. In off peak hours the agent browses a set of URLs the user wants to have monitored in search for new relevant pages. If the user does not specify URLs to be monitored the agent uses a set of chosen keywords to query popular search engines and assess the relevance of the returned pages.

In (Åberg and Shahmehri, 1999) the authors propose the concept of a human assistant working in an electronic shop. When browsing the shop, a user can ask for help and a human assistant establishes contact with the user by means, for example, of a chat window. By both having access to a database characterising the user’s previous interactions with the shop and being able to monitor his current actions on the site the assistant can provide invaluable help.

Another example of a user assistant is described in (Chalmers et al., 1998; Chalmers, 2000), a recommender system that focuses its predictions from patterns of usage, rather than from contents, and is thus able to handle heterogeneous data. The system builds a local repository containing the URLs of the files viewed in the user browser. Similarly, the system records the activity of the user’s xemacs editor, and therefore the paths stored contain information characterising the use of both tools. Periodically, the recent entries are analysed and the system searches the repository of paths for past occurrences of those entries. The entries which followed those past oc-
2.1. Hypertext and Hypermedia

currences, and which were not recently used, are ranked and suggested to the user. The repository of paths can be individual or shared by several users.

2.1.7 HTML and XML

Presently, most of the pages available on the web are written in HyperText Markup Language (HTML), (Musciano, 1998). The HTML language consists mainly in a small set of tags that identify the different types of components in a page; examples of page components are the title, a table, and a figure. Each component is displayed according to its type definition. Because HTML has a fixed syntax and limited complexity it is easy to learn and it is relatively simple to implement a tool to display documents formatted with it. These characteristics were important in promoting the widespread use of the web. However, as the web has grown the weaknesses of HTML have become evident. In fact, the HTML syntax provides a rigid formatting of a page, ignores syntax violations, doesn’t provide the means to deal with out-dated links and, more importantly, doesn’t allow a detailed definition of the document’s structure.

Recently, an eXtensible Markup Language, (XML), has emerged aiming to overcome the HTML weaknesses, (Mace et al., 1998; Bosak, 1997). The three major aspects in which XML differs from HTML are: being an extensible language, allowing the definition of the document’s logical structure, and allowing the enforcement of structure validation. The adoption of XML will enable the definition and validation of new document types. The document type-definitions can help automated tools to understand the logical structure of documents and provide an enormous potential to improve the accuracy of search engines’ technology. XML also enhances the way information is displayed on the user browser, for example, by providing different ways to view a document’s components without having to download different versions of the document. It will also enable the automatic creation of overview diagrams.

XML can change the way people navigate the web by providing the means to build tools to help overcome, at least in part, the navigation problem. However, XML requires more effort in the development of web pages due to its higher complexity, and that can be a barrier to its widespread use. Moreover, the effort required for the specification of a document type definition will not always be justified. Thus, it is expected
that a significant number of HTML documents will continue to exist on the web.

2.2 Data Mining and Knowledge Discovery

In recent years, the advance in computer technologies and the decrease in their cost has expanded the means available to collect and store data. As an immediate consequence, the amount of information stored has been increasing at a very fast pace.

Traditional data analysis techniques are useful to create informative reports from data and to confirm predefined hypothesis about the data. However, the huge volumes of data being collected create new challenges for such techniques as businesses look for ways to make use of the stored data to gain an edge over competitors. It is reasonable to believe that data collected over an extended period contains hidden knowledge about the business or patterns characterising customer behaviour. For example, the manager of a large supermarket would like to have answers to questions such as who will buy, what will be bought and in what quantities. To answer such questions the data analyst needs new tools and techniques to explore the data in search for answers to questions which were not considered when the data was collected. The answers to such questions are, in general, not explicitly available in the data.

Data mining and knowledge discovery is a research field that focuses on the development of tools which search for hidden patterns in large collections of data. In (Fayyad et al., 1996a) knowledge discovery in databases is defined as the non-trivial process of identifying valid, novel, potentially useful, and ultimately understandable patterns in data. The knowledge discovery process is composed essentially of three different stages: the data preparation, the search for patterns, and the knowledge interpretation and evaluation.

Data preparation is the step responsible for cleaning the data. For example, removing noisy data and deciding what action to take regarding missing data. In addition, some data mining algorithms require the data to be pre-processed into an amenable format. Search for patterns is the data mining step of the knowledge discovery process. At this stage algorithms are run on the data and patterns are extracted from it. Finally, the step of knowledge interpretation and evaluation is responsible for assessing the success
2.2. Data Mining and Knowledge Discovery

of the overall process. The careful analysis of the results given by the data mining step may suggest the necessity for better data cleaning or for different settings of the data mining algorithm parameters. Also, not all patterns extracted from the data are useful and this step aims at identifying the patterns which represent new and potentially useful knowledge. Overviews of the overall knowledge discovery process are available in (Frawley et al., 1991; Fayyad et al., 1996a; Chen and Yu, 1996).

2.2.1 Data Mining Techniques

We will now present a more detailed characterisation of the data mining stage in the knowledge discovery process. The two main goals of data mining are prediction and description. By prediction we understand the use of attributes whose value is known to predict the value of an unknown attribute. By description we understand the identification of patterns to summarise the contents of a large data set.

Data mining techniques can be divided into five classes of methods (Fayyad, 1998) that are briefly described below. The first class is Classification. Classification methods aim at finding common properties among a set of objects and mapping those objects into a set of predefined classes. A set of training examples is used to develop a description of each class, and the resulting class descriptions are used to classify future data. An example is the classification of the clients of an insurance company according to the probability of submitting a claim. Clients classified in the higher risk classes have to pay higher premiums.

Clustering is the second class of data mining methods. A clustering technique aims at identifying a finite set of categories to describe a data set. The difference to classification is that in clustering it is not known in advance which classes will be used. A clustering example is the search for groups of clients (i.e., clusters of clients) with similar spending habits in a supermarket. The third class is the class of Summarisation techniques. These techniques aim at inferring a compact description of a large data set. A common example is the application of the association rules technique to a big database of sales transactions. The inferred association rules show which items have high probability of being bought together in a transaction.

Dependency modelling is the fourth class and consists of methods which aim at
finding a model describing dependencies between variables. An example would be the methods focused on finding functional dependencies between attributes of a relational database. The fifth and last class is Deviation detection. This class contains techniques aimed at detecting unusual changes in the data relatively to the expected values. Such techniques are useful, for example, in fraud detection, where the inconsistent use of credit cards can identify situations where a card is stolen. The inconsistent use of a credit card could be noted if there were transactions performed in different geographic locations within a given time window.

2.2.2 Mining Association Rules and Sequential Patterns

The data mining techniques most relevant to our work are those for the discovery of association rules in a database of transactions, (Agrawal et al., 1993; Park et al., 1995; Agrawal et al., 1996). An association rule is a statement of the form $A \Rightarrow B$, where $A$ and $B$ are sets of attributes. A rule that holds in a collection of transactions means that transactions which contain $A$ have a certain probability of also containing $B$.

A typical application area for the association rules techniques would be a database of supermarket sales transactions. In this type of scenario a rule is a statement of the type “40% of the supermarket customers that buy bread and ham also buy lettuce”. Each transaction is characterised by a customer ID and the items the user bought in a visit to the supermarket. The set of transactions has to be converted into a table of binary attributes, such that each row corresponds to a transaction and each column to a product. If a transaction contains a 1 in a given column it means that the corresponding product was bought in that visit to the supermarket. If the attribute has the value 0 it means that the product wasn’t bought in that same visit. Moreover, an association rule $A \Rightarrow B$ is characterised by both its support and its confidence. The support of a rule is given by the percentage of all transactions that contain all the attributes in $A$ and $B$. The confidence of a rule is given by the percentage of the transactions containing $A$ that also contain $B$.

The process of finding all the association rules with confidence and support above the respective thresholds is a two stage process. The first stage consists in finding all sets of items with support above the support threshold; these sets are called large item-
sets. The second step consists in computing for each large itemset the confidence for all its expressions with the form of a rule; the expressions whose confidence is above the confidence threshold are the rules. The second step of the process is trivial, therefore, the problem of mining association rules can be viewed as the problem of finding all the itemsets with support above a given threshold. The most well known algorithm to perform such task is the Apriori algorithm, which is described in (Agrawal et al., 1996).

Another type of useful patterns in a database of sales transactions are the sequential patterns, (Agrawal and Srikant, 1995; Srikant and Agrawal, 1996). While the association rules technique searches for items that were bought together in a transaction, sequential patterns search for items that were bought by the same customer but not necessarily in the same transaction. An example of a sequential pattern could be "A user who has rented the video of Return of Jedi in a subsequent visit to the store is likely to rent the video of Star Wars". (Note that the rents don’t need to occur in consecutive visits.)

A sequence is defined as an ordered list of itemsets. For each customer the transactions are ordered by transaction time and viewed as a sequence, the customer-sequence. A customer supports a sequence if that sequence is contained in the customer-sequence. The support of a sequence is given by the percentage of customers' sequences that contain that sequence. The problem of finding sequential patterns consists of finding all sequences with support above a given threshold. An efficient algorithm used to search for sequential patterns is given in (Srikant and Agrawal, 1996).

2.3 Web Data Mining

Web data mining has been defined as the application of data mining techniques to web data, (Etzioni, 1996; Mena, 1999). Research in web data mining is guided by two long-term goals: the development of techniques to aid the design of good web sites and the development of techniques to assist the individual user when navigating the web.

On the one hand, there is a need for techniques to aid business decision support on electronic commerce. An example in this context is the necessity of understanding the user behaviour in order to place focused advertising on the web. A second example
2.3. **Web Data Mining**

is the design of web sites which achieve business objectives, for example, a site that leads a user to pages with products having higher profit margins. Moreover, the analysis of network traffic is important in the determination of the hardware requirements for achieving the desired quality of service.

On the other hand, the research community is also driven by the goal of providing the individual user with tools and services aimed at enhancing the navigation experience. A user will benefit by having access to search engines with improved search technologies, to web sites with personalised interfaces, and to personal tools that help him deal with the vast amount of information and navigation options on the web. These are areas of research that, if successful, can help the user overcome the navigation problem and consequently have a better web experience.

In web data mining there are currently three main research directions:

(i) mining for information,
(ii) mining the web link structure, and
(iii) mining for user navigation patterns.

Mining for information focuses on the development of techniques for assisting a user in finding documents that meet a certain criterion, see for example (Craven et al., 1998). Mining the link structure aims at developing techniques to take advantage of the collective judgement of web page quality which is available in the form of hyperlinks. In fact, links can be viewed as a mechanism of implicit endorsement, see (Chakrabarti et al., 1999a). Finally, mining for user navigation patterns focuses on techniques which study the user behaviour when navigating the web, see (Borges and Levene, 2000a).

Note that some actual search engines use techniques from the three categories described above; however, a search engine has a more general approach in which every document crawled is indexed and ranked. Therefore, search engines are usually not classified as tools which performs web content mining. Surveys of the research being carried out in web data mining are available in (Cooley et al., 1997; Garofalakis et al., 1999; Chakrabarti, 2000).
2.3.1 Web Content Mining

Web content mining focuses on techniques for searching the web for documents whose content meets a certain criterion. The documents found are used to build a local knowledge base. One approach to this problem is the definition of query languages focused on resource discovery on the web, see for example (Mendelzon et al., 1996; Zaïane and Han, 2000). In (Mendelzon et al., 1996) the authors model the pages and links on the web as a relational database with two tables, the first table characterises the pages and the second the links between pages. An SQL-like language is defined for the context. Links are characterised according to their locality and the cost of a query is computed as a function of the cost of the links necessary to traverse in order to answer the query. Zaïane and Han make use of a multi-layered database model to transform the unstructured data on the web into a form amenable to database technology, (Zaïane and Han, 2000). Specialised tools are used to extract information from web pages in order to identify relevant documents. Abstract characterisations of the relevant documents are stored in the local database. A query language is provided which enables the querying of high level characterisations in the local database and, if more detail is needed, the actual web resources are queried via a search engine.

The approach taken by (Craven et al., 1998) consists in developing techniques to automatically create and maintain a computer-understandable knowledge base whose contents mirrors that of the web. A set of document classes is specified and training examples are provided. Subsequently, a crawler traverses the web, or a web site, in search of documents that fit into the predefined classes. The documents found are stored in the database and traditional query techniques can be used to query the repository. In (Brin, 1998) a system for extracting a relation from the web is proposed. An example of such relation is a list of all the books referenced on the web. The system contains a function which converts a set of training examples into a list of patterns. Those patterns are then used to search the web for similar documents. Another application of this tool could be to build a relation with the name and address of restaurants referenced on the web.
2.3. Web Structure Mining

Web structure mining is a research field focused on using the analysis of the link structure of the web in order to identify relevant documents. The underlying idea is to see a hyperlink as a form of endorsement and therefore, take advantage of the collective judgement of a page in the assessment of its quality.

In (Spertus, 1997) the author proposed the use of the information contained in hypertext links to find relevant pages. For example, a page that is one link away from an index page related to computer science has a considerable probability of being relevant to the subject. For pages two or three links away the probability is lower but it should be higher than a random page on the web. In addition, the author proposes the use of the hierarchy information contained in the URL to categorise links as upward, downward, crosswise or outward. For example, pages in Yahoo contain mainly downward and outward links. If links are crosswise or downward they will probably lead to pages whose contents is a specialisation of the original page, outward links should lead to pages on the same subject. The author argues that the use of this type of information, when combined with the results returned by a search engine, can be useful in finding individual homepages or new locations of moved pages.

The Google search engine makes use of the web link structure in the process of determining the relevance of a page, see (Brin and Page, 1998; Page et al., 1998). The pages matching a set of keywords are ranked according to a measure of the human interest and attention devoted to each page. The human interest in a page is measured by the quantity of links pointing to it; note that this measure is query independent. To determine the page popularity the authors make use of a random surfer that browses the web at random. When browsing the web at random the number of times a page is visited should be proportional to the number of pages that have links to it; pages visited more often are more popular. The Google search engine achieves good results because while the keyword similarity analysis ensures high precision the use of a popularity measure ensures high quality of the pages returned.

In the Clever project a different approach is adopted in order to make use of the information available in the link structure. The goal of this approach is the identifi-
2.3. Web Data Mining

An algorithm called HITS is proposed which computes a list of hubs and authoritative pages. A hub is defined as a page providing links to a collection of sites on a common topic. An authority is a page that is pointed to by many good hubs. The algorithm starts by building a sample of the web that should be rich in authoritative pages for a topic specified by a set of keywords. This step is done by querying a search engine and the set of pages returned is called the root set. The root set is then expanded in order to include all pages that are linked to by pages in the root set, and all pages that link to pages in the root set. This step assumes that the authoritative pages on the subject should be in the root set, be linked by a page in the root set, or link to a page in the root set. Each page in the extended root set is assigned a non-negative hub-weight and a non-negative authority-weight. An iterative process is then carried out in which pages pointed to by many good hub pages have the authority-weight increased, and pages that point to many good authorities have the hub-weight increased. The process continues until the weights converge, and the pages with higher authority-weight are returned as the authoritative pages in the topic.

2.3.3 Web Usage Mining

Web usage mining is a research field that focuses on the development of techniques and tools to study users' web navigation behaviour. Understanding the visitors' navigation preferences is an essential step in the study of the quality of an electronic commerce site. In fact, understanding the most likely access patterns of users allows the service provider to customise and adapt the site's interface for the individual user (Perkowitz and Etzioni, 1997), and to improve the site's static structure within the underlying hypertext system, (Rosenfeld and Morville, 1998).

When web users interact with a site, data recording their behaviour is stored in web server logs. These log files may contain invaluable information characterising the users' experience in the site. In addition, since in a medium size site log files amount to several megabytes a day, there is a necessity of techniques and tools to help take advantage of their content.
2.3.3.1 Log Files and Data Preparation Issues

A log file is a text file in which every page request made to the web server is recorded. For each request the corresponding log file contains the following information:

1. IP address of the computer making the request;
2. userID, (this field is not used in most cases);
3. date and time of the request;
4. a status field indicating if the request was successful;
5. size of the file transferred;
6. referring URL, that is, the URL of the page which contains the link that generated the request;
7. name and version of the browser being used.

This information can be used to reconstruct the user navigation sessions within the site from which the log data originates. In an ideal scenario each user is allocated a unique IP address whenever he accesses a given web site. Moreover, it is expected that a user visits the site more than once, each time possibly with a different goal in mind. Therefore, a user session is usually defined as a sequence of requests from the same IP address such that no two consecutive requests are separated by more than \( X \) minutes, where \( X \) is a given parameter. In (Catledge and Pitkow, 1995) the authors report an experiment conducted with a web browser that was modified in order to record, among other things, the time interval between user actions on the browser's interface. One interesting result of the study revealed that 25.5 minutes corresponded to 1.5 standard deviations of the average time between user actions, meaning that the probability of a user staying more than 25.5 minutes without placing any page request is very low. As a result of the study many authors have adopted the value of 30 minutes for the time limit between requests within a session, i.e. \( X = 30 \) minutes.

In the general case, however, the task of identifying the sequence of requests made by a given user when visiting a web site is not trivial. In fact, not all requests are recorded in the log file and a user can be allocated more than one IP address during a single session. Therefore, the data preparation stage of web usage mining demands
considerable effort and several authors have been studying this problem, (Pirolli et al., 1996; Pitkow, 1997; Cooley et al., 1999). We will now briefly discuss some aspects of log data preparation.

The first task when using log data is data cleaning, which consists in identifying the useful log entries. The file transfer protocol used on the web requires the establishment of a separate connection for each file requested. Therefore, a HTML page containing images, sound, or video, will originate a request for each file it contains. In most cases the log entry of the HTML file is the only entry corresponding to a file explicitly requested by the user, and all other log entries can be ignored.

Another relevant aspect to take into account when using log data is the widespread use of cache and proxy servers on the web. As a result, not all page requests made to a server are recorded in the log file. In fact, if the browser finds in its cache a copy of a document being requested by the user, the request is not made to the server and the stored copy of the document is displayed. Therefore, although the page is viewed by the user the request is not recorded in the server log file. A similar thing can occur at the proxy level. In addition, the use of proxies raises difficulties in the identification of the requests made by a given computer. A proxy server can be configured in such way that, when a copy of a requested page is not available in the local memory, the page is requested by the proxy to the content provider on behalf of the user. In such cases, the IP address recorded in the log file corresponds to the proxy and not to the user. Note that more than one user can be using the same proxy to browse the same site at the same time. Moreover, it is possible to have different users sharing the same IP address, and some internet service providers dynamically allocate different IP addresses to a single user during a single connection.

There are techniques which help to overcome the problems described above. For example, the use of the referrer field in conjunction with the web site's topology allows one to identify requests that are missing in a session (due to cache use). If two session are disjoint it may allow one to identify two simultaneous sessions with a common IP address. Moreover, the use of the log entry which identifies the browser is useful to distinguish users with different browsers.
In addition, the use of cookies provides a way to track an individual user within a site. If cookies are enabled, when a document is requested by a new user the response includes an unique user identifier which the browser stores in the user's hard disk. All subsequent requests made by the browser to that same site will include the cookie information and therefore, allow the service provider to recognise the user. However, the use of cookies is only possible with the user's consent and its use has raised privacy concerns, which are discussed in Section 2.3.4. Note that cookies only identify browsers and not individual users.

Another technique which is currently being used by some online shops consists of appending a unique identifier to the URL of the first page requested within the site. In addition, all URLs in the page displayed are modified in order to include the same unique identification. This way, every requested URL will uniquely identify its originator, enabling the content provider to keep track of a user within the site. If the user proceeds to the checkout and provides his identification in the paying process, he will enable the service provider to relate the current session with previous sessions of the same user. The user navigation sessions can be accurately reconstructed with this technique since the use of modified URLs can, in general, delude both caches and proxies.

Finally, in a scenario where a browser is setup to record a personal log file, by means of a local proxy, the data characterising the web navigation of a given user can be accurately collected if the browser's cache is disabled.

In conclusion, we note that the accuracy of user sessions inferred from server log files can be severely affected by the use of cache, proxy servers, and IP address sharing. Therefore, techniques such as cookies and URL modification are essential for the identification of requests from the same user in order to enable the accurate reconstruction of the user navigation sessions from log data.

2.3.3.2 Web Usage Mining Techniques

Currently, several commercial log analysis tools are available (Stout, 1997; Mena, 1999). However, these tools have limited analysis capabilities producing only results such as summary statistics and frequency counts of page visits. In the meantime the research community has been studying techniques to take full advantage of the informa-
2.3. Web Data Mining

To that effect we classify the research being done in the field into the following classes:
prediction, customisation, and visualisation; adapting standard data mining techniques;
local data mining architectures; novel data mining models.

Prediction, Customisation, and Visualisation

Some authors have recently proposed the use of a first order Markov model for
predicting user requests on the web. In (Bestavros, 1995) a first order Markov model is
proposed to implement a prefetching service aimed at reducing server load. The model
is built from past usage information and the transition probabilities between pages are
proportional to the number of times both pages were accessed in a predefined time win­
don. We note that the use of a time window results in having transitions with probability
greater than zero between pages that were never accessed consecutively. The results of
the conducted experiments show that the method is effective in reducing both the server
load and the service time. A similar method is proposed in (Padmanabhan and Mogul,
1996) wherein a dependency graph is inferred and dynamically updated as the server
receives requests. There is a node for every requested page, and an arc between two
nodes exists if the target node was requested within \( x \) accesses after the source node;
the weight of an arc is proportional to the number of such requests. The simulations
performed with log data show that a reduction in the retrieval latency can be achieved.
We also note that this method does not guarantee transitions only between consecutive
requests.

In (Pirolli and Pitkow, 1999) the authors present a study of the quality of a \( k \)th
order Markov approximation, \( k \geq 1 \), as a model for predicting user surfing patterns.
Several algorithms for path reconstruction were tested and the results suggest that a
model inferred from longer paths is more accurate. However, the largest reduction in
uncertainty is achieved when moving from a zero-order path reconstruction model to
a first-order path reconstruction model. The model accuracy is measured using the in­
formation theoretic measure of conditional entropy, see Section 2.6. They use the con­
ditional entropy of two models as a measure of the decrease in uncertainty when mov­
ing from one model to another. In addition, they show that the model probabilities are
more stable over time for models with lower order. In a subsequent work, (Pitkow and Pirolli, 1999), the authors propose a method to reduce the size of their Markov model while maintaining its prediction accuracy. An algorithm is provided to mine the longest repeating sub-sequences from log data. Assuming that the longest sequences contain more predictive power, the model is inferred only from such sequences, as opposed to from the complete log files. The performed tests show that it is possible to reduce the model size while maintaining the accuracy in prediction. We note that this work is focused on assessing the predictive power of the model as opposed to providing the analyst with techniques to identify relevant patterns. Moreover, we think that this model overlooks the important characteristic of being incremental. In fact, a sub-sequence which was not frequent may become frequent when more log data becomes available and, in order to take that into account, the model has to be rebuilt from scratch whenever more data becomes available.

In (Perkowitz and Etzioni, 1997) the authors challenged the AI community to use the log data to create adaptive web sites and in (Perkowitz and Etzioni, 1998; Perkowitz and Etzioni, 2000) they present a technique to automatically create index pages from the log data, i.e., pages containing collections of links which the user navigation behaviour suggests are related. A graph is created in which a node corresponds to a page and a link gives the probability of the two connected pages occur in the same session. An algorithm is proposed which makes use of the graph to search for a small number of clusters of pages that tend to appear together in a session. The clusters are the candidate index pages which can be used to customise a web site to suit the users needs.

In (Cadez et al., 2000) the authors propose a methodology for the visualisation of navigation patterns. A model based clustering approach is used in which users presenting similar navigation patterns are grouped into the same cluster. The behaviour of the users within each cluster is represented by a Markov model. In a recent work (Sarukkai, 2000) the authors proposed a system which is used to demonstrate the utility of Markov chain models in link prediction and path analysis on the web. Experimental results are reported which show that a Markov model can be useful both in the prediction of http requests and in the prediction of the next link to be requested.
Adapting Standard Data Mining Techniques

The use of data mining techniques to analyse log data was first proposed by (Chen et al., 1998) and (Yan et al., 1996). The proposed approach consists in mapping the log data into relational tables in a way that enables to analyse the data with adapted versions of standard data mining techniques.

In (Chen et al., 1998; Wu et al., 1998) the log data is converted into a tree, from which is inferred a set of maximal forward references. The maximal forward references are then processed by existing association rules techniques. Two algorithms are given to mine for the rules, which in this context consist of large itemsets (see Section 2.2.2) with the additional restriction that references must be consecutive in a transaction. In our opinion the procedure used to build the maximal forward references tends to over evaluate the links close to the root of the resulting traversal tree. For example, a link from the root to one of its child nodes will appear in all forward references passing through that child, enlarging its support, while this link may have only been traversed once. In (Yan et al., 1996) a method is proposed to classify web site visitors according to their access patterns. Each user session is stored in a vector that contains the number of visits to each page, and an algorithm is given to find clusters of similar vectors. The clusters obtained with this method do not take into account the order in which the page visits took place.

In (Zaïane et al., 1998) a tool is proposed, the WebLogMiner, that integrates data warehousing and data mining techniques for the analyses of log records. The data collected from web server logs is cleaned and a data cube is constructed to represent it. The multi-dimensional cube provides flexibility to manipulate the data in order to view it from different perspectives and compute measures across many dimensions. Standard data mining techniques can be used to analyse the data. According to the authors the construction of the data cube is very demanding and time consuming to build.

Fuzzy techniques are proposed in (Nasraoui et al., 1999) to cluster user navigation sessions into session profiles. The use of fuzzy membership techniques is adopted in order to allow some access patterns to belong to more than one class. Moreover, asso-
2.3. Web Data Mining

Associations between web pages on a given site can be inferred from the resulting profiles.

Global Data Mining Architectures

Some authors have proposed global architectures to handle the web usage mining process. Cooley et al. propose a web site information filter, named WebSIFT, that establishes a framework for web usage mining, (Cooley et al., 2000). The WebSIFT performs the mining in three distinct tasks. The first is a pre-processing stage in which user sessions are inferred from log data. The second stage searches for patterns in the data by making use of standard data mining techniques, such as association rules or mining for sequential patterns. In the third stage an information filter based on domain knowledge and the web site structure is applied to the mined patterns in search for the interesting patterns. Links between pages and the similarity between contents of pages provide evidence that pages are related. This information is used to identify interesting patterns, for example, itemsets that contain pages not directly connected are declared interesting. In (Mobasher et al., 1999) the authors propose to group the itemsets obtained by the mining stage in clusters of URL references. These clusters are aimed at real time web page personalisation. A hypergraph is inferred from the mined itemsets where the nodes correspond to pages and the hyperedges connect pages in a itemset. The weight of a hyperedge is given by the confidence of the rules involved. The graph is subsequently partitioned into clusters and an occurring user session is matched against such clusters. For each URL in the matching clusters a recommendation score is computed and the recommendation set is composed by all the URL whose recommendation score is above a specified threshold.

In (Büchner et al., 1999) an approach, in the form of a process, is proposed to find marketing intelligence from internet data. An $n$-dimensional web log data cube is created to store the collected data. Domain knowledge is incorporated into the data cube in order to reduce the pattern search space. In (Baumgarten et al., 2000) an algorithm is proposed to extract navigation patterns from the data cube. The patterns conform to pre-specified navigation templates whose use enables the analyst to express his knowledge about the field and to guide the mining process. This model does not store the log data in a compact form, and that can be a major drawback when handling very large
daily log files.

In (Masseglia et al., 1999b; Masseglia et al., 1999a) an integrated tool for mining access patterns and association rules from log files is proposed. The techniques implemented pay particular attention to the handling of time constraints, such as the minimum and maximum time gap between adjacent requests in a pattern. The system provides a real time generator of dynamic links which is aimed at automatically modifying the hypertext organisation when a user navigation matches a previously mined rule.

**Novel Data Mining Models**

Finally, several authors have adopted the approach of developing new techniques to be invoked directly on the log data in a way that directly captures the semantic of web navigation. Such techniques are aimed at identifying user web navigation patterns.

The authors of (Schechter et al., 1998) propose to use log data to predict the next URL to be requested, so the server can generate in advance web pages with dynamic content. A suffix tree containing the user paths is generated from the log data, and an algorithm is proposed to predict the next request given the tree and the current user session. This method is very accurate in the way it represents the navigation sessions inferred from log data, however, it is also very demanding in terms of storage space. In fact, every sub-path of a navigation session is stored individually in the tree. The authors propose the use of a threshold on the path length or on the path frequency to reduce the number of potential paths. However, we note that with such threshold the model would not be incremental, in the sense that paths which were previously not frequent enough would not contribute for the frequency assessment of new paths.

In (Spiliopoulou and Faulstich, 1998) the authors propose a log data mining system for the discovery of patterns with predefined characteristics. The system is composed of an aggregation module and a data mining module. The aggregation module infers a tree structure from the data in which the mining is performed by a human expert using a mining query language. However, no performance studies were reported and the use of a query language to find patterns with predefined characteristics may prevent the analyst finding unexpected patterns. In (Spiliopoulou et al., 2000) the system is extended with a methodology for comparing the navigation patterns of customers with those of non-
customers; non-customers are defined as the visitors who did not purchase a product in the site. From the pattern comparison a set of rules is extracted which can help the web site designer to improve the site topology in order to turn more visitors into customers.

In (Pei et al., 2000) the authors propose a novel data structure and a new algorithm to mine web access patterns from log data. The web access sequences are stored in a tree like data structure, the WAP-tree, which is more compact than the initial access sequence database. An efficient algorithm is provided to mine all the frequent patterns in the tree. However, the tree inferred from the data is not incremental since it includes only the frequent access sequences. Moreover, although the algorithm is efficient the performance analysis should take into account the time needed to build the tree, since the input data for the tree construction is in the form used by the algorithm against which the proposed method is compared.

2.3.3.3 This Thesis Approach

The work presented in this thesis is part of an ongoing research with the long term goal of specifying a set of techniques to identify relevant web trails, (Levene and Loizou, 1999d; Zin and Levene, 1999; Borges and Levene, 2000a). In (Levene and Loizou, 1999c) the authors propose a query language aimed at finding the set of hypertext trails relevant to a given set of keywords. Since the approach referred above is proved to be NP-complete, in (Levene and Loizou, 1999d) the model is enriched with probabilities in order to assign a degree of importance to each trail and therefore reduce the search space. The problem is modelled as a hypertext probabilistic automaton whose formalism is given in Section 2.5.

In this work we adapt and extend the hypertext probabilistic automaton model to the problem of studying user web navigation. We propose the alternative formalism of a hypertext probabilistic grammar (HPG) while taking into account the specificity of the problem of mining user web navigation patterns from log data. In a previous work, (Borges and Levene, 1998), we proposed to model log data as a directed graph with the arcs' weights interpreted as probabilities that reflect the user interaction with the site, and we generalised the association rule concept. The model was enhanced in (Borges and Levene, 2000a) by making use of the theory of probabilistic regular grammars and
is presented in its full detail in Chapter 3. Moreover, in (Borges and Levene, 2000c) and (Borges and Levene, 2000b) we have proposed two new heuristics to find patterns on data which enhance the model analysis capabilities; the heuristics are presented in full detail in Chapter 5.

A HPG is a Markov model which assumes that the probability of a link being chosen depends more on the contents of the page being viewed than on all the previous history of the session. Note that this assumption can be weighted by making use of the N-gram concept, see Section 3.3, or dynamic Markov chains techniques, see (Levene and Loizou, 1999a). There are situations in which a Markov assumption is realistic, such as, for example, an online newspaper where a user chooses which article to read in the sports section independently of the contents of the front page. However, there are also cases in which such assumption is not very realistic, such as, for example, an online tutorial providing a sequence of pages explaining how to perform a given task. In such cases, when viewing a page a user can be given with several navigation options, however, the link chosen to browse next depends on the task being studied and, therefore, on the pages seen previously. As such, a user tends to follow often the same long trail. Although there are some cases in which a Markov model is not realistic, we believe that a Markov model enhanced by the N-gram concept will be useful in the study of user web navigation behaviour and, therefore, we adopt one such model as our technique to study user web navigation.

The HPG model can be seen as a usage mining model within the class of 'Novel Data Mining Models', see Section 2.3.3.2. By proposing a new model we aim to provide the log data analyst with a model that directly captures the semantics of user web navigation. It is our belief that although adapting conventional data mining techniques to the web usage mining problem enables to benefit from all the research done in the field, that approach has the drawback of not being focused on the particularities of the problem. By basing our model on the theory of regular grammars and Markov chains we aim to provide a sound theoretical foundation in order for it to be stable and extensible to tackle new questions that may arise in the future. The HPG also provides a compact and incremental way of storing log data, and efficient algorithms to mine for
rules are supplied.

The HPG is comparable to the models proposed by (Spiliopoulou and Faulstich, 1998) and (Schechter et al., 1998) which were developed in parallel with our work. The prefix tree proposed in (Schechter et al., 1998) represents exactly the user sessions inferred from log data. However, the authors state the storage space required to store such a tree as a significant problem, thus, justifying the need for a frequency threshold for the sessions included in the tree. With the use of such threshold the tree is not incremental and, therefore, overlooks a very important characteristic in a web usage mining model.

The model proposed by (Spiliopoulou and Faulstich, 1998) adopts a different philosophy by providing a mining language which enables the analyst to specify structural and statistical criteria for the patterns to be mined for. However, the performance of the method is identified as a problem by the authors. In fact, they refer that many technical improvements are necessary since the search for navigation sequences is much more expensive than the conventional association rules technique. Also, the use of a query language will probably lead to the necessity of using a specialist.

The objective of web usage mining is the development of techniques that make use of the information available in log files to the analysis of user web navigation. A better understanding of the way users navigate the web is important to gain some insight on how to improve the quality of service in a web site. For a small web site the intuition of the web designer along with some usage statistics is probably enough to achieve a good quality. However, on large web sites more advanced analysis techniques are essential to help the designer verify if the web site is achieving its purposes. A model such as the HPG, or those proposed by other authors, can help the task of improving web design by providing a way to identify patterns of usage.

Web usage mining researchers, in general, do not involve end users in the evaluation of the proposed techniques. However, evaluation involving end users is an important factor and the analysis of user navigation behaviour could be enhanced if personal information about the users was made available. Some examples of useful information are the demographic characteristics of the users, their personal characteristics (age, sex), and their personal habits when using the browser (use of back button or
bookmarks). Note that the use of such information raises some privacy concerns. One experiment which could be conducted to perform this type of evaluation consists in asking a group of users to perform a given task in a web site, for example, to fill a shopping basket with a set of products available in the site. The information characterising the user navigation while performing the task could then be collected and analysed in order to study the user performance when executing the task. Measures of performance could be, for example: the percentage of the required products which were found, the time spent in the completion of the task, the number of pages viewed, and the number of unrelated pages downloaded. After analysing the user navigation data the web designer could improve the design of the web site and have a different group of users repeating the experiment in order to assess if the quality of service improved.

A different type of experiment could be conducted in order to assess a model’s usefulness as a prediction tool. Such model could be used by the browser to prefetch web pages that were expected to be requested (or pushed if considered on the server end) or to generate in advance pages with dynamic contents. In this case the measures of the model’s performance could be the percentage of requested documents which were pushed or generated in advance, the increase in data transfer due to push, and the variation in the time it takes for a requested page to be displayed.

### 2.3.4 Privacy Issues in Web Data Mining

Traditional data mining techniques have from its early stages raised concerns about the privacy of individuals, (Brankovic and Estivill-Castro, 1999; O’Leary, 1991). Reservations about the use of personal data for purposes other than those for which the data was collected gain particular relevance when data mining techniques are used. In fact, even when the data does not explicitly contain sensitive information, the identification of general patterns about an individual can be sufficient for guessing confidential aspects about an individual.

With the advent of web data mining techniques similar concerns are emerging about it, especially in what concerns with web usage mining, (Broder, 2000). One aspect which has been raising considerable concern among the community of web users is the cookies technology. **Cookies** is a mechanism that enables the web content provider
to recognise what requests originate from a given computer and, as such, track users' steps within the site. Cookies are useful, for example, in the implementation of individual shopping baskets in a commercial site. Web usage mining researchers make use of cookies to reconstruct individual user sessions from log data for posterior analysis.

Although a user may feel threatened by having data mining techniques analysing his browsing behaviour, in general, such techniques are focused on identifying common behaviour in groups of users. Moreover, even a successfully reconstructed user navigation session only identifies the user IP address and, unless the user chooses to provide some form of identification, it is a long way to the identification of an individual. On the other hand, some websites require users to register in order to gain full access to the site's contents. In such cases, it is possible to match a set of user navigation sessions with a given individual. There is a trade-off between what the user loses in privacy and what he may gain by having his data analysed. If the web content provider understands the preferences of his visitors he is able to adapt the site to the visitors' preferences.

There is currently a programme named TRUSTe dedicated to build consumers' trust on the web, (Benassi, 1999). In fact, in order for a user to agree to provide some form of personal identification and to give permission to the analysis of his data, the user needs to believe that his data won't be misused. The TRUSTe programme establishes a set of privacy practice guidelines regarding the use of user data. Web sites which decide to comply with the guidelines can display the TRUSTe trust-mark. Such sites have the compliance with the guidelines periodically reviewed. By knowing the web site policy regarding the use of personal data the user can make an informed decision about whether or not to provide his personal details.

2.4 Probabilistic Languages

We now give the necessary background concerning the theory of formal languages; refer to (Salomaa, 1973; Harrison, 1978; Hopcroft and Ullman, 1979; Martin, 1997) for more detail.
2.4. Context-Free and Regular Grammars

An alphabet $\Sigma = \{a_1, \ldots, a_n\}$ is a finite nonempty set of symbols and a string, $w$, is a finite-length ordered sequence, possibly empty, of symbols. The length of a string is defined as the number of symbols composing it and is denoted by $|w|$. As usual, $\Sigma^*$ denotes the set of all finite length strings over $\Sigma$, including the empty string $\epsilon$, and $\Sigma^+$ denotes the set $\Sigma^* - \{\epsilon\}$. A language $L$ over $\Sigma$ is any subset of $\Sigma^*$. A grammar, $G$, is a generative device capable of generating all the strings in a language and nothing more.

**Definition 2.2 (Context-free grammar).** A context-free grammar is a four-tuple $G = \langle V, \Sigma, S, P \rangle$, where:

1. $V = \{A_1, A_2, \ldots, A_n\}$ is a finite alphabet of nonterminal symbols;
2. $\Sigma = \{a_1, a_2, \ldots, a_m\}$ is a finite alphabet of terminal symbols such that $V \cap \Sigma = \emptyset$;
3. $S = A_i \in V$ is a unique start symbol;
4. $P$ is a finite set of production rules of type $X \rightarrow w$, where $X \in V$ and $w \in (V \cup \Sigma)^*$.

A string, $w_2$, is obtained by a one-step derivation of string $w_1$, $d_{w_1} : w_1 \Rightarrow w_2$, if when a nonterminal symbol in $w_1$ is replaced by the right-hand side of a production rule in $P$ the result is the string $w_2$. A derivation, $d_{w_n} : w_1 \Rightarrow^* w_n$, is a finite sequence of one-step derivations that derives $w_n$ from $w_1$. A final string is a string composed only of terminal symbols. A string containing nonterminal symbols is called a sentential form.

**Definition 2.3 (Context-free language).** A context-free language is a language that can be generated by a context-free grammar. A grammar’s language is composed by all the final strings derived from the unique start symbol, $L(G) = \{w \in \Sigma^* | S \Rightarrow^* w\}$.

A regular grammar is a specialisation of a context-free grammar.

**Definition 2.4 (Regular grammar).** A grammar is regular if all the productions $P_i \in P$ are of the form $A_i \rightarrow a_k$ or $A_i \rightarrow a_k A_j$, where $A_i, A_j \in V$ and $a_k \in \Sigma$. 
In this thesis we will restrict our attention to regular grammars since this class of grammars provides the necessary formalism for the definition of a Markov process, see Section 2.4.4.

A regular language is a language that can be generated by a regular grammar. In a regular grammar every sentential form has at most one nonterminal symbol. Therefore, a production of type $A_i \rightarrow a_j$ is called a final production because it terminates the derivation and generates a final string. A production of type $A_i \rightarrow a_j A_j$ is called a transitive production because it generates a sentential form. The derivation length, $|d_w|$, of a string $w$ is defined as the number of productions used in its derivation. In a regular grammar the derivation length is equal to the string length, that is, $|d_w| = |w|$.

$$G = \langle V, \Sigma, S, P \rangle$$

- $V = \{A, B\}$
- $\Sigma = \{0, 1\}$
- $S = A$
- $P_1 : A \rightarrow 0A$
- $P_2 : A \rightarrow 1B$
- $P_3 : B \rightarrow 0A$
- $P_4 : B \rightarrow 1B$
- $P_5 : B \rightarrow 1$

Figure 2.1: An example of a regular grammar.

Figure 2.1 shows an example of a regular grammar in which $A$ is the unique start symbol. The only final production is $P_5 : B \rightarrow 1$ and all other productions are transitive productions. An example of a string generated by this grammar is $A \Rightarrow 1B \Rightarrow 11$. Note that the length of this string is 2, which also corresponds to the number of productions used in its derivation. Other examples of strings generated by the grammar in Figure 2.1 are:

- $A \Rightarrow 1B \Rightarrow 1B \Rightarrow 111$,
- $A \Rightarrow 0A \Rightarrow 01B \Rightarrow 011$,
- $A \Rightarrow 0A \Rightarrow 01B \Rightarrow 010A \Rightarrow 0101B \Rightarrow 01011$.

A grammar $G$ is ambiguous if there is a string $w \in \mathcal{L}(G)$ with at least two distinct derivations from the start symbol $S$. Otherwise, $G$ is unambiguous. A language $\mathcal{L}$ is unambiguous if there is an unambiguous grammar $G$ such that $\mathcal{L} = \mathcal{L}(G)$. Otherwise, $\mathcal{L}$ is inherently ambiguous. It is shown in (Salomaa, 1973) that for every regular language there is an unambiguous grammar capable of generating it, that is, every regular
A grammar is reduced if every nonterminal symbol in $V$ can be reached by a derivation from the start symbol $S$, and if its subsequent derivation leads to a final string. In other words, in a non-reduced grammar there are nonterminal symbols which never derive a final string.

A regular language is closed under an operation $\mathcal{U}$ if and only if whenever operation $\mathcal{U}$ is applied to a regular language the result is a regular language. In (Salomaa, 1973) it is shown that the family of regular languages is closed under intersection and complementation.

### 2.4.2 Probabilistic Regular Grammars

Probabilistic languages are useful when the strings generated by a grammar are not all equally likely to occur. In a probabilistic language each string has an attached probability which measures its importance. Follows the formal definition of a probabilistic language according to (Booth and Thompson, 1973) and (Wetherell, 1980).

**Definition 2.5 (Probabilistic language).** A probabilistic language is a pair $L_p = (\mathcal{L}, p)$ where $\mathcal{L}$ is a language over an alphabet $\Sigma$ and $p$ is a function $p : \Sigma^* \rightarrow [0, 1]$ such that $p(w)$ represents the probability of string $w$ and meets the following conditions:

1. $w \not\in \mathcal{L} \Rightarrow p(w) = 0$ for all $w \in \Sigma^*$;
2. $w \in \mathcal{L} \Rightarrow 0 < p(w) \leq 1$ for all $w \in \Sigma^*$;
3. $\sum_{w \in \mathcal{L}} p(w) = 1$.

A probabilistic language is generated by a probabilistic grammar.

**Definition 2.6 (Probabilistic grammar).** A probabilistic grammar is a 2-tuple $G_p = (G, p)$ where $G$ is a grammar and $p$ is function $p : P \rightarrow [0, 1]$ such that $P$ is the set of productions and $p(P_i)$ represents a probability assigned to production $P_i$. The function $p$ meets the following condition:

$$\{\forall j | 1 \leq j \leq n\}, \quad \sum_{P_i \in C_j} p(P_i) = 1,$$

where $C_j$ is the class of productions in $P$ which have the $A_j$, $1 \leq j \leq n$, nonterminal symbol in their left-hand side.
The probability of a string derivation, \( d_w : S \Rightarrow^* w \), is given by the product of the probabilities of the productions used in the derivation of the string:

\[
p(d_w) = \prod_{1 \leq k \leq |d_w|} p(P_k),
\]

where \( p(P_k) \) is the probability of the \( k \)th, \( 1 \leq k \leq |d_w| \), production in the derivation and \(|d_w|\) is the derivation length. The probability of a string, \( p(w), w \in \mathcal{L}(G) \), is given by the sum of the probabilities of all its derivations in \( G \):

\[
p(w) = \sum_{d_w \in G} p(d_w).
\]

Note that, if a grammar is unambiguous we have that \( p(d_w) = p(w) \) since for each string there is only one derivation.

In addition, a probabilistic grammar \( G_p \) is consistent if it generates a probabilistic language, that is, if:

\[\sum_{w \in \mathcal{L}(G)} p(w) = 1.\]

The skeleton of a probabilistic grammar, \( G_p \), is the (not probabilistic) grammar \( G \) having the production \( X \rightarrow \beta \) if and only if \( G_p \) has the production \( X \rightarrow \beta \) with probability \( p > 0 \).

**Definition 2.7 (Probabilistic regular grammar).** A probabilistic regular grammar is a probabilistic grammar whose skeleton is a regular grammar.

According to (Wetherell, 1980) a probabilistic grammar is inconsistent when some of the production probabilities are not distributed to the final strings. It can be shown that every probabilistic regular grammar whose skeleton is a reduced regular grammar is consistent.

**Proposition 2.8.** Every reduced probabilistic regular grammar is consistent.

**Proof:** The test proposed in (Wetherell, 1980) to verify the consistency of a grammar consists in building a square matrix \( A \) of order \(|V|\) where \( A_{ij} \) corresponds to the expected number of times the nonterminal \( A_j \) will occur after a one-step derivation from \( A_i \). If there is a \( n, n < \infty \), such that matrix \( A^n \) has all row sums \( < 1 \) then the grammar is consistent. In \( A^n, A^n_{ij} \) corresponds to the number of times the nonterminal \( A_j \) will occur after the \( n \)-step derivations from \( A_i \).
2.4. Probabilistic Languages

In a regular grammar every sentential form has at most one non-terminal and if every nonterminal can derive a terminal symbol (i.e., the grammar is reduced) it will eventually lead to a final string, therefore, every row-sum tends to 0 when \( n \to \infty \). □

\[
G_p = \langle G, p \rangle = \langle (V, \Sigma, P, S), p \rangle
\]

\[
\begin{align*}
V &= \{A, B\} \\
\Sigma &= \{0, 1\} \\
S &= A \\
P_1 &: A \to 0A \quad p(P_1) = 0.5 \\
P_2 &: A \to 1B \quad p(P_2) = 0.5 \\
P_3 &: B \to 0A \quad p(P_3) = 0.3 \\
P_4 &: B \to 1B \quad p(P_4) = 0.4 \\
P_5 &: B \to 1 \quad p(P_5) = 0.3
\end{align*}
\]

Figure 2.2: An example of a probabilistic regular grammar.

Figure 2.2 shows an example of a probabilistic grammar whose skeleton is the grammar if Figure 2.1. With this grammar the derivation of string 011 is:

\[ d_{011} = A \Rightarrow 0A \Rightarrow 01B \Rightarrow 011, \]

and its probability is:

\[ p(011) = p(P_1) \cdot p(P_2) \cdot p(P_5) = 0.5 \cdot 0.5 \cdot 0.3 = 0.075. \]

2.4.3 Finite Automata

An alternative way of representing a formal language is via an accepting device called an automaton. Regular languages are accepted by deterministic finite automata, which are defined in (Hopcroft and Ullman, 1979) as:

**Definition 2.9 (DFA).** A deterministic finite automaton, DFA, is a five-tuple \( M = < Q, \Sigma, \delta, q_0, F > \) where \( Q = \{q_1, \ldots, q_n\} \) is a finite set of states, \( \Sigma = \{a_1, \ldots, a_m\} \) is a finite input alphabet, \( q_0 \in Q \) is the initial state, \( F \subseteq Q \) is the set of final states, and \( \delta \) is the transition function mapping of \( Q \times \Sigma \) to \( Q \) such that \( \delta(q_i, a) \) represents the state reached when in state \( q_i \) the input symbol \( a \) is read.

**Definition 2.10 (NFA).** A finite automaton is nondeterministic, NFA, if it allows zero, one, or more transitions from a state on the same input symbol, that is, \( \delta \) is a relation and not necessarily a mapping.

The definition of \( \delta \) can be extended to be a mapping of \( Q \times \Sigma^* \) to \( Q \) such that \( \delta(q_i, w) \) is the state reached when in state \( q_i \) the input string \( w \) is read. Note that \( \delta(q, \varepsilon) = \)
2.4. Probabilistic Languages

A string $w$ is said to be accepted by a finite automaton if $\delta(q_0, w) = p$ for some $p \in F$.

The language accepted by a finite automaton $M$ is the set $\mathcal{L}(M) = \{ w | \delta(q_0, w) \in F \}$. It is shown in (Hopcroft and Ullman, 1979) that a language is regular if there is a finite automaton that accepts it.

In a DFA for each state there is a unique transition on each symbol. Therefore, there is exactly one path for a given input string $w \in \Sigma^*$ starting at $q_0$ meaning that every DFA is unambiguous. Every regular grammar whose corresponding finite automaton is a DFA is unambiguous; see (Hopcroft and Ullman, 1969) for the algorithm to obtain the finite automaton corresponding to a regular grammar.

\[
M = (Q, \Sigma, \delta, q_0, F) \quad \text{with} \quad Q = \{ A, B \}, \quad \Sigma = \{ 0, 1 \}, \quad \delta = \begin{cases} 
\delta_1 : \delta(A, 0) = A \\
\delta_2 : \delta(A, 1) = B \\
\delta_3 : \delta(B, 0) = A \\
\delta_4 : \delta(B, 1) = B 
\end{cases}
F = \{ B \}
\]

Figure 2.3: The finite automaton corresponding to the grammar in Figure 2.1.

Figure 2.3 gives the definition of the automaton corresponding to the grammar in Figure 2.1. Alternatively, a finite automaton can be represented by a transition diagram as the one given in Figure 2.4.

As an example we now give the acceptance path for the string 111 in the automaton defined in Figure 2.3:

\[
\delta(A, 111) = \delta(\delta(A, 11), 1) = \delta(\delta(A, 1), 1) = \delta(B, 1) = B \in F.
\]

A probabilistic finite automaton (PFA) is a finite automaton that has a probability attached to each transition between states. A PFA having at most one transition between every two states corresponds to a Markov chain, see Section 2.4.4.
2.4.4 Markov Chains to Model Probabilistic Regular Grammars

As it was referred in Section 2.3.3.3 the model we propose in this thesis for the study of user web navigation corresponds to a Markov chain. A finite Markov chain, (Kemeny and Snell, 1960; Feller, 1968; Norris, 1997), models a process which is characterised by a finite set of states. The process moves from state to state and the probability of moving next to a given state depends only on the present state. A matrix of transition probabilities characterises the probability of the transitions between any two states. A Markov chain is a natural model for the derivation process of a probabilistic regular grammar. In fact, in a sentential form generated by a regular grammar there is at most one nonterminal symbol and the choice of the production to be applied next depends only on such symbol. The theory of finite Markov chains is useful for the computation of statistical properties of a probabilistic regular language.

A Markov chain is characterised by a set of states \( X = \{x_1, \ldots, x_n\} \), a row vector with the states' initial probabilities \( \pi = \{\pi_1, \ldots, \pi_n\} \) and a square transition matrix \( T \) of order \( n \) representing the one-step transition probabilities between states.

The states of a Markov chain are classified according to their accessibility. We say that a set of states is an ergodic set if every state can be reached from every other state in that set and the set cannot be left once entered. On the other hand, a set of states form a transient set when every state is reachable from any other state in the set but the set can be left. The states in a transient set are the transient states. A state which is in an ergodic set is called an ergodic state. Moreover, if a state is the unique element in an ergodic set it is called an absorbing state since once entered it cannot be left, that is, \( T_{ii} = 1 \). A chain whose non-transient states are absorbing is called an absorbing Markov chain. A chain whose set of all states form an ergodic set is called irreducible Markov chain. A Markov chain is periodic if the number of steps necessary to return to a state after leaving it is a multiple of some integer \( p > 1 \). Otherwise the chain is aperiodic.

Figure 2.5 gives examples of different types of Markov chains which are represented by their transition diagrams. The chain in Figure 2.5 (a) is an absorbing Markov chain since it contains the absorbing state \( C \). The states \( A \) and \( B \) form a transient set
2.4. Probabilistic Languages

(a) (b) (c)

Figure 2.5: Examples of Markov chains.

of states. The chain in Figure 2.5 (b) contains a set of transient states, \( \{A, B\} \), and an ergodic set of states, \( \{C, D\} \). This chain will eventually enter the ergodic set of states. When that occurs the chain is reduced to its ergodic set and its behaviour can be studied as an irreducible and aperiodic chain. Finally, a periodic chain is represented in Figure 2.5 (c). Note that, after leaving state \( C \) a multiple of 2 steps are needed to return to it. This last chain is also irreducible.

We will now give the method to obtain the Markov chain corresponding to a probabilistic regular grammar.

Consider a probabilistic regular grammar \( G_p = \langle G, p \rangle \) where \( G = \langle V, E, S, P \rangle \) and \( V = \{A_1, \ldots, A_n\} \). The corresponding Markov chain will be built as follows:

1. \( X = \{x_1, \ldots, x_n\} \cup \{F\} \) is the finite set of \( n+1 \) states. \( F \) is an additional final state.

2. \( \pi = \{\pi_1, \ldots, \pi_{n+1}\} \) is the vector of initial probabilities. Since \( A_i = S \) is a unique start state we make \( \pi_i = 1 \) and for \( A_j \neq S \) we make \( \pi_j = 0 \).

3. The transition matrix \( T \) is defined as follows:
   
   (i) for all \( i \) and \( j \) we make \( T_{i,j} = 0 \);
   
   (ii) if there is a production rule of the type \( A_i \rightarrow a_j A_j \) we make \( T_{i,j} = p(A_i \rightarrow a_j A_j) \);
   
   (iii) if there is a production of type \( A_i \rightarrow a_j, a_j \in \Sigma \), we make \( T_{i,n+1} = p(A_i \rightarrow a_j) \), where \( n+1 \) corresponds to the state \( F \).
2.5. Hypertext Probabilistic Automata

\[ X = \{A, B, F\} \]
\[ \pi = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \]
\[ T = \begin{bmatrix} 0.5 & 0.5 & 0 \\ 0.3 & 0.4 & 0.3 \\ 0 & 0 & 1 \end{bmatrix} \]

Figure 2.6: The Markov chain equivalent to the grammar given in Figure 2.2.

(iv) finally, we make \( T_{n+1,n+1} = 1 \), where \( n+1 \) corresponds to the state \( F \).

In Figure 2.6 we give the Markov chain corresponding to the probabilistic regular grammar given in the example of Figure 2.2. Note that with the transformation method given the resulting chain has an absorbing state \( F \). The chain reaches the absorbing state when a final string is obtained.

The theory of Markov chains is useful for studying global measures of the derivation process. Examples of such measures are the average and the variance of the string length, the expected number of occurrences of a particular nonterminal until the chain is absorbed, or the probability of the chain going to state \( q \) after leaving state \( q_i \). Techniques to compute such measures are described in (Kemeny and Snell, 1960).

2.5 Hypertext Probabilistic Automata

In (Levene and Loizou, 1999c) the authors propose to model the semantics of a hypertext database (see Definition 2.1) by means of a hypertext finite automaton (HFA), which is a special case of a deterministic finite automaton. In a hypertext finite automaton each state corresponds to a page in the hypertext system and there is a one-to-one correspondence between the set of states and the automaton's alphabet. A transition from a state to another is allowed only if there is a hyperlink connecting the corresponding hypertext pages. Moreover, every state is both an initial and a final state meaning that an user can enter the hypertext system at any page and leave it at any page. A string is accepted by the automaton if it corresponds to a trail a user can follow in the hypertext system being modelled.

A HFA can have probabilities attached to its transitions in order to assign a measure of importance to the strings accepted by it. Such a HFA is a hypertext probabilistic automaton (HPA) and was first introduced by (Levene and Loizou, 1999d). We now
2.6 Information Theory

present the definition of a HPA.

Definition 2.11 (Hypertext probabilistic automata). A hypertext probabilistic automaton (abbreviated HPA) with cut-point \( \lambda \) representing a Hypertext database, \( \mathcal{H} = \langle S, \mathcal{R} \rangle \), is a six-tuple of the form \( P^\lambda_{\mathcal{H}} = \langle A, Q, M, \pi, Q^i, Q^f \rangle \), where

1. \( A = \{ a_1, \ldots, a_n \} \) is a finite alphabet, with \( n = |S| \).
2. \( Q = \{ q_1, \ldots, q_n \} \) is the set of states, with \( n = |S| \).
3. \( M = \{ p(q_i, q_j) \} \) is a stochastic \( n \times n \) matrix, that is \( \forall q_i, q_j \in Q, p(q_i, q_j) \geq 0 \) and \( \forall q_i \in Q, \sum_{j=1}^n p(q_i, q_j) = 1 \), where \( p \) is a function from \( S \times S \) to \([0, 1]\). In addition, \( M \) satisfies the constraint that \( p(s_i, s_j) > 0 \) if and only if \( (s_i, s_j) \in \mathcal{R} \).
4. \( \pi = \{ p(q_i) \} \) is a \( n \)-dimensional stochastic row vector, that is \( \forall q_i \in Q, p(q_i) \geq 0 \) and \( \sum_{i=1}^n p(q_i) = 1 \). In addition, \( \pi \) satisfies the constraint that \( p(s_i) > 0 \) if and only if \( \exists s_j \in Q \) such that either \((q_i, q_j) \in \mathcal{R}\) or \((q_j, q_i) \in \mathcal{R}\).
5. \( Q^i = Q \) and \( Q^f = Q \) are the sets of initial and final states, respectively.

Note that a HPA is a special case of the more general definition of a probabilistic finite automaton given in (Paz, 1971; Rabin, 1963; Fu, 1971).

The probability of a string, \( w = a_1 \ldots a_n \), being accepted by the automaton is given by:

\[
p(w) = p(q_1) \cdot p(q_1, q_2) \cdot \ldots \cdot p(q_{n-1}, q_n).
\]

Finally, the language accepted by a HPA is defined as being composed by the set of all strings whose acceptance probability is greater than a cut-point, \( \lambda \).

Definition 2.12 (HPA language). Let \( M \) be an HPA, the language accepted by \( M \) with cut-point \( \lambda \) is \( L(M, \lambda) = \{ w \in \Sigma^*, p(w) > \lambda \} \).

2.6 Information Theory

In this section we present a brief introduction to the information theoretic measure of entropy. The reader is referred to (Cover and Thomas, 1991) for more detail.
The entropy is a measure of the uncertainty in the outcome of a random variable. For example, if we have a random variable characterising the outcome of an experiment, the entropy is a measure of the amount of information required on average to describe such an experiment. If all the outcomes of a random variable are equally likely the uncertainty is maximal. In addition, an experiment with more alternative results is expected to have higher uncertainty.

We let \( Y \) be a random variable with range \( \{y_1, y_2, \ldots, y_n\} \), where \( p(y_i) = p_i \) denotes the corresponding probabilities. The entropy of \( Y \), \( H(Y) \), is defined in (Shannon, 1948) as:

\[
H(Y) = - \sum_{i=1}^{n} p(y_i) \log p(y_i).
\]

When the \( \log \) function is on base 2 the entropy corresponds to the number of bits necessary to characterise the variable. The entropy can be viewed as the average number of binary questions necessary to guess the value of \( Y \).

When the outcome of the variable is certain the entropy takes its minimum value. The entropy takes its maximum value when the random variable is uniformly distributed. For example, if \( Y = \{y_1, y_2, y_3\} \) is such that \( p(y_1) = 1/4, p(y_2) = 2/4 \) and \( p(y_3) = 1/4 \), the entropy of \( Y \) is

\[
H(Y) = - (\frac{1}{4} \log \frac{1}{4} + \frac{2}{4} \log \frac{1}{2} + \frac{1}{4} \log \frac{1}{4}) = 1.5.
\]

If \( p(y_1) = p(y_2) = p(y_3) = 1/3 \) we have that

\[
H(Y) = - \left( \frac{1}{3} \log \frac{1}{3} + \frac{1}{3} \log \frac{1}{3} + \frac{1}{3} \log \frac{1}{3} \right) = \log(3) = 1.585,
\]

and if \( p(y_1) = 1, p(y_2) = 0 \) and \( p(y_3) = 0 \) there is no uncertainty in the outcome of the variable, therefore, the entropy is \( H(Y) = 0 \).

When a random variable \( Y \) is broken down into two successive random variables \( Y_1 \) and \( Y_2 \), its entropy, \( H(Y) \), can be obtained by the weighted sum of the individual values of \( H(Y_1) \) and \( H(Y_2) \), see Figure 2.7.

If we have two random variables, \( Y = \{y_1, y_2, \ldots, y_n\} \) and \( X = \{x_1, x_2, \ldots, x_m\} \), with a joint distribution \( p(x_i, y_j) \) the joint entropy or the two variables is defined as:

\[
H(X, Y) = - \sum_{i=1}^{m} \sum_{j=1}^{n} p(x_i, y_j) \log p(x_i, y_j).
\]
Moreover, the conditional entropy of \( X \) given \( Y \) is:

\[
H(X/Y) = -\sum_{i=1}^{n} p(y) \sum_{j=1}^{n} p(x_i/y_j) \log p(x_i/y_j),
\]

where \( p(x_i/y_j) \) denotes the conditional probability \( x_i \) given that the outcome of \( y_j \) is known. Having an experiment characterised by two random variables, the conditional entropy is a measure of the uncertainty remaining after the outcome of one of the variables is known. If the variables are independent the conditional entropy is equal to the variable entropy.

The conditional and joint entropies are related in a way similar to that of conditional and joint probabilities, that is

\[
H(X, Y) = H(X) + H(Y/X).
\]

Moreover the following two inequalities hold, noting that equality is verified if and only if the two variables are independent:

\[
H(X, Y) \leq H(X) + H(Y),
\]

\[
H(X/Y) \leq H(X).
\]

2.7 Summary

In this chapter we have presented an overview of research done in the field of web technologies. The definition of the navigation problem was given and several approaches aimed at helping users to overcome it were presented. In addition, we have presented a brief overview of the field of data mining and knowledge discovery and its application to web data. An overview of the main research projects in the field of web usage...
mining was given. Finally, we have presented the concepts relevant to the thesis from the theory of formal languages, discrete Markov chains, and information theory.
Chapter 3

Hypertext Probabilistic Grammars

In this chapter we present the formal definition of the hypertext probabilistic grammar model. A collection of users’ navigation sessions, inferred from log files (or any other similar source), provides the input to the model which represents the subset of the hypertext system traversed by these users. We also present an algorithm to compute all the grammar strings with the required characteristics, as well as an extension of the model which takes into account the assumed user memory via the Ngram concept; see (Charniak, 1996). Finally, the concept of hypertext probabilistic grammar entropy is presented which provides a useful measure of the model’s properties.

3.1 Formal Definition

A hypertext system is essentially a network of linked documents together with an interface that allows a user to browse the documents’ content and navigate across links. When browsing a hypertext system, such as the web, a user is able to access the contents of the pages in a non-sequential way, as opposed to traditional information systems which are mainly sequential in nature. In fact, a user has the flexibility of starting the navigation at any given page and choosing to follow only the links to pages that are of interest to him. The essential characteristics of a hypertext system were previously formalised as a hypertext database, (Levene and Loizou, 1999c), which is a directed graph whose nodes represent units of information and whose arcs allow a user to follow links between the information units; see Definition 2.1. The links connecting the pages determine which are the pages a user can visit in sequence when browsing the hypertext
Figure 3.1: An example of a web topology.

<table>
<thead>
<tr>
<th>ID</th>
<th>Trail</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A₁ → A₂ → A₃ → A₄</td>
</tr>
<tr>
<td>2</td>
<td>A₁ → A₅ → A₃ → A₄ → A₁</td>
</tr>
<tr>
<td>3</td>
<td>A₅ → A₂ → A₄ → A₆</td>
</tr>
<tr>
<td>4</td>
<td>A₅ → A₂ → A₃</td>
</tr>
<tr>
<td>5</td>
<td>A₅ → A₂ → A₃ → A₆</td>
</tr>
<tr>
<td>6</td>
<td>A₄ → A₁ → A₅ → A₃</td>
</tr>
</tbody>
</table>

Figure 3.2: An example of a collection of user navigation trails.

system. We call the set of links in the hypertext system the *reachability relation*, \( \mathcal{R} \), as they represent all the trails the user can follow. The formal definition of a trail in a hypertext system now follows.

**Definition 3.1 (Trail).** A *trail* is a finite sequence of pages visited by following links in a hypertext system.

In Figure 3.1 we give an example of a web topology and in Figure 3.2 we give an example of a collection of navigation trails a user can follow in the topology. We will use this topology and set of trails as a running example throughout the chapter. From now on, we use the terms web and hypertext system interchangeably as well as the terms page and hypertext page.

The methodology of the hypertext probabilistic grammar model is summarised in Figure 3.3. A collection of user navigation sessions is inferred from log data and modelled as a hypertext weighted grammar (see Definition 3.4). The weighted grammar is
then transformed into a hypertext probabilistic grammar (see Definition 3.6) to which data mining techniques are applied in order to extract patterns from the navigation sessions.

In this thesis we make use of the theory of regular grammars (see Section 2.4) to formalise the user interaction with the web, which is given by the set of trails a user followed. The collection of admissible user navigation trails is modelled as a hypertext grammar (or simply HG) which is in the class of regular grammars and will be our representation of a hypertext database. We freely utilise the duality between grammars and automata in both our figures and terminology, (Hopcroft and Ullman, 1979), for example, we sometimes refer to a nonterminal as a state since there is a state corresponding to each nonterminal in the equivalent automaton.

Figure 3.3: The methodology of the hypertext probabilistic grammar model.

Note that according to Definition 2.1 $S$ represents the set of states in the hypertext database and $\mathcal{R}$ the reachability relation; $\vert S \vert$ represents the number of states in the hypertext database. The formal definition of a hypertext grammar now follows.

**Definition 3.2 (Hypertext grammar).** A hypertext grammar representing a hypertext database, $\mathcal{H} = < S, \mathcal{R} >$, is a four-tuple $HG = < V, \Sigma, S, P >$ where:

1. $V = \{ S, A_1, \ldots, A_{\vert S \vert}, F \}$ is a finite alphabet of $\vert S \vert + 2$ nonterminal symbols.
   
   $S$ is a unique start state, $F$ is a unique final state, and there is a one-to-one and onto mapping from $V \setminus \{ S, F \}$ to $S$,

2. $\Sigma = \{ a_1, \ldots, a_{\vert S \vert} \}$ is a finite alphabet of terminal symbols such that there is a one-to-one and onto mapping from $\Sigma$ to $S$, or equivalently from $\Sigma$ to $V \setminus \{ S, F \}$,

3. $P$ is a finite set of productions of the form $X \rightarrow yY$ with $X, Y \in V$ and $y \in \Sigma \cup \epsilon$ such that:

   (i) $S \rightarrow a_iA_i \in P$, $\forall A_i \in V$, 


(ii) $A_i \rightarrow a_j A_j \in P$ if and only if $(s_i, s_j) \in \mathcal{R}$,
(iii) $A_i \rightarrow F \in P$, $\forall A_i \in V$,
(iv) $F \rightarrow \epsilon \in P$.

In a hypertext grammar each nonterminal symbol $A_i \in V - \{S, F\}$ corresponds to a page and the productions of type (ii) represent the links between pages (the reachability relation). Both $S$ and $F$ are artificial states representing the start and finish state of a navigation session, and while productions of type (i) mean that the navigation can start at any given page, productions of type (iii) mean that the user can stop the navigation at any page. Finally, the production (iv) derives a final string, composed only of terminal symbols, by eliminating the nonterminal symbol $F$. We call the productions having symbol $S$ on their left-hand side start productions, the productions corresponding to links transitive productions, and the productions having symbol $F$ on their right-hand side final productions.

<table>
<thead>
<tr>
<th>Hypertext grammar’s productions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i) $S \rightarrow a_1 A_1$</td>
</tr>
<tr>
<td>$P_{1,1}$</td>
</tr>
<tr>
<td>(ii) $A_1 \rightarrow a_2 A_2$</td>
</tr>
<tr>
<td>$P_{2,1}$</td>
</tr>
<tr>
<td>(iii) $A_2 \rightarrow a_1 A_1$</td>
</tr>
<tr>
<td>$P_{2,3}$</td>
</tr>
<tr>
<td>(iv) $F \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$P_{3,1}$</td>
</tr>
<tr>
<td>$P_{3,2}$</td>
</tr>
<tr>
<td>$P_{3,3}$</td>
</tr>
<tr>
<td>$P_{3,4}$</td>
</tr>
<tr>
<td>$P_{3,1}$</td>
</tr>
<tr>
<td>(ii) $A_5 \rightarrow a_3 A_3$</td>
</tr>
<tr>
<td>$P_{4,1}$</td>
</tr>
<tr>
<td>(iii) $A_3 \rightarrow a_4 A_4$</td>
</tr>
<tr>
<td>$P_{4,2}$</td>
</tr>
<tr>
<td>(iv) $F \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$P_{5,1}$</td>
</tr>
<tr>
<td>$P_{5,2}$</td>
</tr>
<tr>
<td>(ii) $A_6 \rightarrow a_5 A_5$</td>
</tr>
<tr>
<td>$P_{5,3}$</td>
</tr>
<tr>
<td>(iii) $A_6 \rightarrow a_7 A_7$</td>
</tr>
<tr>
<td>$P_{6,1}$</td>
</tr>
<tr>
<td>(iv) $F \rightarrow \epsilon$</td>
</tr>
<tr>
<td>$P_{6,2}$</td>
</tr>
<tr>
<td>$P_{6,3}$</td>
</tr>
<tr>
<td>$P_{7,1}$</td>
</tr>
<tr>
<td>(ii) $A_7 \rightarrow a_6 A_6$</td>
</tr>
<tr>
<td>$P_{7,2}$</td>
</tr>
<tr>
<td>(iii) $A_7 \rightarrow a_7 A_7$</td>
</tr>
<tr>
<td>$P_{8,1}$</td>
</tr>
<tr>
<td>(iv) $F \rightarrow \epsilon$</td>
</tr>
</tbody>
</table>

Figure 3.4: The production rules of the hypertext grammar corresponding to the web topology in the running example.

The productions are assigned a two-index numbering, $P_{i,j}$, to facilitate their reference. The first index identifies a group of productions with the same nonterminal in
3.1. Formal Definition

their left-hand side and the second index gives the position of a production within its group of productions. The numbering system is exemplified in Figure 3.4 which shows the productions corresponding to the HG representing the hypertext database in the running example.

The language generated by a hypertext grammar is called its hypertext language and corresponds to the trails a user can follow when navigating through the hypertext system.

We now show that every hypertext grammar is unambiguous (see Section 2.4), that is, there is only one possible derivation for any string in the grammar.

**Proposition 3.3 (HG are unambiguous).** Every hypertext grammar is unambiguous.

**Proof.** From Definition 3.2 it follows that there is a one-to-one and onto mapping from the set of terminal symbols, $\Sigma$, and the set of nonterminal symbols corresponding to pages, $V = \{S, F\}$. Since every transitive production is of the type $X \rightarrow a_i A_i, X \in V = \{F\}, A_i \in V = \{S, F\}$ and $a_i \in \Sigma$, the grammar corresponds to a deterministic finite automaton and, as such, there is only one possible derivation for each string in the grammar. □

As a result of Proposition 3.3 we make use of the terms trail and string interchangeably.

The user interaction with a hypertext system is characterised by a collection of user navigation sessions which consists of the collection of trails traversed by the user when navigating within a given time-window. One example of such user sessions consists of sessions inferred from server log files (see Section 2.3.3). In very simplistic terms, a user session can be defined as a sequence of requests made by a given user such that no two consecutive requests are separated by more than $X$ minutes, where $X$ is a given parameter. It should be noted that more advanced data preparation techniques, such as those described in (Cooley et al., 1999), can be used in the data pre-processing stage in order to take full advantage of the information available in the log files. Another example of user sessions are those collected by a web browser that is setup to store a personal log file, via a proxy, containing the user browsing history when navigating the web. The model we propose assumes as its input a collection of user navigation sessions...
and it is out of the scope of this work to fully characterise the necessary techniques needed for pre-processing the data.

While a hypertext grammar models the set of trails a user can traverse in a hypertext system, the actual user interaction with it is modelled as an hypertext probabilistic grammar (or simply HPG). A HPG is a HG where to each production we attached a probability. In order to formalise the HPG concept we first introduce the concept of hypertext weighted grammar (or simply HWG). A HWG is obtained by incorporating the information contained in the users’ sessions into the corresponding hypertext grammar. From a collection of user navigation sessions we obtain:

1. The number of times each page, $A_i$, was requested, $|A_i|$;
2. The number of times each page was the first state in a session, $|SA_i|$;
3. The number of times each page was the last state in a session, $|A_iF|$; and
4. The number of times a sequence of two pages appears in the sessions, $|A_iA_j|$, which gives the number of times the corresponding link was traversed.

A HWG is a hypertext grammar where each production has an associated weight corresponding to either the number of times a session started in the corresponding page (in the case of start productions), the number of times the corresponding link was traversed (in the case of the transitive productions), or the number of times a session terminated in the corresponding state (in the case of final productions). The formal definition of a hypertext weighted grammar now follows.

**Definition 3.4 (Hypertext weighted grammar).** A hypertext weighted grammar is a 2-tuple $HWG = \langle HG, W \rangle$ where $HG$ is a Hypertext grammar and $W$ is an ordered set of weights, $w_{i,j} \in W$, that are assigned to the corresponding productions $P_{i,j} \in P$. The weights for the productions are assigned according to the expressions:

(i) $W(S \rightarrow a_iA_i) = |SA_i|$,  
(ii) $W(A_i \rightarrow a_jA_j) = |A_iA_j|$,  
(iii) $W(A_i \rightarrow F) = |A_iF|$,  
(iv) $W(F \rightarrow \epsilon) = 1$.  


Every HWG is a reduced grammar, see Section 2.4, in the sense that any state is reachable with a derivation from the start symbol and its derivation leads to a final string. A final string is a string containing only terminal symbols.

**Proposition 3.5 (HWG are reduced).** Every hypertext weighted grammar is a reduced grammar.

**Proof.** This property holds trivially since every state in a HWG occurred in at least one user navigation session starting at the start state $S$ and finishing at the final state $F$. Therefore, every state in a grammar inferred from the navigation sessions is included in a path from $S$ to $F$. □

Moreover, every state in a HWG is balanced. A state is said to be balanced if the sum of the weights of its in-links is equal to the sum of the weights of its out-links. We call the value resulting from the sum of the in-links or out-links of a state the *state degree*. The degree of a state is equal to the number of times the corresponding page was requested, $|A_i|$

Figure 3.5 shows the automaton corresponding to the hypertext weighted grammar inferred from the collection of trails in the running example; the value of a state degree is given by the number in bold and italic next to that state. We note that only the productions with positive weights are represented. Therefore, only the pages which were browsed at the beginning of a session have a corresponding start production, only the pages which were browsed at the end of a session have a corresponding final production, and only the transitive productions that correspond to links traversed by the user are included in the HWG.

The *normalisation* of a HWG results in a HPG and consists in the computation of the production probabilities, which are proportional to the production weights. To that effect, we provide the analyst with $\alpha \in [0, 1]$ as a parameter to attach the desired weight to the fact of a state corresponding to the first page browsed in a navigation session. If $\alpha = 0$, only states which were the first in a session have probability greater than zero of being in a start production. In this case, only those strings which start with a state that was the first in a navigation session are induced by the grammar. On the other hand,
Figure 3.5: The transition diagram for the hypertext weighted grammar corresponding to the collection of trails in the running example.

if $\alpha = 1$ the probability of state being in a start production is proportional to the total number of times the state was visited in the collection of navigation sessions. Therefore, when $\alpha = 1$ the probability of a start production is proportional to the number of times the corresponding state was visited, implying that the destination node of a production with higher probability corresponds to a state that was visited more often. The parameter $\alpha$ can take any value between 0 and 1, providing a balance between the two scenarios described above. As such, $\alpha$ gives the analyst the ability to tune the model for the search of different types of patterns in the user navigation. Finally, the probability of a transitive production is assigned in such a way that it is proportional to the frequency with which the corresponding link was traversed. The formal definition of a hypertext probabilistic grammar now follows.

**Definition 3.6 (Hypertext probabilistic grammar).** A hypertext probabilistic grammar with cut-point $\lambda \in (0, 1)$ representing a hypertext database, $\mathcal{H} = < S, R >$, is a 2-tuple $HPG = < HWG, P >$ where $HWG$ is a hypertext weighted grammar and $P$ is a set of probabilities, $p_{i,j} \in P$, that are assigned to the corresponding productions $P_{i,j} \in P$. The production probabilities are assigned according to the following expressions:

(i) $p(S \rightarrow a_i A_i) = \alpha \frac{|A_i|}{\sum_{j=1}^{m} |A_j|} + (1 - \alpha) \frac{|S A_i|}{\sum_{j=1}^{m} |S A_j|}$,

(ii) $p(A_i \rightarrow a_j A_j) = \frac{|A_i A_j|}{|A_i|}$,

(iii) $p(A_i \rightarrow F) = \frac{|A_i F|}{|A_i|}$,

(iv) $p(F \rightarrow \epsilon) = 1$. 

### Hypertext probabilistic grammar’s productions

<table>
<thead>
<tr>
<th>Production</th>
<th>$\alpha = 0$</th>
<th>$\alpha = 0.5$</th>
<th>$\alpha = 1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1 \rightarrow a_1 A_1$</td>
<td>0.33</td>
<td>0.25</td>
<td>0.167</td>
</tr>
<tr>
<td>$A_2 \rightarrow a_2 A_2$</td>
<td>0.0</td>
<td>0.08</td>
<td>0.167</td>
</tr>
<tr>
<td>$A_3 \rightarrow a_3 A_3$</td>
<td>0.0</td>
<td>0.11</td>
<td>0.208</td>
</tr>
<tr>
<td>$A_4 \rightarrow a_4 A_4$</td>
<td>0.17</td>
<td>0.17</td>
<td>0.167</td>
</tr>
<tr>
<td>$A_5 \rightarrow a_5 A_5$</td>
<td>0.50</td>
<td>0.35</td>
<td>0.208</td>
</tr>
<tr>
<td>$A_6 \rightarrow a_6 A_6$</td>
<td>0.0</td>
<td>0.04</td>
<td>0.083</td>
</tr>
</tbody>
</table>

**Figure 3.6:** The production rules of the hypertext probabilistic grammar corresponding to the collection of trails in the running example.

Figure 3.6 shows the production probabilities for the HPG in the running example (see Figure 3.2) for three different values of the $\alpha$ parameter. In the example, we have a collection of 6 user navigation sessions with a total of 24 page requests where, for example, page $A_1$ was requested 4 times, 2 of which as the first state in a session. Therefore, for $\alpha = 0.5$ we have the probability of production $S \rightarrow a_1 A_1$ given by $p(S \rightarrow a_1 A_1) = 0.5 \cdot 4/24 + (1 - 0.5)2/6 = 0.25$. When $\alpha = 0$ then $p(S \rightarrow a_1 A_1) = (1 - 0)2/6 = 0.33$. Moreover, $p(A_2 \rightarrow a_3 A_3) = 3/4 = 0.75$ and $p(A_4 \rightarrow F) = 1/4 = 0.25$.

We will now show that every HPG is a reduced grammar.

**Proposition 3.7 (HPG are reduced).** Every hypertext probabilistic grammar is a reduced grammar.

**Proof.** All productions in a HWG have probability greater than zero in the corresponding HPG, therefore, the normalisation of a HWG maintains all its paths from $S$ to $F$ and can introduce some alternative ones when $\alpha > 0$. $\square$

We say that a string is included in the HPG’s language if its derivation probability is above a given cut-point, $\lambda$. The derivation probability is given by the product of the
probabilities of the productions used in the derivation, \( p(w) = \prod_{1 \leq k \leq |d_w|} p(P_k) \), where \( p(P_k) \) is the probability of the \( k^{th} \) production, \( 1 \leq k \leq |d_w| \), and \( |d_w| \) the derivation length. The HPG's cut-point is composed of two distinct thresholds \( \lambda = \theta \cdot \delta \), where \( \theta \in (0, 1) \) is the support threshold and \( \delta \in (0, 1) \) the confidence threshold.

The intuition behind the support, \( \theta \), is that it is the factor of the cut-point responsible for pruning out the strings whose first derivation step has low probability, corresponding to a subset of the hypertext system rarely visited. Note that, when \( \alpha > 0 \) there can be up to \( n \) start productions with probability greater than zero, where \( n \) is the number of grammar states. Therefore, if \( n \) is large the probabilities of these productions are of a much smaller order than those of the transitive productions and the gap is aggravated the larger the grammar is. Thus, when setting the support value we should take into account the number of grammar states. For example, if \( \alpha > 0 \) and all the pages have an initial probability greater than zero, \( \theta = 1/n \) means that only pages which were visited a number of times above the average will be considered as being the first symbol of a string. Similarly, when \( \alpha = 0 \) we should take into account the number of navigation sessions when setting the value of the support threshold. In this case, the number of start productions with probability greater than zero is equal to the number of states that were the first in a session, \( f \). Therefore, the average probability of such productions is \( 1/f \) and if the support is set with this value only productions having a value above the average will pass the support threshold test.

The confidence is the factor of the cut-point responsible for pruning out strings whose derivation contains transitive productions with small probability. In order to set the confidence value we should take into account the grammar's branching factor, that is, the average number of out-links per page in the underlying hypertext topology. For example, if the hypertext system has, on average, five out-links per page the average link probability is \( 1/5 = 0.2 \). Therefore, if we want to include in the grammar's language the strings derived by productions with probability above the average and composed by \( m \) symbols, the confidence threshold should be set to \( \lambda = 0.2^{(m-1)} \). Note that the factor \( (m-1) \) is due to the fact that the first symbol in the string is obtained by the start productions which are evaluated against the support threshold.
The values of the support and confidence thresholds give the analyst control over the number, probability and length of the trails to be included in the grammar’s language. In addition, by allowing the analyst to specify the cut-point value by means of its components it is made easier for him to gain some insight on its value.

The formal definition of a hypertext probabilistic language now follows.

**Definition 3.8 (Hypertext probabilistic language).** The language generated by a $HPG$ is $\mathcal{L}^\lambda = \{ w \mid w \in \Sigma^*, S \Rightarrow^* w \text{ and } p(w) > \lambda, \text{ with } \lambda \in (0, 1) \}$.

In Definition 3.8 we choose to consider the case where the cut-point has a fixed value for all derivation steps. An alternative definition could be to set the cut-point with the value $\lambda = \theta$ when evaluating the start productions and with the value $\lambda = \theta \delta$ when evaluating longer trails. When using this alternative definition the support is more effective in pruning out strings with low initial probability; a method which makes use of this definition is proposed in Section 5.4.

Note that a string can be included in a HPG’s language without its derivation reaching the final state $F$. We stipulate that the sub-trails traversed on the way to the final state $F$ should also belong to the language since they represent trails navigated by the user. Therefore, a string whose generation terminates in the state $F$ contains additional information, meaning that the user not only traversed the corresponding trail but also finished the session in the last state of the derivation.

Given a $HPG$ and its cut-point the rule-set is defined as the set of maximal strings (or maximal trails) in the language, where a string is maximal if it is not a proper prefix of any other string in the language. Therefore, while a maximal string is a rule and is explicitly given in the rule-set, a non-maximal strings is a sub-rule and is implicitly given by its maximal rule.

In Figure 3.7 we show the HPG’s strings for four different configurations of the model inferred from the sessions in the running example. The value of the support was fixed at $1/6$ while the confidence takes the values 0.4 and 0.6. (We note that by defining the cut-point as $\lambda = \theta \delta$ it is possible to cover a range of its values by varying only one of its components, in this case the confidence.) From the first two sets of strings we can notice how both the number of strings and its average length decreases when the cut-
3.1. Formal Definition

<table>
<thead>
<tr>
<th>string</th>
<th>prob.</th>
<th>string</th>
<th>prob.</th>
<th>string</th>
<th>prob.</th>
<th>string</th>
<th>prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1A_5A_2$</td>
<td>0.075</td>
<td>$A_1A_5$</td>
<td>0.125</td>
<td>$A_1A_5$</td>
<td>0.083</td>
<td>$A_1$</td>
<td>0.167</td>
</tr>
<tr>
<td>$A_2$</td>
<td>0.083</td>
<td>$A_3$</td>
<td>0.104</td>
<td>$A_2A_3$</td>
<td>0.125</td>
<td>$A_2A_3$</td>
<td>0.125</td>
</tr>
<tr>
<td>$A_3$</td>
<td>0.104</td>
<td>$A_4$</td>
<td>0.167</td>
<td>$A_3A_4$</td>
<td>0.083</td>
<td>$A_3$</td>
<td>0.208</td>
</tr>
<tr>
<td>$A_4A_1$</td>
<td>0.083</td>
<td>$A_5A_2A_3$</td>
<td>0.159</td>
<td>$A_4A_1$</td>
<td>0.083</td>
<td>$A_4$</td>
<td>0.167</td>
</tr>
<tr>
<td>$A_5A_2A_3$</td>
<td>0.159</td>
<td>$A_5A_3$</td>
<td>0.142</td>
<td>$A_5A_2A_3$</td>
<td>0.094</td>
<td>$A_5A_2$</td>
<td>0.125</td>
</tr>
<tr>
<td>$A_5A_3$</td>
<td>0.142</td>
<td></td>
<td></td>
<td>$A_5A_3$</td>
<td>0.083</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>$A_6$</td>
<td>0.083</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.7: The strings in the language generated by the grammar in the running example for several values of the cut-point and of the $\alpha$ parameter.

point increases. Also, although $A_3$ was never the first state in a navigation session both languages include strings starting with it. This occurs because $A_3$ has a high frequency of traversal and the value of the parameter $\alpha = 0.5$ attaches a positive weight to its occurrence. The last two languages correspond to grammars inferred when attaching the same weight to all types of traversals, i.e., $\alpha = 1.0$. Therefore, rules starting at $A_2$, $A_3$ and $A_6$ are induced even though these states were never the first in a navigation session. Finally, when comparing the first and third languages we can see that the string which starts with $A_1$ is shorter in the second case while the strings starting with $A_2$ or $A_3$ are longer. This fact is explained by the relative weight given by the parameter $\alpha$ to the initial probabilities of the states.

For the sake of completeness we now give the definition of an induced grammar which is the non-probabilistic grammar that generates the strings of a HPG with a cut-point.

**Definition 3.9 (Induced grammar).** Let $HG$ be a hypertext grammar then $\varphi(HG) = \langle V, \Sigma, S, P' \rangle$ is the induced grammar, where $P' = P \cup \{ A_i \rightarrow \epsilon | A_i \in V - \{ S, F \} \}$.

Note that the productions of type $A_i \rightarrow \epsilon, A_i \in V - \{ S, F \}$ are not included in the definition of a hypertext grammar, see Definition 3.2, because it would result in an ambiguous grammar.
3.2 HPG as a Markov Chain

Given the proposed definition, a HPG can alternatively be seen as a finite and discrete Markov chain, (Kemeny and Snell, 1960; Feller, 1968). In fact, each production rule produces at most one new nonterminal in a string being derived and, at any stage, a derivation contains at most one nonterminal. Therefore, the derivation of the strings in a HPG can be modelled by a Markov chain and this analogy is useful to the computation of statistical properties of the language such as its entropy; see Section 3.6.

A Markov chain is fully characterised by a set of states \( X = \{x_1, x_2, \ldots, x_n\} \), a square transition matrix, \( T \), of order \( n \) that gives the one-step transition probabilities between states, and a row vector with the states' initial probabilities, \( \pi = \{\pi_1, \pi_2, \ldots, \pi_n\} \).

In order to convert a HPG to the corresponding Markov chain we will replace the production \( F \rightarrow \epsilon \) by the production \( F \rightarrow S \). This new production is given two alternatives regarding its probability. In the first alternative we set the production probability to zero, \( p(F \rightarrow S) = 0 \), and as a result we have an absorbing Markov chain; in the second alternative we set the production probability to one, \( p(F \rightarrow S) = 1 \), and as a result we have an irreducible Markov chain.

The conversion method for the case where \( p(F \rightarrow S) = 0 \) now follows.

**Definition 3.10 (HPG to an absorbing MC).** A HPG can be converted into an equivalent absorbing Markov chain by applying the following transformation:

1. \( X = \{x_1, \ldots, x_{|V|-1}\} = V - \{S\} \) is the finite set of states, \( |X| = |V| - 1 \).
   
   The state \( x_{|X|} \) corresponds to the nonterminal \( F \) which is the absorbing state.

2. \( \pi_i = p(S \rightarrow a_iA_i) \) for \( 1 \leq i < |X| \) and \( \pi_i = 0 \) for \( i = |X| \).

3. The transition matrix \( T \) is defined as follows:
   
   (i) for \( 1 \leq i, j < |X| \) we have \( T_{i,j} = p(A_i \rightarrow a_jA_j) \);
   
   (ii) for \( 1 \leq i < |X| \) we have \( T_{i,|X|} = p(A_i \rightarrow F) \) and \( T_{|X|,i} = 0 \);
   
   (iii) \( T_{|X|,|X|} = 1 \).

Follows a proof that the conversion method given in Definition 3.10 results always in an absorbing Markov chain.
3.2. HPG as a Markov Chain

Figure 3.8: An example of a HPG.

\[
X = \{A_1, A_2, F\}, \quad \pi = \begin{bmatrix} 0.3 & 0.7 & 0.0 \end{bmatrix}, \quad T = \begin{bmatrix} 0 & 0.5 & 0.5 \\ 0.6 & 0.0 & 0.4 \\ 0.0 & 0.0 & 1 \end{bmatrix}
\]

Figure 3.9: The absorbing Markov chain corresponding to the HPG in Figure 3.8.

Proof. The fact that the transformation proposed in Definition 3.10 results in an absorbing Markov chain follows from Proposition 3.7, which states that every state \( A_i \in V - \{S, F\} \) is included in at least one path from \( S \) to \( F \). Since the state \( F \) is an absorbing state which is reachable from every state, the chain is an absorbing chain. \(\square\)

Figure 3.8 gives an example of a HPG and Figure 3.9 the corresponding absorbing Markov chain.

The conversion method for the case where \( p(F \rightarrow S) = 1 \) now follows.

Definition 3.11 (HPG to an irreducible MC). When \( \alpha > 0 \), every HPG has an equivalent irreducible and aperiodic Markov chain which can be obtained by the following transformation:

1. \( X = \{x_1, \ldots, x_{|V|-2}\} = V - \{S, F\} \) is the finite set of states, \( |X| = |V| - 2 \).
2. \( \pi_i = p(S \rightarrow a_i A_i) \) for \( 1 \leq i \leq |X| \).
3. The transition matrix \( T \) is defined as follows, for \( 1 \leq i, j \leq |X| \) we have
\[
T_{i,j} = p(A_i \rightarrow a_j A_j) + \left(p(A_i \rightarrow F) \cdot p(S \rightarrow a_j A_j)\right).
\]
3.2. HPG as a Markov Chain

\[ X = \{A_1, A_2\}, \quad \pi = \begin{bmatrix} 0.3 & 0.7 \end{bmatrix}, \quad T = \begin{bmatrix} 0.15 & 0.85 \\ 0.72 & 0.28 \end{bmatrix} \]

Figure 3.10: The irreducible Markov chain corresponding to the HPG in Figure 3.8.

Figure 3.10 gives the Markov chain obtained by applying the conversion method in Definition 3.11 to the example of Figure 3.8. For example,

\[ T_{1,1} = p(A_1 \rightarrow F) \cdot p(S \rightarrow a_1 A_1) = 0.5 \cdot 0.3 = 0.15 \text{ and} \]
\[ T_{2,1} = p(A_2 \rightarrow a_1 A_1) + \left( p(A_2 \rightarrow F) \cdot p(S \rightarrow a_1 A_1) \right) = 0.6 + 0.4 \cdot 0.3 = 0.72. \]

A proof that the transformation in Definition 3.11 results in an irreducible Markov chain now follows. (A note about the notation: \( T \) is a square matrix, denotes \( T \) multiplied by itself \( x \) times, and \( T_{i,j}^{(x)} \) denotes the probability of going from \( i \) to \( j \) in \( x \) steps.)

**Proof.** From Proposition 3.7 it follows that \( \forall A_i \in V - \{F\}, \exists x > 0 : T_{A_i,F}^{(x)} > 0 \) and \( \forall A_i \in V - \{S\}, \exists y > 0 : T_{S,A_i}^{(y)} > 0. \) If we let \( T_{F,S}^{(1)} = 1 \) then \( T_{A_i,S}^{(x+1)} > 0, \forall A_i \in V - \{F\}. \) Since \( T_{S,A_i}^{(1)} > 0, \forall A_i \in V - \{F\} \) then \( \exists z > 0 : T_{A_i,A_j}^{(z)} > 0, \forall A_i, A_j \in V, \) which means that the chain is irreducible. \( \square \)

And now we prove that the chain obtained by Definition 3.11 is also aperiodic.

**Proof.** A chain is periodic when the number of transitions required on leaving any state to return to that same state is a multiple of some integer \( p > 1, \) where \( p \) is the chain period. Moreover, if a chain is irreducible there is always a path of length \( l \) which takes the chain from a state to any other state. Therefore, if from every state it is possible to reach a state such that \( T_{i,i}^{(1)} > 0 \) then the chain is aperiodic, because for every path of length \( l \) there is always a path of length \( l + 1 \) which includes the transition \( T_{i,i}^{(1)} > 0. \)

From Definition 3.11 it follows that \( \forall A_i \) such that \( p(A_i \rightarrow F) > 0 \) we have \( T_{i,i} > 0, \) and from Definition 3.6 it follows that there is at least one such state. Thus, since the chain is irreducible it is also aperiodic. \( \square \)

Finally, we should note that when \( \alpha = 0, \) there are some HPGs which don't have an equivalent irreducible and aperiodic chain, as it is shown by the example of Figure 3.11.
3.3. The Ngram Model

The HPG model makes the assumption that the choice of the page to browse next depends only on the current page on the user's browser and therefore, a HPG corresponds to a Markov chain, see Section 3.2. It follows that there can be strings in the language induced by the grammar which correspond to trails that were not traversed by a user. Although this assumption is somewhat controversial, we note that it is severely limiting to model the exact sessions since the probability of long trails being followed several times in exactly the same way is low. Moreover, when a user is inspecting a web page and deciding which link to follow next only few, if any, of all the previously visited pages affect his decision. Consider, for example, a user browsing an online supermarket and having to decide which link to follow next while inspecting a page with special offers. His decision will probably depend more on the quality of the offers themselves than on all the pages visited beforehand. On the other hand, the probability of choosing a link is not completely independent of the browsing history. Therefore, a model that blends these two modes of behaviour provides a better representation of the user's navigation history. Thus, we make use of the Ngram concept, (Chamiak, 1996), to improve the model accuracy in representing the user sessions. In our context the Ngram concept is used to determine the assumed user memory when navigating the web. A Ngram model considers that only the previous $N - 1$ pages visited have a direct effect.

Figure 3.11: An example of a HPG whose equivalent irreducible Markov chain (for $\alpha = 0$) is periodic.
on the probability of the next page to browse. In other words, we assume that the choice of the page to browse next does not depend on all the pages previously seen but only on the last $N - 1$. We call $N - 1$ the history depth of a HPG.

From a collection of user navigation sessions the relative frequency with which each page was requested determines its zero-order probability, that is, the probability of a page being chosen independently of any previously visited pages. A onegram model is inferred by computing such probabilities for every page. In the example of Figure 3.2 page $A_1$ was visited 4 times in a total of 24 page requests, therefore, its zero-order probability is $p(A_1) = 4/24$.

Similarly, the first-order probabilities are estimated by computing the relative frequencies of all the bigrams in the collection of sessions. A bigram is no more than a pair of consecutive requests. In this model, the probability of the next choice depends only on the current position and is given by the frequency of the bigram divided by the overall frequency of all bigrams with the same initial position. In the example of Figure 3.2 $A_2A_3$ has a frequency of 3 and the only other bigram with $A_2$ in its initial position is $A_2A_4$ which has a frequency of 1; therefore, $p(A_2A_3) = 3/(3 + 1) = 3/4$.

The second-order model is estimated by computing the relative frequencies of all trigrams and higher orders can be computed in a similar way.

In a $N$ gram model each state corresponds to a navigation trail having $N - 1$ pages. Therefore, the grammar does not generate strings, $w$, of length shorter than the history depth, $|w| < N - 1$; similarly, only those trails whose length is greater than $N$ can be used in the grammar's construction.

The drawback of the $N$ gram model is the exponential increase in the number of its states as the history depth increases. In a language whose alphabet has $l$ symbols the $N$ gram model can lead to a grammar with $l^N$ states. However, in the case of HPGs the number of states is limited by the web topology and the worst case can only occur if the reachability relation corresponds to a complete directed graph, and that is rarely the case on the web. In fact, for a web site having $n$ pages and an average of $BF$ out-links per page ($BF$ stands for branching factor) the use of the $N$ gram is expected to result in $n \cdot BF^{N-1}$ states. In addition, the user navigation sessions are short on average,
3.3. The Ngram Model

see (Huberman et al., 1998), and \( N \) should be less or equal to the average session length. Otherwise too many sessions have to be discarded. In conclusion, there is a trade-off between the size of the user memory modelled (measured by the history depth) and the grammar complexity (measured by the number of states).

Figure 3.12 shows the trigram HWG for the collection of trails given in Figure 3.2. For example, the weight of production \( A_1A_5 \rightarrow A_5A_3\) is given by the number of times the trigram \( A_1A_5A_3\) occurs in the collection of sessions. A HPG is obtained by the normalisation of the HWG corresponding to the \( N \)gram, without any loss of the model properties.

One question that arises when making use of the \( N \)gram model is whether or not it is possible to determine the value of \( N \) which is more appropriate to model a given collection of user navigation trails. One answer for this question is to make use of a \( \chi^2 \) (chi-square) test to compare the probabilities given by consecutive grams. This test can be used to compare the predictive power of two models with different order. For example, in order to compare the onegram with the bigram model we calculate the expected frequencies of the bigrams while assuming independence (i.e. while assuming a onegram model) and the results are compared with the observed frequencies. If the difference is not significant we opt for the lower order model since it is more compact for an equivalent accuracy. We now describe the \( \chi^2 \) test following (Chatfield, 1973)
3.3. The N-gram Model

and (Attneave, 1959), noting that in order for the test to be suitable to our context we will need to modify the way the number of degrees of freedom is set.

Let $ABABAABAB$ be a user session in which $A$ and $B$ occur $N_A = 5$ and $N_B = 4$ times respectively, in a total of 9 one-grams, $N_1 = 9$. If we assume independence the probability of page $A$ being visited is $p(A) = N_A/N_1 = 5/(5+4)$ and the probability of $B$ is $p(B) = N_B/N_1 = 4/(5+4)$. From these probabilities we can infer the expected frequency of a given bigram when independence is assumed. For example, the expected frequency of $AA$ is given by

$$\hat{e}_{AA} = N_A \cdot p(A) = 5 \cdot (5/9) = 2.78;$$

where $N_A$ is the number of $A$s in the original session and $p(A)$ the probability of an $A$ occur in a session. Since we are assuming independence, the probability of two consecutive $A$s in the session is $N_A \cdot p(A)$. Note that the observed number of $AA$s is 1. If we calculate the expected and observed frequency for all possible bigrams we can, by the means of a $\chi^2$ test, assess how close the two models are and therefore, assess if it worth to increase the model complexity to the next level for the consequent gain in accuracy.

One difficulty arises from the fact that there are, in general, less bigrams than one-grams and trigrams than bigrams, unless we have a single circular session. This problem is more significant if we have many different sessions. Because of this variable sample space size, the test is more accurate if the marginal frequencies are used to calculate the frequencies for the lower order model. For example, in the session of the example we have the following bigram frequencies $AA : 1, AB : 4, BA : 3, BB : 0$, and a total of $N_2 = 8$ bigrams. The expected frequency of $AA$ assuming independence and using the marginal frequencies to compute the probabilities is

$$\hat{e}_{AA} = N_{A.} \cdot p(A) = N_{A.} \cdot (N_{A.}/N_2),$$

where $N_{A.}$ represents the number of bigrams with $A$ in the first position and $N_{A.}$ the number of bigrams with $A$ in the second position. Using a similar notation, the expected frequency of a trigram when a first order model is assumed is estimated by

$$\hat{e}_{ijk} = N_{ij.} \cdot p(k/j) = N_{ij.} \cdot (N_{jk.}/N_{j.}),$$

where $N_{j.}$ represents the number of trigrams with symbol $j$ in the second position.
3.3. The N gram Model

Having the estimated, \( \hat{\alpha}_{ijk} \), and the observed, \( \alpha_{ijk} \), frequencies for all the grams we can perform a standard \( \chi^2 \) test, (Wonnacott and Wonnacott, 1990), where the overall measure of deviation is given by:

\[
\chi^2 = \sum_{ijk} \frac{(\alpha_{ijk} - \hat{\alpha}_{ijk})^2}{\hat{\alpha}_{ijk}}.
\]

The next step is to determine the number of degrees of freedom, d.f., in order to compare the experimental value, obtained from the data, with the critical value, obtained from the theoretical distribution. It should be noted that a web topology does not correspond to a complete graph, therefore, the usual definition for the degrees of freedom does not apply in this case. In a standard \( \chi^2 \) test the number of d.f. is given by the number of terms in the overall measure of deviation, for example, when comparing the bigram model with the trigram model we would have d.f. = \( N_0^3 - 1 \), where \( N_0 \) is the number of different symbols in the session.

For our test we estimate the number of d.f. in the following way. For a given \( N \)-gram the first \( N - 1 \) symbols correspond to a state in the HPG. Having all the \( N \)-grams with expected frequency greater than zero, we group them by the first \( N - 1 \) symbols, and for each group we count the number of elements and subtract one, that gives the number of d.f. per state. The overall number of d.f. is obtained by the d.f. sum over all states. The reasoning behind this estimation is that the frequencies of the traversals from a state are not independent. In fact, since we know the number of visits to a state, we can express the number of traversals of one of its out-links in terms of the others out-links. In addition, we need to ignore comparisons which correspond to inadmissible transitions. For example, a transition from \( AB \) to \( BC \) can only exist if there is an \( i \) and a \( j \) such that \( N_{iBC} > 0 \) and \( N_{ABj} > 0 \). Otherwise the expected value of \( N_{ABC} \) is zero, which we interpret as the non-existence of a hypertext link for such traversal.

Once we have obtained the number of d.f. the statistical test can be performed by comparing the experimental value with the critical value. If the first is greater than the second the hypothesis that the two models are similar is rejected. In such event, the gain in accuracy when opting for the higher order model is worth the cost of the consequent increase in the model complexity. In this case, a new test should be performed for a the
3.3. The N gram Model

### Marginal frequencies

<table>
<thead>
<tr>
<th>trigram $(ijk)$</th>
<th>$\hat{a}_{ijk}$</th>
<th>$a_{ijk}$</th>
<th>$\frac{(a_{ijk} - \hat{a}<em>{ijk})^2}{\hat{a}</em>{ijk}}$</th>
<th>$\chi^2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AAA</td>
<td>$\frac{1}{4}$</td>
<td>0</td>
<td>0.25</td>
<td></td>
</tr>
<tr>
<td>AAB</td>
<td>$\frac{3}{4}$</td>
<td>1</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>ABA</td>
<td>$\frac{3}{3}$</td>
<td>3</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>ABB</td>
<td>$\frac{0}{3}$</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BAA</td>
<td>$\frac{1}{4}$</td>
<td>1</td>
<td>0.08</td>
<td></td>
</tr>
<tr>
<td>BAB</td>
<td>$\frac{3}{4}$</td>
<td>2</td>
<td>0.03</td>
<td>0.44</td>
</tr>
<tr>
<td>BBA</td>
<td>$\frac{0}{4}$</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>BBB</td>
<td>$\frac{0}{4}$</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

$$\chi^2 = 0.44$$

Figure 3.13: An example of the computation of the $\chi^2$ test.

next higher order of the model. Note that the test results should be taken as an indication of the required order for the model because the accuracy of the $\chi^2$ approximation depends on whether or not there is sufficient data available to characterise the various grams.

When the number of $d.f.$ is large, i.e. $d.f. > 30$, $\sqrt{2\chi^2}$ can be approximated by a normal distribution with mean $\sqrt{2\cdot d.f. - 1}$ and unit standard deviation, (Cramér, 1946).

Figure 3.13 exemplifies the $\chi^2$ test for a single trail $ABABAABAB$ when comparing the bigram with the trigram model. In this case we cannot reject the hypothesis that the two models are different since the experimental value, $\chi^2 = 0.44$, is much smaller than the critical value, $\chi^2_{0.05,2} = 5.99$, obtained from $\chi^2$ distribution statistical tables, (Wonnacott and Wonnacott, 1990). Therefore, the additional complexity of a higher order model does not give the necessary return in terms of improved accuracy. Note that the trigrams with null expected value are not taken into consideration for the computation of the number of degrees of freedom.

An alternative way of assessing the required model complexity is by making use of the conditional entropy concept, (Chatfield, 1973). The conditional entropy of a random variable, $X$, given another random variable, $Y$, corresponds to the average uncertainty of the conditional distribution, $H(Y/X) = \sum p(x, y) \log p(y|x)$. When com-
3.3. The Ngram Model

Comparing the probability distribution of the onegram model to that of the bigram model, the conditional entropy gives the difference between the average uncertainty of the two distributions. That difference can be interpreted as the average amount of new information given by the second symbol of a bigram relatively to the information contained in the first symbol of the same bigram. Therefore, we estimate the conditional entropy for successive orders of the HPG in order to assess whether or not an increase in the model’s order gives a significant decrease in the conditional uncertainty. We note again that, when comparing models with two consecutive orders, the marginal frequencies should be used to estimate the probabilities of the lower order model, for example $p(y/x) = \frac{N_{xy}}{N_x}$.

In (Chatfield, 1973) a plot and a hypothesis test is proposed to assess if the decrease in conditional uncertainty when increasing the model’s order is significant. The proposed test is an approximation of the $\chi^2$ test described above, and as such it will not be presented here. However, the plot is useful as an auxiliary visual tool to determine the model’s order. The definition of the plot now follows.

1. For $N = 0$ we have the case where the data is not analysed and all the outcomes are given the same probability. In this case we have the maximum uncertainty which is given by $H_0 = \log(N_0)$, where $N_0$ is the number of symbols;

2. For $N = 1$ we have a model were independence is assumed, in this case $H_1 = -\sum_i (p(i) \log p(i))$;

3. For $N = 2$ we have $H_2 = -\sum_{i,j} p(i,j) \log p(y/x)$ and similarly for higher orders.

In Figure 3.14 we show the entropy plot for an example sequence of 96 occurrences of two different symbols, $A$ and $B$. In this case there is a clear drop from the $H_1$ to $H_2$ which indicates that the biggest gain in accuracy occurs when we move from a model assuming independence to a first-order model. Comparatively, the conditional uncertainty after one symbol is known is relatively small.

As a final remark to the Ngram model, we note that it is possible to borrow the smoothing technique concept from the speech recognition field, (Charniak, 1996). This
3.4 Incremental Model

One important characteristic of the HPG model is being incremental and providing a compact way of storing the information contained in log files. In fact, the information contained in a new log file can be incorporated into a HWG without having to access the previously incorporated files. Moreover, a HWG provides a compact representation of
the log data since a trail which was traversed several times is represented only once in a HWG. A HPG is obtained from the HWG normalisation. We note that the incremental property only holds when the analyst is working with a constant history depth since, when using an N-gram, it is not possible to accurately transform a HWG with a given $N$ into a HWG with a different history depth.

**Algorithm 3.1:** The pseudo-code for the incremental algorithm to infer a HWG from a collection of trails.

In Algorithm 3.1 we present the pseudo-code for the method used to incrementally build a HWG. We let $T$ be the collection of user navigation sessions and $|T|$ its cardinality, $T_i$ be the $i^{th}$ session in the collection and $T_i[j]$ be the $j^{th}$ page of session $i$. We also let $N - 1$ be the history depth of the grammar, and $StateSet$ be the set of grammar states; the method $StateSet.update()$ updates the $StateSet$ with the last inferred state, either by creating a new state or by updating the statistics of an existing state. A $State$ corresponds to $N - 1$ consecutive page requests $State[1], \ldots, State[N - 1]$; the method $State.shiftLeft()$ performs the assignment $State[i] = State[i + 1]$, for $i = 1, \ldots, N - 2$. In addition, we let $Productions$ be the set of all grammar productions and the method $Productions.update()$ updates the set with the last inferred production, either by creating a new production or by updating the statistics of an existing pro-
3.5 Exhaustive Computation of All Rules

We now present a data mining algorithm aimed at inducing a grammar's rule-set for a given cut-point, that is, inducing the set of maximal strings in the language. This can be achieved by performing a generalisation of one of the well known graph exploration methods such as a Depth-First Search (DFS), (Tarjan, 1972), or a Breadth-First Search (BFS), (Weiss, 1993). In practice we have noticed that although the DFS is conceptually simpler it consumes more memory due to its recursive nature, especially when the cut-point is set with a low value and the depth of exploration is large. Therefore, we have adopted a modified BFS algorithm as the standard method for the exhaustive computation of all the rules of a grammar.

The BFS algorithm works by constructing a tree with the start state \( S \) as its root, wherein each branch of the tree is explored until its probability falls below the cut-point. A branch of the tree having probability above the cut-point is called a candidate trail (or a candidate rule) if it still has some expansions to be evaluated. A branch with probability above the cut-point and with no more expansions to be evaluated is called a rule (or a maximal trail). We note that since a HPG does not contain any cycles having probability one there is the guarantee that the algorithm will terminate.

Figure 3.15 shows the exploration tree for the HPG inferred from the collection of trails in the running example given in Figure 3.2, with \( \alpha = 0 \), and for the cut-point \( \lambda = 0.1 \). The dashed lines correspond to links whose inclusion in a trail being explored would lead to a trail with probability below the cut-point. Such links are inadmissible expansions of the trails. The probability given next to a link corresponds to the probability one there is the guarantee that the algorithm will terminate.

In Algorithm 3.2 we give the pseudo-code for the BFS algorithm. We let \( \lambda \) be the grammar's cut-point and \( n \) be the number of states. We let \( CT \) and \( CT_i \) be two different sets of candidate trails and \( RS \) be a set of rules. The method \text{CT.empty()} checks if a set of candidate trails is empty and the method \text{X.push()} inserts a new trail into a trail-set, where \( X \in \{CT, CT_i, RS\} \). We let a trail be represented by \( wA_i \).
where \( w \) represents its set of terminal symbols and \( A_i \) its unique nonterminal symbol; \( P_{i,j}, 1 \leq j \leq P_i \), represents the set of productions whose left-hand side is \( A_i \) and \( P_i \), its cardinality.

We now give the average case complexity analysis for the BFS Mining algorithm. We measure the algorithm's complexity by the number of iterations where an iteration is defined as a link evaluation in the exploration tree.

Consider a HPG with \( n \) states, \( l \) links and cut-point \( \lambda \). The grammar's branching factor is defined as \( BF = l/n \) and the length of a trail is given by the number of terminal symbols it contains. When ignoring the cut-point the expected number of grammar's trails having length \( m \), \( \#w_m \), is given by the following expression:

\[
E(\#w_m) = n \cdot BF^{(m-1)} = n \left( \frac{l}{n} \right)^{(m-1)}.
\]

Moreover, when \( \alpha > 0 \), and every state has positive initial probability, the average probability of a production from the start state is \( 1/n \) and the average probability of a production from a state corresponding to a page, \( A_i \in V - \{S, F\} \), is \( 1/BF = n/l \).
Algorithm 3.2: The pseudo-code for the Breadth-first search algorithm.

Therefore, the average probability of a trail with length \( m \), \( p(w_m) \), is

\[
E(p(w_m)) = \frac{1}{n} \left( \frac{n}{l} \right)^{(m-1)}.
\]

Given a cut-point, \( \lambda \), the expected average trail length, \( \Delta \), can be estimated by solving the equation

\[
E(p(w_\Delta)) = \lambda \equiv \frac{1}{n} \left( \frac{n}{l} \right)^{(\Delta-1)} = \lambda \equiv (\Delta - 1) \ln \left( \frac{n}{l} \right) = \ln(\lambda n) \equiv \Delta = \frac{\ln(\lambda n)}{\ln \left( \frac{n}{l} \right)} + 1.
\]

If \( \lambda = \theta \cdot \delta = (1/n)\delta \) we have

\[
\Delta = \frac{\ln(\lambda n)}{\ln \left( \frac{n}{l} \right)} + 1 = \frac{\ln(\delta)}{\ln \left( \frac{n}{l} \right)} + 1 = \frac{\ln(\delta)}{-\ln(BF)} + 1. \tag{3.1}
\]
This last expression shows that for a given confidence threshold and branching factor, \( BF = l/n \), the expected average rule length is constant and independent of the number of states. Note that the support threshold was considered to be proportional to the number of states, as recommended in Section 3.1.

Moreover, for a given \( \delta \) the expected number of trails is

\[
E(\#w_\Delta) = n \left( \frac{l}{n} \right)^{(\Delta - 1)} = n \left( \frac{l}{n} \right)^{\ln(\delta)/\ln(n/l)} . \tag{3.2}
\]

And using the expression \( a^b = e^{b \ln(a)} \) Equation 3.2 can be rewritten as

\[
E(\#w_\Delta) = n \left( \frac{l}{n} \right)^{\ln(\delta)/\ln(n/l)} = n \left( BF \right)^{\ln(\delta)/\ln(l/n)} = n \left( e^{-\ln(BF)} \right)^{\ln(\delta)} = n \cdot e^{-\ln(\delta)} = \frac{n}{\delta} . \tag{3.3}
\]

Finally, the expected number of iterations, \( \text{Iter} \), is

\[
E(\text{Iter}) = E(\#w_\Delta) \cdot (\Delta + \frac{l}{n}) = \frac{n}{\delta} \cdot \left( \frac{\ln(\delta)}{\ln\left( \frac{n}{l} \right)} + 1 + \frac{l}{n} \right) . \tag{3.4}
\]

In the last expression \( (\Delta + l/n) \) represents the average number of links it is necessary to check in order to obtain a rule. The term \( l/n \) is justified because all out-links from the last state in the trail have to be evaluated in order to assess if they correspond to inadmissible expansions of the trail.

We will now state the result of Equation 3.4 as a theorem.

**Theorem 3.12.** Given a fixed support, \( \theta = O(1/n) \), and confidence, \( \delta \), thresholds the average number of iterations needed to find all the grammar's strings having probability above the cut-point, \( \lambda = \theta \delta \), varies linearly with the number of states in the HPG.

Finally, we note that the intuition behind Equation 3.3 is that \( 1/\delta \) gives an upper bound to the number of trails we can pack into \( \delta \).
3.6 The Entropy of a HPG

The entropy measures the uncertainty in the outcome of a random variable, $Y$. If $H(Y) = 0$ it means that the outcome is certain; if $H(Y) = \log(n)$ it means that we have complete uncertainty and all the $n$ outcomes are equally probable. In a HPG the sample space is the set of all generated strings and, if there is no information about the user interaction with the web, the model which makes less assumptions about the user behaviour assigns an equal probability of being visited to all pages and the same probability of being chosen to all links. The entropy is maximum in this case and there is no point in looking for patterns since every trail is equally likely. On the other hand, if the entropy is close to zero it means that there is a small set of outcomes with high probability. In a HPG it means that there is a small set of trails with high probability and therefore, a small set of rules should contain enough information to characterise the user navigation behaviour. The intuition behind a grammar's entropy is that if its value is close to zero there should be a small set of strings with high probability. On the other hand, if the entropy is high there should be a large number of strings with similar and low probability.

In (Soule, 1974) the author calls the entropy of a probabilistic grammar the sentential entropy since the sample space is defined as the set of all grammar sentences (or strings). Follows the definition of sentential entropy according to (Soule, 1974) where $G_p$ represents a probabilistic grammar, $\mathcal{L}(G_p)$ the language generated by the grammar, $w$ a string in the language, and $p(w)$ the probability of the string $w$.

Definition 3.13 (Grammar entropy). The entropy of a probabilistic grammar, $H(G_p)$, is defined as

$$H(G_p) = -\sum_{w \in \mathcal{L}(G_p)} p(w) \log p(w).$$

The analogy between a HPG and a finite Markov chain, see Section 3.2, is useful for the computation of the grammar's sentential entropy. It is shown in (Soule, 1974) that if a grammar is unambiguous the derivational entropy is equal to the sentential entropy. The derivational entropy is defined as the entropy of a random variable where the sample space is the set of all string derivations in the grammar. According to (Soule, 1974) when a probabilistic grammar is viewed as a Markov chain, characterised by the
transition matrix $T$ and with a unique absorbing state, the derivational entropy is defined as the vector

$$H(G_p) = (I - T)^{-1} \eta,$$

where $I$ is the identity matrix, $\eta$ is a vector with the same order as $T$ having $\eta_i = H(A_i)$ defined as the entropy of the state $A_i$. The *entropy of a state* is defined as the entropy of the choice among all the productions which have nonterminal $A_i$ in its left-hand side. For example, the entropy of state $A_1$ in the grammar of Figure 3.8 is

$$H(A_1) = - \left( p(A_1 A_2) \log p(A_1 A_2) + p(A_1 F) \log p(A_1 F) \right).$$

In (Cover and Thomas, 1991) the *entropy rate* of an irreducible Markov chain with a stationary distribution vector $\mu$ is defined as

$$H(G_p) = - \sum_{ij} \mu_i T_{ij} \log T_{ij}.$$ 

In this thesis we consider two modified versions of the above definitions for the entropy of a HPG. The first definition relates to the case where the HPG has $P(F \rightarrow S) = 0$ and the grammar corresponds to an absorbing Markov chain. In this case we defined the grammar’s *Soule entropy* as

$$H_s(HPG) = H(\pi) + \pi (I - T)^{-1} \eta,$$

where the transition matrix $T$ is that obtained by Definition 3.10, having both the row and column corresponding to state $F$ eliminated. As usual, $\pi$ is the vector of the initial probabilities of the Markov chain. This definition corresponds to the entropy vector defined by Soule, averaged by the vector of initial probabilities plus the entropy of vector $\pi$. We include $H(\pi)$ in the definition in order to take into account the randomness relative to the choice of the initial page. In addition, we average the vector which results from Soule’s definition in order to obtain a single value as the HPG’s entropy.

The second definition corresponds to the case where $p(F \rightarrow S) = 1$. In this case we define the grammar’s entropy rate as

$$H_r(HPG) = H(\pi) + \sum_{ij} \pi_i T_{ij} \log T_{ij},$$
where the transition matrix $T$ is the same transition matrix as the one used in the $H_s$ definition. This definition corresponds to the Cover definition plus the entropy of vector $\pi$ which is taken as an estimator of $\mu$. Note that $\pi$ is proportional to the number of times each state was visited. Again, we include $H(\pi)$ in the definition in order to take into account the randomness relative to the choice of the initial page.

Alternatively, in the latter definition we could have considered the transition matrix $T$ to be the one obtained by the transformation method given in Definition 3.11, that is, the equivalent irreducible Markov chain. However, we have decided to use the original matrix since it is this matrix that is used by our data mining algorithm for the computation of the grammar's strings.

In Figure 3.16 we give an example of two different HPGs which have the same entropy rate if we don’t include the entropy of the initial probabilities vector in the definition. In Figure 3.16 (a) we have $H_r = \log(2) + 0.5 \log(2) + 0.5 \log(2)$ and in Figure 3.16 (b) we have $H_r = \log(1) + \log(2)$.

![Figure 3.16: Two HPGs to exemplify the entropy rate definition.](image)

### 3.7 Summary

In this chapter we have presented the formal definition of the hypertext probabilistic grammar (HPG) model. The HPG model aims at capturing user web navigation patterns. We gave two methods to transform a HPG into a Markov chain and we proved
that a HPG can be viewed as an irreducible and aperiodic Markov chain. Moreover, we proposed the use of the $N$-gram concept to model the assumed user's memory when browsing the web. The chi-square statistical test is proposed as a method to assess the order of the $N$-gram model that gives the best trade-off between the model size and its accuracy in representing the user sessions. We gave an algorithm to incrementally infer a HPG from log data and a breadth-first search (BFS) algorithm to compute the set of grammar strings having probability above a given cut-point. We proved that the BFS algorithm is, on average, linear in the variation of the number of iterations with the number of states in the grammar. Finally, we proposed the use of entropy as a measure of the statistical properties of a HPG's language.
Chapter 4

Experimental Evaluation

In this chapter we present the experiments conducted to assess the performance and effectiveness of the proposed HPG model. Experiments with both synthetic and real data were performed in order to evaluate both the model's behaviour with several different configurations and its potential usefulness in practice.

All the software was implemented with the Java programming language, version 1.2, on a Unix platform running Sun Solaris 2.5.1. We used the database management system Oracle 7.3.4 to store the data; JDBC was used to connect the Java programs with the database. In the experiments all algorithms were run in main memory only and we were limited to 40Mb of RAM.

4.1 Synthetic Data Experiments

Experiments with synthetic data provide the means for evaluating many different topologies and configurations of the HPG model. The motivation behind this set of experiments was to confirm the theoretical analysis of the algorithm and to obtain a general picture of the model's sensitivity to various parameter configurations.

4.1.1 Synthetic Data Generation Method

The synthetic data generation method used in this set of experiments consisted in randomly generating hypertext weighted grammars (HWGs). We have chosen to create HWGs instead of log files because the generation of random log files implies the previous creation of a web topology. And, having generated a web topology it is more efficient to generate the weights of the links (in order to obtain a HWG) than to gener-
ate log files and infer a HWG from them. The corresponding HPG can then be obtained by the HWG normalisation.

To randomly generate a HWG we need to specify the number of states, the probability distribution for the number of out-links per state, and the probability distribution for the links' weight. Having these parameters specified, the method used to create a HPG consists of four consecutive steps:

1. Randomly create a directed graph given the required number of states and the probability distribution for the number of out-links per state.
2. For each state in the grammar assign a weight to each out-link according to the chosen probability distribution.
3. Verify if the resulting grammar is reduced, that is, if every state is included in a path from the state \( S \) to the state \( F \). If that is not the case, add an out-link to \( F \) on every state not included in a path to \( F \). (Note that all the synthetic data experiments were conducted with \( \alpha = 1 \) meaning that every state has a path from \( S \).) At this stage we have a valid HWG.
4. Normalise the HWG, that is, calculate the production probabilities in order to obtain the corresponding HPG.

To characterise both the HPG's branching factor and the weight of each link we have adopted the Poisson distribution, (Feller, 1968). The decision to adopt such distribution was based on its universality as a distribution that models a great variety of phenomena which are characterised by the number of events occurring in a period of time. In our case the events are the number of out-links from a page or the number of user traversals of a link. By comparing the synthetic data experiment's results with the real data results we will be able to assess if the adopted synthetic data generation model is reasonable.

Recent research suggests that the number of in-links per page follows a Zipf distribution and the number of out-links per page roughly follows the same distribution, (Kumar et al., 1999). In (Broder et al., 2000) these results were confirmed especially for the distribution of the in-links. According to this last study, the number of out-links per page exhibits a Zipf distribution; however, the results suggest that pages with a small
number of out-links follow a different, possibly Poisson, distribution. In (Kumar et al., 1999) the authors refer to the high probability of deviating significantly from the mean as being an important characteristic of a Zipf distribution. Note that a Poisson distribution also possesses this property.

### 4.1.2 Experimental Results

The experiments with synthetic data were conducted for grammars with size varying between 1000 and 20000 states (the upper limit was imposed by the amount of RAM available). The grammar's branching factor (indicated as BF in the figures) varied from an average of 3 out-links per page to an average of 9 out-links per page and the average links' weight was set to vary between 3 and 9. Moreover, the confidence threshold varied between 0.1 and 0.9 and the support threshold was fixed with the value $1/n$ for each grammar, where $n$ is the number of states. Note that we can vary the overall cut-point value within a given range by varying its confidence component; therefore, in the figures the cut-point is indicated by the value of the confidence threshold. Finally, for each configuration 30 runs were performed and the presented results correspond to the average of the 30 runs.

![Diagram](image)

**Figure 4.1:** The variation of the number of iterations and the running time with the number of states.

Figure 4.1 (a) shows the variation of the number of iterations with the grammar's number of states for several values of the cut-point and for a branching factor fixed at 3.
The results confirm the linearity indicated by Equation 3.4; note that we define an iteration as the evaluation of a link. Similarly, Figure 4.1 (b) shows the variation of the algorithm’s running time for the same parameters. In this plot the time corresponds to real running time, as opposed to CPU time, since with the Java programming language there is no clear way of measuring the CPU time. Moreover, when measuring the algorithm’s performance we did not take into account the time needed to store the induced rule-set in the database, since it would represent a significant part of the overall running time and would provide a negative bias to the configurations generating large rule-sets. The results in Figure 4.1 (b) show that the running time displays a variation close to linear with the number of states.

Figure 4.2 also reports results regarding the algorithm’s complexity. In Figure 4.2 (a) the confidence threshold is fixed at 0.3 and the variation of the number of iterations with the number of states is shown for different values of the branching factor (BF). It can be said that, in general, an increase in the branching factor leads to an increase in the number of iterations, in spite of the expectation of rules to be shorter. The results reported in Figure 4.2 (b) give another perspective over Equation 3.4, showing that for a fixed number of states and branching factor there is an exponential increase in the number of rules with the decrease of the confidence threshold.
4.1. Synthetic Data Experiments

Figure 4.3 confirms that in practice there is a proportionality between the number of iterations and the number of rules induced. In fact, both the number of rules and the number of iterations display a linear variation with the number of states for a given branching factor and confidence threshold.

![BFS (BF=3)](image)

Figure 4.3: The variation of the number of induced rules with the number of states.

The results reported in Figure 4.4 (a) correspond to a variation of the parameter of the Poisson distribution for the links' weight. It can be noted that the parameter of the distribution does not significantly influence the variation of the number of iterations with the number of states. In fact, when normalising a HWG to obtain a HPG only the relative values of the weights should make a difference in the probabilities obtained. Therefore, from now on whenever we refer to a randomly generated HPG we are using a Poisson distribution with average 3 for the links weights.

Figure 4.4 (b) gives the variation of the average rule length with the number of states. The results confirm Equation 3.1 which states that for a given branching factor the average rule length does not depend on the grammar's size. (Recall that the value for the support threshold was set in a way that takes into account the number of states.) It can also be seen in the figure that, in order to obtain longer rules the confidence has to be set with a low value.
4.2. Real Data Experiments

In this section we report the results of the experiments conducted with real data. We note that, real data provides the means for assessing the usefulness and performance of the HPG model in a real world scenario.

4.2.1 Real Log Files

The real log files used in the reported experiments were obtained from the authors of (Perkowitz and Etzioni, 2000) and downloaded from


These files contain two months of log access data from 1997 of the web site

http://machines.hyperreal.org/

According to the authors, the web site contains around 2500 pages and receives approximately 10000 requests per day from around 1200 users. The data was made anonymous and the caching disabled so that every page had to be requested even when being revisited.

We used the data without any special cleaning apart from eliminating the .gif and .jpg requests. We divided each month in four subsets, each corresponding to the page requests made in one week. One of the weeks was discarded because it presented char-
acteristics significantly different from the other seven weeks, namely in the number of states inferred which was much larger than the number given by the owners of the data in (Perkowitz and Etzioni, 2000). This last fact indicates that the data should be cleaned. However, since data cleaning is a complex process which was not in the scope of this thesis, together with the belief that the probabilistic nature of the model could partially overcome the dirtiness of the data (by, for example, ignoring misspelled requests), we have decided to use the data without further cleaning it (see Section 2.3.3.1 for a discussion on data cleaning).

4.2.2 Experimental Results

In this section we first report on the results regarding the algorithm’s performance with real data and its sensitivity to varying the values of the parameters. After that, we discuss the usefulness of the HPG model by making use of the induced rules to carry out a study of users’ navigation behaviour within the site.

Figure 4.5 (a) shows the variation of the number of iterations with the grammar’s number of states for two different values of the confidence threshold. We note that all points for $n > 5000$ correspond to runs of a $N$gram configuration since the web site to which the data belongs has around 2500 states. Since, in general, an increase in the value of $N$ leads to a decrease in the grammar’s branching factor, these results should be seen just as an indication of the model’s scalability. In fact, as we saw in Figure 4.2 (a), a decrease in the branching factor results in a decrease in the number of iterations. Also, while in the plots of Figure 4.5 each point represents a single run, in the plots of Figure 4.1 each point corresponds to the average result of 30 runs.

From the comparison of Figures 4.5 (a) and (b) we can confirm that there is a proportionality between the number of induced rules and the number of performed iterations. In fact, both measures present a close to linear relation with the grammar’s number of states.

Figure 4.6 (a) shows the average variation of the number of induced rules with the value of the confidence threshold for the seven weeks in the real data set; a point in the plot represents the average of the seven weeks’ results for a given value of the confidence. These results are similar to those obtained with synthetic data, which are showed
4.2. Real Data Experiments

Figure 4.5: The variation of the number of iterations and the number of rules with the number of states for the real data.

in Figure 4.2 (b). Also, the results are consistent with the behaviour suggested by Equation 3.3. To complement the analysis, Figure 4.6 (b) gives the variation of both the average rule length (ARL) and the maximum rule length (MRL) with the confidence, where a point in the plot represents the average of the seven weeks.

Figure 4.6: The variation of the number of induced rules and their length with the confidence threshold.

Following the results presented in Figure 4.5 and Figure 4.6 we can say that there is a similarity between the results obtained with real data and those obtained with the
4.2. Real Data Experiments

synthetic data. Thus, we are provided with strong evidence that the method used for the generation of the synthetic data is adequate for the intended purposes.

In Figure 4.7 we show the ten states with the highest initial probability for the HPGs inferred from the real log files with $N = 2$ and three different values of the $\alpha$ parameter. We note that these results correspond to the average probability of the seven weeks. Recall that, according to Definition 3.6, when $\alpha = 0$ the initial probability of a state is proportional to the number of sessions that have the corresponding page as the first page viewed. Also, for $\alpha = 1$ the initial probability of a state is proportional to the number of times the corresponding page was requested; $\alpha$ can take any value between 0 and 1 in a way that provides a balance between the two above scenarios.

Figure 4.7 (a) shows the ten pages with the highest initial probability when $\alpha = 0$. The reported probabilities reveal that more than 35% of the sessions started at the homepage, /, of the web site. The pages /Analogue-Heaven/ and /samples.html are also important starting points for users' navigation in the site. In fact, the above three pages are used as the navigation starting point in close to 50% of the sessions.

Figure 4.7 (b) shows the top-ten pages for $\alpha = 1$. In this case, the higher the probability the more popular the page is among the users since, when $\alpha = 1$, the initial probability of a state is proportional to the number of times the corresponding page was requested. It is interesting to note that the homepage is not only the preferred page to start the navigation, see Figure 4.7 (a), but also the most viewed page in the site. In contrast, the page /Analogue-Heaven/ is only ranked as the seventh most viewed in spite of being the second preferred as a navigation starting point. A close analysis of rules whose first page is /Analogue-Heaven/ reveals that the vast majority of such rules have as the second page the homepage, /. Two questions arise from this fact: Why do the users choose that page to start the navigation? (maybe the URL string is attractive), and why do the users tend to follow next to the homepage? (maybe the page has only external links or users do not find its contents interesting). Such questions suggest that something should be done about the design and contents of that page in order to take full advantage of its popularity in a way that encourages navigation within the site.
### 4.2. Real Data Experiments

#### Average results for the seven weeks with $\alpha = 0$

<table>
<thead>
<tr>
<th>Ranking</th>
<th>URL</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/</td>
<td>0.366</td>
</tr>
<tr>
<td>2</td>
<td>/Analogue-Heaven/</td>
<td>0.089</td>
</tr>
<tr>
<td>3</td>
<td>/samples.html</td>
<td>0.041</td>
</tr>
<tr>
<td>4</td>
<td>/manufacturers/</td>
<td>0.022</td>
</tr>
<tr>
<td>5</td>
<td>/manufacturers/Roland/info/</td>
<td>0.019</td>
</tr>
<tr>
<td>6</td>
<td>/manufacturers/Roland/TB-303/</td>
<td>0.016</td>
</tr>
<tr>
<td>7</td>
<td>/schematics.cgi</td>
<td>0.014</td>
</tr>
<tr>
<td>8</td>
<td>/manufacturers/Korg/</td>
<td>0.011</td>
</tr>
<tr>
<td>9</td>
<td>/DR-660/</td>
<td>0.011</td>
</tr>
<tr>
<td>10</td>
<td>/manufacturers/Moog/</td>
<td>0.010</td>
</tr>
</tbody>
</table>

#### Average results for the seven weeks with $\alpha = 1$

<table>
<thead>
<tr>
<th>Ranking</th>
<th>URL</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/</td>
<td>0.088</td>
</tr>
<tr>
<td>2</td>
<td>/manufacturers/</td>
<td>0.050</td>
</tr>
<tr>
<td>3</td>
<td>/samples.html?MMAgent</td>
<td>0.027</td>
</tr>
<tr>
<td>4</td>
<td>/samples.html</td>
<td>0.026</td>
</tr>
<tr>
<td>5</td>
<td>/manufacturers/Roland/</td>
<td>0.020</td>
</tr>
<tr>
<td>6</td>
<td>/links/</td>
<td>0.018</td>
</tr>
<tr>
<td>7</td>
<td>/Analogue-Heaven/</td>
<td>0.017</td>
</tr>
<tr>
<td>8</td>
<td>/categories/software/Windows/</td>
<td>0.016</td>
</tr>
<tr>
<td>9</td>
<td>/search.html</td>
<td>0.015</td>
</tr>
<tr>
<td>10</td>
<td>/categories/software/</td>
<td>0.014</td>
</tr>
</tbody>
</table>

#### Average results for the seven weeks with $\alpha = 0.5$

<table>
<thead>
<tr>
<th>Ranking</th>
<th>URL</th>
<th>Prob.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>/</td>
<td>0.227</td>
</tr>
<tr>
<td>2</td>
<td>/Analogue-Heaven/</td>
<td>0.053</td>
</tr>
<tr>
<td>3</td>
<td>/manufacturers/</td>
<td>0.036</td>
</tr>
<tr>
<td>4</td>
<td>/samples.html</td>
<td>0.033</td>
</tr>
<tr>
<td>5</td>
<td>/samples.html?MMAgent</td>
<td>0.017</td>
</tr>
<tr>
<td>6</td>
<td>/manufacturers/Roland/</td>
<td>0.015</td>
</tr>
<tr>
<td>7</td>
<td>/manufacturers/Roland/TB-303/</td>
<td>0.013</td>
</tr>
<tr>
<td>8</td>
<td>/manufacturers/Roland/info/</td>
<td>0.012</td>
</tr>
<tr>
<td>9</td>
<td>/links/</td>
<td>0.011</td>
</tr>
<tr>
<td>10</td>
<td>/schematics.cgi</td>
<td>0.011</td>
</tr>
</tbody>
</table>

Figure 4.7: The top-ten pages for several values of the $\alpha$ parameter.
Also, it is interesting to note that several pages containing information about specific manufacturers of music equipment are used as navigation starting points. This fact is probably reflective of the users' interests. Pages such as /categories/software/ and /search.html are popular among the users but not as a first page for the navigation. Finally, Figure 4.7 (c) shows the ten pages with the highest initial probability for \( \alpha = 0.5 \), which corresponds to a mixture of the two above scenarios.

In Figure 4.8 the characteristics of the rule-sets induced from three of the weeks are shown; the other four weeks present similar characteristics. These results correspond to runs with \( N = 2, \delta = 0.5 \), and \( \theta = 1/n \), where \( n \) is the number of states. As usual, NR stands for number of rules, ARL for average rule length, and MRL for maximum rule length. As expected, it is verified that when the value of \( \alpha \) increases the number of rules also increases since there are more states with positive initial probability. Similarly, an increase on the value of \( \alpha \) leads to a decrease in the average rule length since the expected value of the initial probability is smaller and, therefore, a trail tends to have smaller probability. The aggregated results summarising the characteristics of the rule-sets induced by the weekly grammars reveal stability across the weeks of the users' navigation behaviour. Also, for example, for \( \alpha = 1 \) and \( \delta = 0.5 \) there are 198 rules which are common to the seven rule-sets. These facts suggest that the aggregated results and the set of common rules have the potential of being useful as predictions for the user behaviour in the forthcoming weeks.

<table>
<thead>
<tr>
<th></th>
<th>Week 1</th>
<th></th>
<th>Week 2</th>
<th></th>
<th>Week 3</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>NR</td>
<td>ARL</td>
<td>MRL</td>
<td>NR</td>
<td>ARL</td>
<td>MRL</td>
</tr>
<tr>
<td>0</td>
<td>821</td>
<td>2.4</td>
<td>5</td>
<td>1020</td>
<td>2.2</td>
<td>5</td>
</tr>
<tr>
<td>0.5</td>
<td>992</td>
<td>2.0</td>
<td>5</td>
<td>1075</td>
<td>1.9</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>1035</td>
<td>1.8</td>
<td>4</td>
<td>1257</td>
<td>1.8</td>
<td>4</td>
</tr>
</tbody>
</table>

Figure 4.8: Characteristics of the rule-sets induced with \( N = 2 \) and \( \delta = 0.5 \) for several values of \( \alpha \).

One interesting analysis that can be carried out is the assessment of the web site coverage achieved by a given rule-set, that is, the percentage of the total number of pages in the site that are traversed by at least one rule in the rule-set. In Figure 4.9 we report the results of such analysis for rule-sets induced with \( \delta = 0.5, \theta = 1/n \), and for both \( \alpha = 0 \) and \( \alpha = 1 \); n.s. stands for number of states and NR for number of
4.2. Real Data Experiments

<table>
<thead>
<tr>
<th>Number of states covered by the top (X)% of the rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\alpha = 1, \delta = 0.5, \text{ and } \theta = 1/n)</td>
</tr>
<tr>
<td>(X = 100%)</td>
</tr>
<tr>
<td>----------</td>
</tr>
<tr>
<td>week 1</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 2</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 3</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 4</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 5</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 6</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>week 7</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Avg.</td>
</tr>
</tbody>
</table>

(a)

(b)

Figure 4.9: The percentage of pages in the web site covered by the best \(X\)% of the induced rules.

The rule-set that covers the largest part of the web site is composed of rules that traverse only 33.5% of the pages. Moreover, if we take into account only the longer rules, by considering for example only the top 25% of the rules, the coverage is much smaller. (Note that, in order to identify the best \(X\)% rules in a rule-set we sort the rules by length and only after by probability.) Also, for \(\alpha = 0\) a smaller number of rules is induced (see Figure 4.8) and the rule-sets achieve a more restricted coverage of the web site. However, as \(X\) gets smaller the difference becomes irrelevant.

The results reveal that users tend to browse more heavily a small sub-set of the web site. Therefore, improvements in the design of that restricted portion of the site
will benefit the majority of the users. On the other hand, the web site designer should analyse the subset of pages less frequently viewed in order to assess why those pages are not as popular. It may be the case that the contents of the less popular pages is more specific and not of interest to the average user. It may also be the case that such pages are not easy to find for users that start the navigation at the more popular entry points.

<table>
<thead>
<tr>
<th>Page URL</th>
<th>Page ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>1</td>
</tr>
<tr>
<td>/categories/software/</td>
<td>2</td>
</tr>
<tr>
<td>/categories/software/Windows/</td>
<td>3</td>
</tr>
<tr>
<td>/categories/software/Windows/README/</td>
<td>4</td>
</tr>
<tr>
<td>/categories/software/Windows/V909V03.TXT</td>
<td>5</td>
</tr>
<tr>
<td>/categories/software/Windows/dw8000.readme</td>
<td>6</td>
</tr>
<tr>
<td>/categories/software/Windows/WSEdit1c.txt</td>
<td>7</td>
</tr>
<tr>
<td>/categories/software/Windows/dumpster.txt</td>
<td>8</td>
</tr>
<tr>
<td>/categories/software/Windows/faders.txt</td>
<td>9</td>
</tr>
<tr>
<td>/categories/software/Windows/k4v311_readme.txt</td>
<td>10</td>
</tr>
<tr>
<td>/features/</td>
<td>11</td>
</tr>
<tr>
<td>/features/first-synth.html</td>
<td>12</td>
</tr>
<tr>
<td>/guide/</td>
<td>13</td>
</tr>
<tr>
<td>/guide/finding.html</td>
<td>14</td>
</tr>
<tr>
<td>/links/</td>
<td>15</td>
</tr>
<tr>
<td>/links/sites.html</td>
<td>16</td>
</tr>
<tr>
<td>/manufacturers</td>
<td>17</td>
</tr>
<tr>
<td>/new.html</td>
<td>18</td>
</tr>
<tr>
<td>/search.html</td>
<td>19</td>
</tr>
<tr>
<td>/Analogue-Heaven/</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 4.10: URL codification.

In Figure 4.10 we codify the relevant URLs, each with an integer number, in order to facilitate their reference in the analysis that follows. Figure 4.11 presents some examples of rules mined from the real data with \( \delta = 0.5, \theta = 1/n, \alpha = 1 \), and for both \( N = 2 \) and \( N = 3 \). All given rules occur in every week for the corresponding parameters. For \( N = 2 \) there are 198 rules which are common to all weeks and for \( N = 3 \) there are 185 common rules; Figure 4.11 shows the best ten rules for each of the two sets of parameters. In this work, the measure of a rule's usefulness is its probability and since the probability is inversely proportional to the rule's length we sort first the rules by length (Len.) and only after by probability (Prob.). Note that a higher probability means more usage, in the sense that more users have chosen to follow the
4.2. Real Data Experiments

<table>
<thead>
<tr>
<th>N = 2</th>
<th>N = 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1, 13, 14, 17</td>
</tr>
<tr>
<td>2</td>
<td>2, 3, 4, 3</td>
</tr>
<tr>
<td>3</td>
<td>1, 11, 12, 11</td>
</tr>
<tr>
<td>4</td>
<td>13, 14, 17</td>
</tr>
<tr>
<td>5</td>
<td>3, 5, 3</td>
</tr>
<tr>
<td>6</td>
<td>1, 13, 17</td>
</tr>
<tr>
<td>7</td>
<td>15, 16, 15</td>
</tr>
<tr>
<td>8</td>
<td>19, 20, 1</td>
</tr>
<tr>
<td>9</td>
<td>1, 18, 1</td>
</tr>
<tr>
<td>10</td>
<td>17, 1, 17</td>
</tr>
</tbody>
</table>

(a) (b)

Figure 4.11: Example of real data rules mined with δ = 0.5, θ = 1/n, and α = 1.

corresponding trail when browsing the web site.

For the results given in Figure 4.11 (a) each week of data was modelled as a HPG corresponding to a first order Markov chain, i.e., using a N gram with N = 2, and the α parameter was set with the value α = 1. In Figure 4.11 (b) the results correspond to a HPG having N = 3 and α = 1. As expected, the rules induced by the HPG with the larger N are significantly longer. Some of the rules in Figure 4.11 (b) correspond to expansions of rules in Figure 4.11 (a), for example, rule 2 in (a) is a prefix of rule 3 in (b) and rule 5 in (a) it is included in rule 4 in (b).

From the analysis of the rules in Figure 4.11 we notice that there is a considerable probability for users to, after viewing a page, return to the page that linked to that page, i.e., several rules are of the form a, b, a, c. By being aware of this fact the web site designer can determine if this behaviour is natural or if it has to do with some undesired characteristic of the site design or topology. For example, rules 2, 5, and 10 in Figure 4.11 (b) represent users viewing pages linked from the page /categories/software/Windows/, that is, page 3. Rule 5 shows that users tend to follow in sequence pages 6 and 10, which are both of the type 'readme'. Remember that this rule (and all the others in the Figure 4.11) is included in the rule-sets of all the weeks and, therefore, represents navigation behaviour which is very frequent among users. Thus, it may be worth to provide users with the possibility of viewing the pages of type 'readme' without having to go back to page 3 by providing links connecting
Similarly, rule 2 reveals that users tend to view in sequence pages 5 and 7, and rule 10 reveals that users tend to view in sequence pages 8 and 9. It can be the case that the contents of page 5 is related to that of page 7, and that the content of pages 8 and 9 are related. As such, it may be useful for users to provide direct links between pages 5 and 7 and between pages 8 and 9, in order to create trails linking pages with related content. Other pages whose content is considered related could be added to these trails. The analysis of log data from future weeks could be used to assess whether or not the new trails become traversed by the users.

A different aspect to be looked at is the one illustrated by rules 6 and 8 in Figure 4.11 (b). Rule 6 shows that is common among users to go back to the homepage immediately after viewing the contents of page 3. Thus, it seems that the links provided in page 3 are considered uninteresting by such users. In order to have a clear picture of the meaning of this rule it would be useful to know the average time users spent on page 3 before going back. Rule 8 reveals that there are also users which give up on following links from page 3 after viewing page 4, which is a global readme page describing the contents of the pages linked by page 3. This last case could be better analysed by knowing if these users were first time visitors to the page or users viewing the readme page to assess if there were new links provided in page 3.

One last interesting comment regarding the rules in Figure 4.11 is the frequency with which the homepage appears as the last state in a rule. This fact shows the important role that the homepage has as a reference stop for the navigation.

We now proceed with an analysis of the rules induced by a grammar representing the first week of usage, the grammar was built with $\alpha = 1$ and mined with $\delta = 0.5$ and $\theta = 1/n$. The rules starting at the homepage, which is the most popular page to start a navigation session, show that there is a significant probability of a user browsing next either page /manufacturers/, page /samples.html?MMAgent, or page /guide/.

Although the page /samples.html?MMAgent is not one of the most popular pages to start the navigation we notice that there is a significant number of rules with it as the first page. Also, the two most common subsequent pages are
categories/drum-machines/samples/ and the homepage /. The analysis of the page /samples.html?MMAgent reveals that the links to sample files are not organised into categories; the only collection of sample files organised into a category is the class of drum-machines. Having noticed that other pages regarding music-machines components are organised into fourteen categories, it is surprising that the samples’ page is not organised in the same way. Therefore, by being aware that when looking for sample files users tend to follow the only link which corresponds to a category or go back to the homepage, the web designer is given a clear indication that the page /samples.html?MMAgent should be redesigned in order to provide the links organised into categories.

One interesting rule may be /ecards/, /ecards/cards/, /ecards/cards/96301/, /ecards/cards/96301/0

which occur in five of the seven weeks as one of the rules with higher probability and it is the only rule whose starting page is /ecards/. Interestingly, these particular pages are no longer available in the web site.

The rule ranked as second among the rules composed by four pages induced from the HPG corresponding to the first week is

/, /guide/, /guide/finding.html, /manufacturers/.

This rule should be investigated because other rules imply that the homepage, /, has a direct link to the /manufacturers/ page and several users are reaching the latter page by a considerable longer path. It may be the case that the existing direct link is not clearly visible or that the text describing the link is not illustrative of the contents of the connected page.

In this section we have tried to illustrate the kind of analysis that is possible to carry out by making use of the HPG model. Such analysis is aimed at helping the web site designer to become aware of the sequences of pages having higher probability of being viewed in sequence. By using the HPG model the web site designer can gain some insight on the user behaviour and assess if pages with high probability of being viewed in the same session should be linked directly in order to facilitate user navigation. Similarly, the identification of pages not frequently visited is an essential step in
the analysis of the web site quality. Such pages may be difficult to reach from the popular entry points or have specialised contents not of the interest for the average user.

4.3 N-gram Analysis

The first order Markov chain provides an approximate model for the user navigation trails whose accuracy can be improved by the N-gram model. As was stated in Section 3.3 there is a trade-off between the model complexity (measured by its number of states) and its accuracy in representing the input trails. In our opinion, if the order of the chain is too high the model is uninteresting because user navigation sessions are short on average, see (Huberman et al., 1998), and a session shorter than the model's order has to be discarded. In addition, the probability of a very long trail being exactly repeated in the future is not very high.

In Figure 4.12 (a) we show the variation of the number of HPG states with the history depth for some of the weeks, the remaining weeks present a similar pattern. (Remember that history depth = N − 1.) The number of states in a N-gram is much smaller than the average case which is given by $n \cdot BF^{(N-1)}$, where $n$ represents the number of states and $BF$ the branching factor. The relatively small number of states in a N-gram suggests that some of the available trails are not considered interesting by users. However, it can also be the case that the data available is insufficient to characterise all the N-grams in a HPG with higher order, either because the users sessions are typically short or because the data should be collected during a longer period.

In Figure 4.12 (b) shows a plot of the absolute frequency of the sessions length. In the plot it can be seen that the majority of the sessions have less than five page requests. The data presented is the average for the seven weeks. Note that the plot in Figure 4.12 (b) resembles a Zipf distribution. In (Levene et al., 2000) we studied the hypothesis that the length of user navigation sessions follows Zipf's law and we have confirmed that hypothesis for two different real data sets.

We now present the results of the $\chi^2$ test which was performed in order to assess the best order for the HPG model. The test was performed for all the seven weeks and we varied the order of the model from a zeroagram to a sixgram. For each week the statis-
4.3. N-gram Analysis

Figure 4.12: Variation of the grammar size with the history depth and an histogram for the sessions' length.

tical test was performed for consecutive orders of the model. The null hypothesis was that the two consecutive orders were equivalent. If the experimental value (obtained from the data) takes a value greater than the critical value (obtained from the theoretical distribution), the equivalence hypothesis is rejected and the two models are said to be significantly different. If that is the case we say that the cost of going to the next complexity level is smaller than the gain in accuracy obtained with the higher order model.

The results of the conducted tests showed that in 5 of the 7 weeks we could not reject the hypothesis that a second order model was significantly different to the corresponding first order model. In one of the weeks we could reject that the first and second order models were equivalent but we could not reject the hypothesis that the third and second order models were equivalent. Finally, in one of the weeks we rejected all the tests until we reached the order 6. If we consider that this last case was an exception, it can be said that in many cases a first order model is a good representation of the data and that a second order model is enough for most of the cases. Figure 4.13 presents the test results for 3 of the seven weeks. Note that the Normal approximation was used as an approximation of the $\chi^2$ due to the very large number of degrees of freedom. The critical value, 1.96, was given by the statistical table for the Normal distribution where
4.4 Entropy Analysis

$P(Z \geq 1.96) = 0.25$, see (Wonnacott and Wonnacott, 1990).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>17201025</td>
<td>4522000</td>
<td>2858</td>
<td>48005720</td>
<td>7230717</td>
<td>5995</td>
<td>87055726</td>
<td>6469380</td>
<td>9598</td>
</tr>
<tr>
<td>2</td>
<td>258297</td>
<td>170648</td>
<td>134.54</td>
<td>636060</td>
<td>363660</td>
<td>275.05</td>
<td>1053317</td>
<td>280602</td>
<td>702.29</td>
</tr>
<tr>
<td>3</td>
<td>43050.2</td>
<td>44175</td>
<td>-3.80</td>
<td>99706.5</td>
<td>100180</td>
<td>-1.05</td>
<td>181744.8</td>
<td>85741</td>
<td>188.79</td>
</tr>
<tr>
<td>4</td>
<td>14354.3</td>
<td>15258</td>
<td>-5.24</td>
<td>38726.0</td>
<td>39311</td>
<td>-2.09</td>
<td>76801.0</td>
<td>37959</td>
<td>116.39</td>
</tr>
<tr>
<td>5</td>
<td>2288.1</td>
<td>3095</td>
<td>-11.02</td>
<td>7016.1</td>
<td>9043</td>
<td>-16.02</td>
<td>14149.6</td>
<td>8885</td>
<td>34.92</td>
</tr>
</tbody>
</table>

Figure 4.13: The $\chi^2$ test for some of the weeks in the real data.

Figure 4.14 shows the plot for the variation of the conditional entropy with the history depth for the real data used in the experiments. The plot shows that the biggest drop in the conditional entropy takes place when the $N = 2$, which corresponds to the first-order Markov model. This confirms the results obtained with the statistical test.

![Grammar conditional entropy with the history depth](image)

Figure 4.14: Variation of the conditional entropy with the $N$ of the $N$gram model.

4.4 Entropy Analysis

In order to verify the utility of the grammar's entropy as an estimator of the statistical properties of the grammar's language we have calculated for each grammar size and confidence threshold the correlation between: (i) the entropy and the number of rules, (ii) the entropy and the number of iterations, and (iii) the entropy and the average rule length. Since the results were not very promising and in order to assess the effect that both the confidence and support thresholds have on the grammar's entropy, we decided
to measure the entropy of a grammar inferred from the set of mined rules, which we call the *posterior grammar*. A posterior grammar is no more than a HPG whose input data is composed by a set of rules mined from a HPG, which we will call the *parent grammar*. We let PHr stand for the entropy rate of the posterior grammar, PHs for the Soule entropy of the posterior grammar, and MHR for the entropy of the parent grammar but using the vector of initial probabilities of the posterior grammar as the estimator for the stationary vector (the P stands for posterior and the M for mixed to indicate that MHR corresponds to a mixed definition).

![Entropy vs. number of rules with random data](image1)

![Entropies vs. number of rules with real data](image2)

**Figure 4.15**: Correlation between the grammar entropy and the number of rules.

Figure 4.15 (a) shows the results for randomly generated grammars with 2000 states. Each point in the plot represents, for a given configuration (size and confidence), the correlation between the number of rules mined and the entropy of the grammar. The results show that both PHr, PHs and MHR present a high correlation with the number of rules, above 0.8, but that PHr is a better estimator. The results regarding MHR suggest that it is possible to obtain a good estimator for the number of rules from the original grammar provided we are able to efficiently compute a good estimator for the stationary vector. In fact, the analysis of the results suggests that the stationary vector is of fundamental importance in the overall quality of the estimator. The experimental results have also shown that the entropy is not a good estimator of the number of iterations or of the average rule length. Note that the posterior grammar concept should be seen as
4.5 Summary

In this chapter we have presented the results of the experiments which were conducted to assess the usefulness of the hypertext probabilistic grammar model. Experiments with both synthetic and real data were conducted and the results suggest the validity of the synthetic data generation model. Moreover, the experimental results confirmed the theoretical analysis regarding the breadth-first search algorithm complexity, which was given in Section 3.5. Tests with the N-gram model suggested that a second order Markov model provides, in most cases, a good trade-off between the model accuracy and its size. Finally, some preliminary experiments regarding the use of entropy as an estimator of the number of rules were presented. The results showed that there is a correlation between the entropy and the number of rules but more work needs to be done in order for this measure to be a useful parameter for the analyst.
Chapter 5

Heuristics for Mining High Quality Navigation Patterns

In this chapter we present two new heuristics which aim at providing the analyst with enhanced control over the size and quality of the induced rule-set. The first heuristic implements an iterative deepening search wherein a set of rules is incrementally augmented by first expanding the trails with high probability. A stopping parameter is provided which measures the distance between the induced rule-set and its corresponding maximal set. The maximal rule-set is the set of rules returned by the BFS algorithm and includes all trails with probability above the cut-point; we use this set as the reference for the heuristics’ results. The second heuristic aims at finding longer rules composed of links with above the average probability. A dynamic threshold is provided whose value is set in a way that keeps its value proportional to the length of the trail being evaluated. In order to assess the usefulness of both heuristics extensive experiments were conducted with both real and synthetic data.

5.1 Motivation

The breadth-first search algorithm (BFS), presented in Section 3.5, is efficient in computing the set of all grammar strings with probability above the specified cut-point. However, it has the drawback of potentially returning a very large number of rules for small values of the cut-point and the rules are too short when the cut-point is close to one. This problem is illustrated in Figure 5.1 (a) that shows the variation of the number
of rules with the confidence threshold, and in Figure 5.1 (b) that shows the variation of both the average rule length (ARL) and maximum rule length (MRL) with the same threshold. From these plots it is clear that in order to obtain long rules the threshold has to be set with a low value but such a decrease in the cut-point leads to a very large number of rules.

We note that in this thesis we consider that a rule's quality is related to both its length and probability. In fact, due to the probabilistic nature of the HPG model the probability of a rule is inversely proportional to its length. Also, short strings with high probability can be implicitly characterised by longer rules which have such strings as prefix. Therefore, the analyst of log data may be interested in obtaining a relatively small set of long rules.

![Figure 5.1: Characteristics of the rule-set induced by the BFS algorithm.](image)

Despite the fact that it is possible to rank the rules in a large rule-set by their length and probability, or by some other criteria, and in that way identify the best rules, the manipulation of large rule-sets limits the algorithm's performance. On the one hand, when a rule-set is too large to be kept in main memory secondary memory has to be used, such as a swap file in the hard disk, which leads to slower performance. On the other hand, the database access time needed to store a large rule-set can be a severe limitation when the analyst is interactively performing tests with several model configurations. These problems led us to the study of heuristics to give the analyst more control over the qual-
ity of the inferred rule-set in terms of both the number and length of the rules obtained. Such heuristics are presented in the following sections.

5.2 Iterative Deepening Heuristic

The first heuristic implements an iterative deepening search (or simply ID) wherein the rule-set is incrementally built until it has the required characteristics. To that effect, a stopping parameter is provided which measures the distance between the rule-set being built and the corresponding maximal rule-set. The maximal rule-set is the set of rules induced by the BFS algorithm and includes all trails with probability above the cut-point. The maximal rule-set is used as reference for the heuristic’s results. The analyst specifies the value of a stopping criterion, which works as a threshold for the stopping parameter. The search for rules stops when the value of the stopping parameter surpasses the value of the stopping criterion. By setting the value of the stopping criterion the analyst is given control over the number and length of the rules obtained; the rule-set obtained is a proper subset of the maximal set.

The ID heuristic works as follows: an exploration tree is built from the start state and in the first iteration only the trails with length one are explored. Having built the tree with depth one the value of the stopping parameter is computed and evaluated against the stopping criterion, specified by the analyst. If the stopping criterion is met the exploration stops, otherwise the depth of the tree is incremented by one and all the trails with length two are explored. The stopping parameter is then re-computed and compared again with the stopping criterion; the process continues until the stopping criterion is met.

When a trail is being explored every out-link from its last state is evaluated to determine if it corresponds to an admissible expansion or an inadmissible expansion of the trail. A trail expansion is admissible if it leads to a trail with probability above the cut-point and inadmissible if not. Moreover, a trail is maximal when it has no admissible expansions and a trail is a candidate when it has at least one admissible expansion.

In the sequel $L^\lambda$ stands for the grammar’s language with cut-point $\lambda$, and $|L^\lambda|$ de-
notes its cardinality. The language $\mathcal{L}^\lambda$ is obtained by the BFS algorithm and includes only maximal trails. A sub-trail is implicitly defined since it is a prefix of a maximal trail. We let $k$ be the current depth of the exploration tree, measured by the number of links, and $\mathcal{L}_k^\lambda$ be the set of trails in the exploration tree, each trail with length less or equal than $k$ and probability above the cut-point. The language $\mathcal{L}_k^\lambda$ includes both the maximal and candidate trails from the exploration tree. Moreover, we let $T^\lambda$ be a random variable denoting the length of a trail and $P(T^\lambda \leq k)$ be the probability of a trail in the language $\mathcal{L}^\lambda$ be no longer than $k$. Given $\mathcal{L}_k^\lambda$, $P(T^\lambda \leq k)$ is computed with a one-step lookahead that is performed in order to identify both the admissible and inadmissible expansions, and consequently the maximal and candidate trails, in $\mathcal{L}_k^\lambda$. The probability $P(T^\lambda \leq k)$ corresponds to the sum of the probabilities of the maximal trails in $\mathcal{L}_k^\lambda$ and the inadmissible expansions of the candidate trails in the same set.

$P(\mathcal{L})$ stands for the sum of the probabilities of all trails in $\mathcal{L}$ where $\mathcal{L} \in \{ \mathcal{L}^\lambda, \mathcal{L}_k^\lambda \}$. Note that $P(\mathcal{L}_k^\lambda) \geq P(T^\lambda \leq k)$ since while both sets include the maximal trails with length less or equal to $k$, $\mathcal{L}_k^\lambda$ includes the candidate trails which are prefixes (and therefore have higher probability) of their inadmissible expansions included in $T^\lambda \leq k$. Also, $P(\mathcal{L}_k^\lambda) \geq P(\mathcal{L}^\lambda)$ since every trail in $\mathcal{L}_k^\lambda$ is a proper prefix of a maximal trail in $\mathcal{L}^\lambda$. Finally, for $k$ large enough $\mathcal{L}_k^\lambda = \mathcal{L}^\lambda$.

The idea behind the heuristic is to iteratively compute $\mathcal{L}_k^\lambda$ for $k = \{ 1, 2, \ldots \}$ until $\mathcal{L}_k^\lambda$ is a good enough approximation of $\mathcal{L}^\lambda$. For a given $k$ we measure how close $\mathcal{L}_k^\lambda$ is to $\mathcal{L}^\lambda$ by the value of the stopping parameter, which is defined as the remaining probability left to explore. For a given exploration tree, the stopping parameter is computed with a one-step lookahead and is defined as:

$$\eta_k = P(\mathcal{L}_k^\lambda) - P(T^\lambda \leq k).$$

A better estimator for $\eta_k$ can be computed with $n$-step lookahead, where if $n$ is large enough $P(T^\lambda \leq n) = P(\mathcal{L}^\lambda)$; however, the use of a $n$-step lookahead would lead to a decrease in performance.

Note that the value of the stopping parameter is defined as the sum of the probabilities of the admissible expansions of the candidate trails. The exploration stops when the stopping parameter is sufficiently small; we say that a number is sufficiently small
5.2. Iterative Deepening Heuristic

if it is less than some predefined $\tau$, $\tau > 0$, called the stopping criterion. At the end of
the exploration the rule-set is given by $\mathcal{L}_k$, since both the maximal and the candidate
trails are considered rules when the stopping criterion is met. Finally, note that $\eta_k/\lambda$ is
an upper bound estimator of the number of trails that can still be mined.

![Diagram](image)

Figure 5.2: Example of an exploration tree obtained with the ID heuristic for $k = 2$ and
$\lambda = 0.15$.

In Figure 5.2 we present an example of an exploration tree to clarify the concepts
introduced in this section. In the figure a pair of numbers next to a link represents the
probability of the link followed by the probability of the trail beginning in state $S$. For
$k = 2$ and $\lambda = 0.15$, before performing the lookahead, the set of candidate trails is
$\{A_1A_3, A_1A_4, A_2A_5\}$. When a one-step lookahead is performed from the last state of
the candidate trail $A_1A_3$ it is verified that both $A_3A_7$ and $A_3A_8$ are inadmissible expan­sions, therefore, $A_1A_3$ is a maximal trail. In addition, the lookahead identifies $A_4A_{10}$ as
an inadmissible expansion of $A_1A_4$ and $A_4A_9$ as an admissible expansion of the same
trail, thus, $A_1A_4$ is a candidate trail. Finally, both $A_5A_{11}$ and $A_5A_{12}$ are admissible ex-
5.2. Iterative Deepening Heuristic

Expansions of $A_2A_5$ meaning that $A_2A_5$ is a candidate trail.

Once all the trails with length less or equal to $k = 2$ are classified we can compute the value of the stopping parameter. To that effect, $P(\mathcal{L}_2^\lambda)$ represents the probability of the set of maximal and candidate trails whose length is less or equal to 2, that is:

$$P(\mathcal{L}_2^\lambda) = P(A_1A_3) + P(A_1A_4) + P(A_2A_5) = 0.25 + 0.25 + 0.4 = 0.9.$$ 

In addition, $P(T^\lambda \leq 2)$, is the probability of the length of a maximal trail be less or equal to 2, and is given by the sum of the probabilities of both the maximal trails and the inadmissible expansions of the candidate trails,

$$P(T^\lambda \leq 2) = P(A_1A_3) + P(A_1A_4A_10) = 0.25 + 0.025 = 0.275.$$ 

Therefore, the value of the stopping parameter is:

$$\eta_2 = P(\mathcal{L}_2^\lambda) - P(T^\lambda \leq k) = 0.9 - 0.275 = 0.625,$$

which corresponds to the sum of the probabilities of all admissible expansions of the candidate trails. Finally, the language with cut-point $\lambda = 0.15$ is

$$\mathcal{L}^\lambda = \{ A_1A_3, A_1A_4A_9A_{15}, A_2A_5A_{11}A_{17}, A_2A_5A_{12} \}$$

and $P(\mathcal{L}^\lambda) = 0.25 + 0.2 + 0.22 + 0.16 = 0.83$, therefore, we can verify that: \( P(\mathcal{L}_2^\lambda) = 0.9 > P(\mathcal{L}^\lambda) = 0.83 > P(T^\lambda \leq 2) = 0.275 \).

5.2.1 Synthetic Data Experiments

We have conducted a set of experiments with the ID approach and although the heuristic is efficient the stopping criterion is not fine enough. Figure 5.3 (a) shows the variation of the stopping parameter, $\eta_k$, with the exploration depth, and Figure 5.3 (b) shows the variation of the percentage of rules obtained with the exploration depth. In both figures it is shown that almost 80% of the stopping parameter variation occurs for $k \leq 2$, therefore, it is difficult to control the size of the induced rule-set with the value of the stopping criterion. In fact, if the stopping criterion is set at 0.52 the tree is explored only until $k = 1$ and we obtain less than 20% of the rules. If the stopping criterion is set at 0.48 the final exploration depth is $k = 2$ and close to 80% of the rules are obtained. The results are very consistent for different grammar sizes, as we can see by the overlap of the lines in the plots. To overcome this problem we decided to improve the concept in a way which would lead to smaller increases in $|\mathcal{L}_k^\lambda|$; this technique is described in the
5.3. Fine-Grained Iterative Deepening Heuristic

In this section we present the fine-grained iterative deepening heuristic (or simply FG) which aims at implementing a concept similar to the presented in Section 5.2 but in a way that gives more control over the number of rules obtained. As with the ID heuristic, an exploration tree containing the trails evaluated is incrementally built from the start state. However, in this case the tree is characterised by its size, which is measured by the number of links composing the trails already explored. At each stage, a link is chosen to expand a trail from the tree and the value of the stopping parameter is computed. Thus, the size of the tree is augmented by a single link at a time. If the value of the stopping parameter meets the stopping criterion the exploration stops. Otherwise, a new link is chosen to expand a trail in the tree and the process continues until the stopping criterion is met.

The FG heuristic explores the trails selectively, in contrast with the ID heuristic with which every admissible expansion of a candidate trail is explored at the same time. When selectively exploring an admissible expansion of a candidate trail the other existing admissible expansions of the candidate will continue unexplored. As a result, there can be a prefix of the resulting trail with admissible expansions still needed to be

Figure 5.3: Variation of the stopping parameter and of the percentage of rules mined with the exploration depth.
explored. Thus, in order to formalise the description of the FG heuristic we need to introduce a new type of trail, the candidate sub-trail. A candidate sub-trail is a prefix of a longer trail (either a maximal or a candidate trail) which has at least one admissible expansion still unexplored. And to recapitulate, a trail is maximal if it has no admissible expansions; a trail is a candidate when it has at least one admissible expansion but is not a prefix of any other candidate or maximal trail.

To clarify the candidate sub-trail concept we give an example from Figure 5.2. A one-step lookahead from state $A_i$ identifies both $A_iA_3$ and $A_iA_4$ as its admissible expansions. If $A_iA_3$ is chosen to be expanded first it will become a candidate trail and the trail $A_i$ will become a candidate sub-trail. Note that the trail $A_i$ has one unexplored admissible expansion, $A_iA_4$, and it is a sub-trail of the candidate trail $A_iA_3$. The fundamental difference between a candidate sub-trail and a candidate trail is that a sub-trail is always a prefix of either a maximal or a candidate trail. When the tree exploration stops, because the stopping criterion is met, the rule-set returned is composed of the maximal trails and the candidate trails; the candidate sub-trails are discarded since they are prefixes of one or more of the other trails.

In the FG heuristic we re-define $k$ to be the total number of links that constitute the exploration tree and we let $l^\lambda_k$ be a set of trails in the tree; $l^\lambda_k$ includes both the maximal and the candidate trails in the exploration tree. We let ${\mathcal L}_k^\lambda$ be the family of all the $l^\lambda_k$ sets, that is, ${\mathcal L}_k^\lambda = \{l^\lambda_k | \#l^\lambda_k = k\}$, where $\#l^\lambda_k$ represents the number of distinct links composing the trails in $l^\lambda_k$. Moreover, we let $P(l^\lambda_k)$ be the sum of the probabilities of all trails in $l^\lambda_k$. (Note that there can be several sets of trails whose total number of links is $k$ and that $l^\lambda_k$ doesn't include the candidate sub-trails.)

We let $t^\lambda$ be a variable denoting the sum of the lengths of a set of trails. Moreover, for a given $l^\lambda_k$, $P(t^\lambda \leq k)$ represents the probability of a trail in $l^\lambda_k$ being a maximal trail. $P(t^\lambda \leq k)$ is computed with a one-step lookahead from the candidate trails in order to identify both the maximal and candidate trails. In similarity to the ID heuristic, $P(t^\lambda \leq k)$ corresponds to the sum of the probabilities of the maximal trails and the inadmissible expansions of the candidate trails.

Now, we let $l^\lambda_0$ be the set of candidate sub-trails in $l^\lambda_k$ and $P(l^\lambda_0)$ be the sum of the
5.3. Fine-Grained Iterative Deepening Heuristic

unexplored admissible expansions of the candidate sub-trails. Then, for a given \( k \) and \( l^*_k \) the stopping parameter is defined as:

\[
\eta_k = P(l^*_k) - P(t^\lambda \leq k) + P(l^3). \]

The search for rules stops when the stopping parameter takes a value sufficiently close to the stopping criterion, \( \tau \). The induced rule-set is the set of explored trails, \( l^\lambda \), and contains both the maximal and candidate trails. Note that at each stage the value of the stopping parameter corresponds to the amount of probability left to explore, which is given by the sum of the probabilities of all the admissible expansions of both the candidate trails and the candidate sub-trails.

We propose two different criteria for deciding which trail to expand first. We call the first the best-trail criterion as it chooses to expand the trail corresponding to the highest trail probability from among all the admissible trail expansions in \( l^\lambda \). At each stage the trail chosen to be augmented will lead to the best possible new trail.

We call the second the greedy criterion as it chooses to expand the link with the highest probability from among all the admissible expansions in \( l^\lambda \). The probabilities of the transitive productions are in general significantly greater than the probabilities of the start productions, especially when \( \alpha > 0 \) and all the grammar states have a positive initial probability. Therefore, for the greedy criterion we multiply the probability of a candidate link by the probability of the start production corresponding to the first state in the trail. This way, the values of the candidate links are normalised in order to make them comparable to the probabilities from the start state. Otherwise, the admissible expansions of a candidate trail would, in general, have a much higher probability than the admissible expansions from the start state.

In Figure 5.2 the ID exploration tree for \( k = 2 \) and \( \lambda = 0.15 \) corresponds to a FG tree for \( k = 5 \) and the same cut-point; in fact, that is the tree obtained with the best-trail criterion. In this case we have that \( l^\lambda = \{ A_1A_3, A_1A_4, A_2A_5 \} \), and 

\[
P(l^\lambda) = P(A_1A_3) + P(A_1A_4) + P(A_2A_5) = 0.9 .
\]

In addition, \( P(t^\lambda \leq 5) = P(A_1A_3) + P(A_1A_4A_{10}) = 0.275 \) , \( P(t^\lambda) = 0 \) and the stopping parameter is 

\[
\eta_S = P(l^\lambda) - P(t^\lambda \leq 5) + P(l^3) = 0.625 .
\]
5.3. Fine-Grained Iterative Deepening Heuristic

Figure 5.4: An example of an exploration tree obtained with the fine-grained heuristic when using the greedy criterion for $k = 5$ and $\lambda = 0.15$.

Note that, in this particular case there are no candidate sub-trails. According to the best-trail criterion the next trail to be expanded is $A_2A_5$ to $A_2A_5A_{11}$, which is the trail having the highest probability among the trails corresponding to admissible expansions.

Figure 5.4 gives another example of an exploration tree obtained with the FG heuristic for $k = 5$ and $\lambda = 0.15$, in this case the greedy criterion was used. The exploration tree is such that $t_5^A = \{A_1, A_2A_5A_{11}A_{17}\}$ wherein $A_1$ is a candidate trail and $A_2A_5A_{11}A_{17}$ is a maximal trail. The trail $A_2A_5$ is a candidate sub-trail since it has the unexplored admissible expansion $A_5A_{12}$. Therefore,

\[ P(t_5^A) = P(A_1) + P(A_2A_5A_{11}A_{17}) = 0.72, \]
\[ P(t^A \leq 5) = P(A_2A_5A_{11}A_{17}) = 0.22, \]
\[ P(t_5^A) = P(A_2A_5A_{12}) = 0.16, \text{ and } \eta_5 = 0.72 - 0.22 + 0.16 = 0.66. \]

According to the greedy criterion the next trail to be expanded is either $A_1A_3$ or $A_1A_4$.

When using the best-trail criterion the induced rule-set contains the best set of trails that is possible to obtain for a given exploration tree size. Note that a maximal trail im-
5.3. Fine-Grained Iterative Deepening Heuristic

Explicitly gives its set of sub-trails, where a sub-trail corresponds to a prefix of the corresponding maximal trail. Herein, we are defining the best set of trails to be the set which maximises the sum of the probabilities of both its maximal trails and its sub-trails. For example, in Figure 5.2 and for \( k = 5 \) the sum of the probabilities of all the trails in \( l^5_5 \) is:

\[
P(A_1) + P(A_1A_3) + P(A_1A_4) + P(A_2) + P(A_2A_5) = 0.5 + 0.25 + 0.25 + 0.5 + 0.4 = 1.9.
\]

For \( k = 5 \) there is no other such set of trails with higher value for the sum of its trails' probabilities.

**Proposition 5.1.** For a given \( k \), the best-trail criterion of the fine-grained heuristic gives the \( l^\lambda_k \) whose sum of the trail probabilities is maximal (including the probabilities of the sub-trails).

**Proof.** This property will be proved by induction.

For \( k = 1 \) the property holds trivially since \( l^1_k \) contains a single trail which was chosen because it has the highest probability.

If for \( k = n \) we have that \( l^\lambda_k \) contains the best set of trails, the property will continue to hold for \( k = n + 1 \) since the unexplored trail with the highest probability is added to \( l^\lambda_k \) in order to obtain \( l^\lambda_{k+1} \). □

Algorithm 5.1 gives the pseudo-code for the fine-grained heuristic wherein, as usual, \( \lambda \) is the HPG's cut-point, \( S \) is the starting state, \( \eta_k \) is the stopping parameter, and \( \tau \) is the stopping criterion. In addition, we let \( CT \) be the set of candidate trails, \( CST \) be the set of candidate sub-trails, and \( RS \) be the set of rules (the maximal trails). The method \( X.push() \) inserts a new trail in a trail-set where \( X \in \{ CST, CT, RS \} \), and the method \( y.lookahead() \) performs a one-step lookahead on \( y \in \{ S, t \} \) where \( t \) is a trail. Moreover, method \( CST.bestTrail() \) returns the best expansion of all the candidate sub-trails and \( CT.bestTrail() \) the best admissible expansion of all the candidate trails; \( \max(t_1, t_2) \) returns the best between the \( t_1 \) and \( t_2 \) trails. If the best trail, \( \bar{t} \), comes from \( CT \) then \( t \) represents the trail which was expanded to obtain \( \bar{t} \) and the method \( CST.update() \) inserts \( t \) in \( CST \) if it still has admissible expansions unexplored.
FineGrained (λ, τ)
1. begin
2. \( k = 0; \)
3. \( S.\text{lookahead}(); \)
4. compute \( \eta_k; \)
5. while \( \eta_k > \tau \) do
6. \( \bar{t} = \max(CST.\text{bestTrail}(), CT.\text{bestTrail}()); \)
7. if \( t \in CT \) then
8. \( CST.\text{update}(); \)
9. end if
10. \( \bar{t}.\text{lookahead}(); \)
11. if \( (\bar{t} \text{ is maximal}) \) then
12. \( RS.\text{push}(\bar{t}); \)
13. else
14. \( CT.\text{push}(\bar{t}); \)
15. end if
16. \( k = k + 1; \)
17. recompute \( \eta_k; \)
18. end while
19. end.

Algorithm 5.1: The pseudo-code for the fine-grained heuristic with the best-trail criterion.

Figure 5.5 exemplifies the computation of the sets \( CST, CT, RS \) and of the stopping parameter when the greedy version of the FG heuristic is applied to the grammar in Figure 5.4. The cut-point was set with the value \( \lambda = 0.15 \). For each member of the \( CST \) set the value inside brackets gives the product of the probability of the trail's first link and the probability of the link corresponding to the admissible expansion, that is, the value for the greedy criterion. For a trail in the \( CT \) set each admissible expansion of the trail is represented inside brackets, the link corresponding to the expansion is given followed by its value for the greedy criterion. The symbol (*) indicates the trail chosen to be expanded in the next iteration which corresponds to the trail with the highest value for the greedy criterion.

Similarly, Figure 5.6 gives the contents of the trail-sets when the best-trail criterion is used. In this case the value inside brackets represents the trail probability, since with this version the trail with highest probability is the one chosen to be expanded.
5.3. Fine-Grained Iterative Deepening Heuristic

<table>
<thead>
<tr>
<th>k</th>
<th>CST</th>
<th>CT</th>
<th>RS</th>
<th>η_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$A_1(0.5)^*, A_2(0.5)$</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$A_2(0.5)^*$</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$ $A_2(A_5, 0.4)^*$</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$ $A_2A_5(A_11, 0.3)^*(A_{12}, 0.2)$</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>4</td>
<td>$A_2A_5A_{12}(0.2)$</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$ $A_2A_5A_{11}(A_{17}, 0.45)^*$</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>5</td>
<td>$A_2A_5A_{12}(0.2)$</td>
<td>$A_1(A_3, 0.25)^*(A_4, 0.25)$ $A_2A_5A_{11}A_{17}$</td>
<td>-</td>
<td>0.66</td>
</tr>
<tr>
<td>6</td>
<td>$A_2A_5A_{12}(0.2)$ $A_1A_4(0.25)^*$</td>
<td>$A_2A_5A_{11}A_{17}$ $A_1A_3$</td>
<td>-</td>
<td>0.41</td>
</tr>
</tbody>
</table>

Figure 5.5: The contents of the sets of trails induced by the FG heuristic when using the greedy criterion for $\lambda = 0.15$.

<table>
<thead>
<tr>
<th>k</th>
<th>CST</th>
<th>CT</th>
<th>RS</th>
<th>η_k</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$A_1(0.5)^*, A_2(0.5)$</td>
<td>-</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>$A_2(0.5)^*$</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>-</td>
<td>$A_1(A_3, 0.25)(A_4, 0.25)$ $A_2(A_5, 0.4)^*$</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>3</td>
<td>-</td>
<td>$A_1(A_3, 0.25)^*(A_4, 0.25)$ $A_2A_5(A_{11}, 0.24)(A_{12}, 0.16)$</td>
<td>-</td>
<td>0.9</td>
</tr>
<tr>
<td>4</td>
<td>$A_1A_4(0.25)^*$</td>
<td>$A_2A_5(A_{11}, 0.24)(A_{12}, 0.16)$ $A_1A_3$</td>
<td>-</td>
<td>0.65</td>
</tr>
<tr>
<td>5</td>
<td>-</td>
<td>$A_1A_4(A_3, 0.225)$ $A_2A_5(A_{11}, 0.24)^*(A_{12}, 0.16)$ $A_1A_3$</td>
<td>-</td>
<td>0.625</td>
</tr>
<tr>
<td>6</td>
<td>$A_2A_5A_{12}(0.16)$</td>
<td>$A_1A_4(A_9, 0.225)^*$ $A_2A_5A_{11}(A_{17}, 0.22)$ $A_1A_3$</td>
<td>-</td>
<td>0.605</td>
</tr>
</tbody>
</table>

Figure 5.6: The contents of the sets of trails induced by the FG heuristic when using the best-trail criterion for $\lambda = 0.15$.

We should note that the fine-grained heuristic was developed with the objective of giving the analyst as much control as possible over the number and quality of the mined rules by setting the value for the stopping criterion. When using the FG heuristic we expect the analyst to choose to explore a relatively small number of trails, otherwise he would use the BFS algorithm instead. Therefore, when devising the FG heuristic we were mainly concerned with providing a stopping criterion with tight control over the number of rules and the heuristic’s performance wasn’t considered a critical issue. There are, however, some simple modifications which can lead to improvements in the heuristic’s performance. The first modification consists in exploring a specified number of links at a time instead of only one link at a time, that is, to expand the $X$ best trails,
where $X$ is a parameter specified by the analyst. The parameter $X$ should be specified in a way that takes into account both the grammar size and the control the analyst wants to have over the number of rules. When $X$ increases the control the stopping parameter gives over the number of rules decreases but the performance improves. Note that when $X$ corresponds to the number of admissible expansions the FG heuristic is equivalent to the ID heuristic.

The second modification consists in keeping separately two small sorted sets of trails from the exploration tree. One set with the best elements from $CT$ and the other with the best elements from $CST$; the size of these sets could be a parameter. When using such sets the next best trail to be expanded could always be found among the trails in those small ordered sets, without having to traverse the original sets. Note that in the standard fine-grained heuristic in order to find the next best trail both the set $CST$ and $CT$ have to be entirely traversed, as it would be very demanding to keep such sets sorted. Since the last expanded trail has always smaller probability than its prefix, the small sets would always contain the best trails. Also, whenever a new trail had to be inserted into $CST$ or $CT$ it would be verified if the new trail was better than at least one trail in the corresponding sorted set. If the answer was positive, the new trail could be inserted or replace the worst trail in the sorted set (depending on whether or not the set was full). When the small sets were empty they would have to be refilled.

5.3.1 Experimental Evaluation

To assess the effectiveness of the proposed heuristic experiments were conducted with both synthetic and real data. The synthetic data generation model used was the same described in Section 4.1.1 and the real data used was the one described in Section 4.2.1.

For both the real data and synthetic data experiments we fixed the support at $1/n$, where $n$ is the number of grammar states, and varied the confidence threshold between 0.1 and 0.9. The stopping criterion was set to vary within the range of 0.1 to 0.98. For the synthetic data, the size of the randomly generated grammars varied between 1000 and 20000 states and all the results presented correspond to the average of 30 runs.

As stated in Section 5.3, the main objective of the fine-grained heuristic is to provide tight control over the number of rules mined by setting the value of the stopping
5.3. Fine-Grained Iterative Deepening Heuristic

criterion. Note that, if the stopping criterion takes the value 0 the rule-set induced by
the heuristic is exactly the same as the given by the breadth-first search algorithm.

Figure 5.7 (a) shows the variation of the number of rules obtained with different
values of the stopping criterion for a set of randomly generated grammars with 10000
states and an average branching factor (BF) of 5 out-links per page. The results show
that the number of rules induced decreases smoothly with the value of the stopping
criterion, the two variables have a relationship with a negative slope whose value de­
creases with the confidence. Therefore, the stopping criterion gives the analyst good
control over the number of rules obtained. Moreover, Figure 5.7 (a) shows that only
for small values of the confidence it is possible to distinguish the number of rules re­
turned by the greedy and best-trail versions.

Figure 5.7 (b) shows the variation of the average rule length (ARL) and the maxi­
imum rule length (MRL) with the stopping criterion when the confidence threshold was
set at 0.1. It can be seen that even for high values of the stopping criterion, which cor­
respond to small rule-sets, it is possible to obtain long rules, especially with the greedy
version. In fact, with the greedy version it is possible to obtain some rules close to the
maximal length with the stopping criterion set to 0.70.

Figure 5.8 shows the fine-grained heuristic performance results for different values
5.3. Fine-Grained Iterative Deepening Heuristic

Figure 5.8: The performance of the fine-grained heuristic with the synthetic data.

of the stopping criterion (SC). While Figure 5.8 (a) gives the variation of the number of iterations with the size of the grammar Figure 5.8 (b) gives the variation of the heuristic running time with the number of states. The performance of the BFS algorithm is also given. As expected, the results show that the FG heuristic is much more time consuming than the BFS. While the BFS algorithm was shown in Section 3.5 to exhibit a linear behaviour the results given herein suggest that the FG heuristic is polynomial. In fact, we were able to fit a quadratic function in all the given curves, always with a correlation greater than 0.999. However, as was stated in Section 5.3, the performance is not a primary issue for the FG heuristic, since the analyst is expected to induce small sub-sets of rules or to perform a step by step induction of the rule-set. Moreover, some directions for the improvement of its performance were also given. Finally, the two versions of the heuristic are indistinguishable in the number of iterations performed although when considering the running time the greedy version performs slightly better.

Figure 5.9 (a) shows the variation of the number of rules obtained with the stopping criterion for the real data. Each point in the plot corresponds to the average results of the seven data sets for a given configuration; each data set corresponds to a week of log data. The results confirm those obtained with synthetic data by showing that there is a smooth variation of the number of rules with the value of the stopping criterion. In fact, a small variation in the value of the stopping criterion will lead to a similar variation in
5.3. Fine-Grained Iterative Deepening Heuristic

the number of rules obtained and that variation is close to uniform throughout the range of the stopping criterion.

Figure 5.9 (b) shows how both ARL and MRL vary with the stopping parameter for the real data sets. The results confirm those obtained with synthetic data. Moreover, it is interesting to note that with the greedy version the average rule length has its maximum value when the stopping criterion is set at 0.7, which is another indication that some long trails tend to be explored first.

Figure 5.9: The variation of the number of rules and the size of the rules with the fine-grained stopping criterion when applied to the real data.

According to the definitions in Section 5.3 the rule-set inferred contains both the candidate and the maximal trails for a given value of the stopping parameter. Alternatively, the heuristic could be set up to give the analyst the maximal trails only.

Figure 5.10 provides a detailed analysis of the rule-set composition by giving the amount of probability in it that corresponds to maximal trails and the amount of probability that corresponds to candidate trails. In fact, Figure 5.10 shows the variation of the sum of the probabilities of the maximal trails (RS) and of the candidate trails (CT) with the value of the stopping criterion. While Figure 5.10 (a) corresponds to the best-trail version Figure 5.10 (b) corresponds to the greedy version. The results show that the best-trail version tends to keep a much larger set of candidate trails and consequently a smaller set of rules. This is explained by the ability of the greedy version to obtain
some maximal trails at early stages of the exploration. This fact helps to explain why the running time of the greedy version is better since when having to locate the trail to expand next, among all the candidate and admissible trails, it has to traverse a smaller set of candidates than the corresponding best-trail version.

5.4 Inverse Fisheye Heuristic

While the fine-grained heuristic is useful to compute a subset of the maximal set (the set of rules returned by the BFS algorithm) the analyst would also benefit if able to compute a rule-set with different characteristics, such as a set of long trails. Due to the probabilistic nature of the HPG model the probability of a trail is inversely proportional to its length. Therefore, in order to obtain long rules the cut-point has to be set with a low value, as can be seen in Figure 5.1, and a low value of the cut-point results in an unmanageable number of rules being obtained. Although, the rule-sets obtained in such a way contain long trails they also include a large number of very short and uninteresting trails; the problems associated with manipulating very large rule-sets are stated in Section 5.1.

In this section we propose a novel heuristic aimed at inducing a relatively small set of long trails composed of links with high probability on average. To that effect, we
5.4. Inverse Fisheye Heuristic

make use of a *dynamic cut-point* which imposes a very strict criterion for short trails and becomes more permissible as the trails get longer.

The idea of a dynamic cut-point was motivated by the fisheye-view concept of Furnas (Furnas, 1986), wherein a method is proposed to visualise large information data structures on relatively small computer displays. In his work, Furnas proposes a measure of the user's interest in a given document. Such measure takes into account the user's location in the information structure, being its value proportional to the global importance of the document and inversely proportional to its distance from the document being displayed. An overview diagram can thus be built in such a way that the detail with which each document is shown on the display is proportional to its score.

We call our dynamic cut-point method the *inverse fisheye* heuristic (IFE) since, as opposed to the Furnas concept, our measure of interest benefits those pages that are further away from the start of a trail being evaluated. With the IFE heuristic the cut-point is defined in such a way that it becomes more permissible as the trails get longer.

We propose two different ways of setting the dynamic cut-point. In the first version the cut-point keeps its value proportional to the depth of exploration, and in the second version the cut-point is devalued by a factor proportional to the expected decrease in the trail probability. Note that there are other ways of defining a dynamic cut-point and that we have chosen these two in order to assess the usefulness of the concept.

We call the first method for setting the dynamic cut-point the *geometric cut-point* version. The initial value for cut-point is set by means of its two components, the support and the confidence thresholds. Moreover, an exploration tree is incrementally built from the start state while the value of the cut-point is updated as a function of the depth of the exploration. The geometric cut-point is defined to be

$$
\lambda = \theta \delta^d
$$

where $\theta$ is the support threshold, $\delta$ is the confidence threshold, and $d$ is the depth of the exploration tree measured by the number of links.

With the geometric version the cut-point is devalued geometrically in a way that keeps its value proportional to the trail's length. In fact, in the beginning of the exploration, when evaluating links from the start state, the depth of the tree is 0 and the cut-
point corresponds to the support threshold value. Note that the support threshold component of the cut-point was defined precisely to evaluate the start productions. In the subsequent stages of the exploration the cut-point incorporates the confidence threshold a number of times corresponding to the number of transitive productions that derive the trails being evaluated.

We call the second method for setting the dynamic cut-point the branching factor cut-point version. Again, an exploration tree is built and the cut-point is devalued as a function of the depth of exploration. With this version, however, the devaluation of the cut-point takes into account the grammar's average branching factor, \( BF = l/n \), where \( n \) denotes the number of grammar states and \( l \) the number of grammar links. The grammar's branching factor corresponds to the expected number of out-links in a state and \( 1/\lfloor BF \rfloor \) to the average probability of an out-link. Therefore, we define the branching factor threshold as

\[
\lambda_B = \theta \text{ if } d = 0 \quad \text{and} \quad \lambda_B = \theta \frac{\delta}{BF^{(d-1)}} \text{ if } d \geq 1,
\]

where \( \theta \) is the support threshold, \( \delta \) is the confidence threshold, and \( d \) is the depth of the exploration tree measured in the number of links. This way, when a trail is expanded with a new link the branching factor cut-point \( \lambda_B \) is devalued by a factor proportional to the expected decrease in the trail probability. Note that for \( d \leq 1 \) we have that \( \lambda_B = \lambda_G \).

With both versions of the IFE heuristic one additional parameter has to be specified which is the maximum depth of exploration, \( \bar{d} \). This parameter sets an upper bound for the length of the rules mined and it is necessary because, for small values of the initial cut-point, there is no guarantee that the exploration will terminate. In fact, there can be a cycle of links generating a trail whose probability decreases at a slower rate than the dynamic cut-point. In such cases a trail would be explored indefinitely.

We now present some interesting properties of the inverse fisheye heuristic.

**Proposition 5.2.** If \( \delta = \frac{1}{BF} \) then \( \lambda_G = \lambda_B \).

**Proof.** If \( d = 0 \) we have \( \lambda_G = \theta \delta^0 = \theta \) meaning that \( \lambda_G = \lambda_B \), \( \forall \delta \).

If \( d = 1 \) we have \( \lambda_G = \theta \delta^1 = \theta \delta \frac{1}{BF} = \lambda_B \), \( \forall \delta \).

If \( d > 1 \) we have \( \lambda_G = \lambda_B \quad \equiv \quad \theta \delta^d = \theta \delta \frac{1}{BF^{d-1}} \quad \equiv \quad \delta = \frac{1}{\sqrt[BF^{d-1}]{BF}} = \frac{1}{BF^d}. \) \( \square \)
This property shows that the two methods for setting the dynamic cut-point are equivalent when $\delta = 1/\text{BF}$.

**Proposition 5.3.** If $\delta > \frac{1}{\text{BF}}$ then $\lambda_B < \lambda_G$.

**Proof.** $\lambda_B < \lambda_G \equiv \theta \delta \frac{1}{\text{BF}^{d-1}} < \theta \delta^d \equiv \delta > \frac{1}{\text{BF}}$. □

This second property shows that when $\delta > 1/\text{BF}$ the branching factor version is more permissible than the geometric version since its value decreases faster.

**Proposition 5.4.** For $\lambda_G$ and a trail $t = ba_1 \ldots a_m$, where $b$ and $a_i$, $1 \leq i \leq m$ are the links composing the trail, we have:

$$\frac{p(a_1) + p(a_2) + \ldots + p(a_m)}{m} > \delta \left( \frac{\theta}{p(b)} \right)^{\frac{1}{m}}.$$

**Proof.** $p(b)p(a_1)p(a_2)\ldots p(a_m) > \theta \delta^m \equiv \left( \frac{p(b)p(a_1)p(a_2)\ldots p(a_m)}{q} \right)^{\frac{1}{m}} \left( \frac{\theta}{p(b)} \right)^{\frac{1}{m}} > \delta \equiv \left( p(a_1)p(a_2)\ldots p(a_m) \right)^{\frac{1}{m}} > \delta \left( \frac{\theta}{p(b)} \right)^{\frac{1}{m}}.$$

and by the theorem of the geometric means, see (Kazarinoff, 1961), it follows that:

$$\frac{p(a_1) + p(a_2) + \ldots + p(a_m)}{m} > \delta \left( \frac{\theta}{p(b)} \right)^{\frac{1}{m}}. \quad \Box$$

This last property implies that when $p(b)$ is close to $\theta$ the average link probability of a rule induced by the geometric version of the IFE heuristic is greater than or equal to the confidence threshold $\delta$. Thus, the rule-set is composed of rules whose average link probability is greater than the confidence threshold. Note, however, that the property is not complete in the sense that not all trails with average link probability greater than $\delta$ are included in the induced rule-set due to the inverse fisheye property.

Figure 5.11 gives an example of an exploration tree where the average branching factor is $\text{BF} = 2$, the support is $\theta = 0.49$, and the initial confidence is $\delta = 0.6$. The values of the dynamic cut-point for the two versions of the heuristic are given in the figure, as well as the rules induced by the two versions and by the BFS algorithm. In Figure 5.11 a pair of numbers next to a link represents the probability of the link, followed by the probability of the trail beginning in the start state $S$. When $d = 0$ both versions of the cut-point correspond to the value of the support threshold, when $d = 1$
both criteria correspond to the product of the confidence and the support thresholds $\delta \theta$, and the inverse fisheye property only starts having effect for $d \geq 2$. Note that with the geometric version only trails whose average probability of the transitive productions is greater than $\delta = 0.6$ are considered rules; trail $A_2A_5A_{12}$ doesn’t meet that criterion since its average link probability is just 0.6. Moreover, trail $A_1A_4A_9$ meets the acceptance criterion for $d = 2$ since its probability, $p(A_1A_4A_9) = 0.18$, is above the value of both dynamic cut-points; however, the trail is not a rule since it is rejected at an early stage when $p(A_1A_4) = 0.2 < 0.3$. Finally, the example also shows that the BFS for the given configuration is ineffective since it induces a single short rule, $A_2A_5$. On the
other hand, if the BFS algorithm is set up to run with the cut-point \( \lambda = 0.176 \) (the final value for the geometric version) the gain relative to the geometric version would only be the short rule \( A_1 A_4 \).

\( \text{InverseFisheye} \ (\delta, \ \theta, \ \bar{d}) \)
1. begin
2. \( d = 0; \)
3. for \( i = 1 \) to \( n \)
4. if \( (p(S \rightarrow A_i) > \theta) \) then
5. \( CT.\text{push}(a_i A_i, p(S \rightarrow A_i)); \)
6. end if
7. end for
8. while \((CT.\text{empty}() == \text{false} \text{ OR } d \leq \bar{d})\) do
9. \( d = d + 1; \)
10. \( \lambda = \theta \delta^d; \)
11. for each \( wA_i \in CR \)
12. \( \text{flag} = \text{false}; \)
13. for \( j = 1 \) to \( P_{i,} \)
14. if \( (p(wA_i) \cdot p(P_{i,j}) > \lambda) \) then
15. \( CT_1.\text{push}(wa_i A_j, p(wA_i) \cdot p(P_{i,j})); \)
16. \( \text{flag} = \text{true}; \)
17. end if
18. end for
19. if \( (\text{flag} == \text{false}) \) then
20. \( RS.\text{push}(w, p(w)); \)
21. end if
22. end for
23. \( CT = CT_1; \)
24. \( CT_1 = \text{null}; \)
25. end while
26. end.

Algorithm 5.2: The pseudo-code for the geometric version of the inverse fisheye heuristic.

Finally, in Algorithm 5.2 we give the pseudo-code for the inverse fisheye heuristic. We let \( \lambda \) be the dynamic cut-point, \( \theta \) be the support threshold, \( \delta \) be the confidence threshold, and \( \bar{d} \) be the maximum exploration depth; \( n \) represents the number of states in the grammar. We let \( CT \) and \( CT_1 \) be two different sets of candidate trails and \( RS \) be a set of rules. The method \( CT.\text{empty}() \) checks if a set of candidate trails is empty.
and the method $X$.push() inserts a new trail into the final rule-set or into a candidate trail-set, where $X \in \{RS, CT, CT_1\}$. We let a trail be represented by $wA_i$ where $w$ represents its terminal symbols and $A_i$ its unique nonterminal symbol; $P_{ij}, 1 \leq j \leq P_{i*}$ represents the set of productions whose left-hand side is $A_i$ and $P_{i*}$ its cardinality.

5.4.1 Experimental Evaluation

To assess the effectiveness of the inverse fisheye heuristic (IFE) experiments were conducted with both synthetic and real data. The synthetic data generation model used was the same described in Section 4.1.1 and the real data set used was the one described in Section 4.2.1.

For both data sets, unless otherwise stated, the reported results were obtained with the support threshold fixed at $1/n$, where $n$ is the number of grammar states; the confidence threshold was set to vary between 0.1 and 0.9. For the synthetic data the size of the randomly generated grammars varied between 1000 and 20000 states and the average branching factor varied between 3 and 7 out-links per page. All the presented results correspond to the average of 30 runs.

As stated in Section 5.4 the goal of the IFE is to induce a relatively small set of long trails and, in addition to specifying the support and confidence thresholds, the user has to specify the maximum depth of exploration, $d$. In the experiments we have set the maximum exploration depth to vary between 3 and 6.

Figure 5.12 (a) shows the variation in the number of mined rules with the confidence threshold when the maximum depth of exploration is set to 6. The IFE heuristic induces smaller rule-sets than the BFS algorithm unless the confidence is set with a very low value. Note that for the IFE heuristic the confidence should be set with a value higher than $1/BF$, otherwise every trail has high probability of being a rule; remember that $BF$ stands for the grammar's branching factor. In fact, $1/BF$ is the average link probability and when the threshold is close to that value every link being evaluated has high probability of passing the threshold test.

Figure 5.12 (b) shows the variation of both the average rule length (ARL) and the maximum rule length (MRL) with the confidence threshold. The results show that both versions of the IFE heuristic induce rule-sets with longer trails than the BFS algorithm.
5.4. Inverse Fisheye Heuristic

Figure 5.12: The characteristics of the rule-set induced by the Inverse fisheye heuristic with synthetic data.

In fact the branching threshold version is able to find trails with the maximum size (set by the exploration depth parameter) even for very high values of the confidence threshold.

When comparing the results given by the IFE heuristic with those given by the BFS algorithm we should take into consideration that the value given for the confidence threshold corresponds to the initial values for the IFE heuristic. For example, for an exploration depth of \( d = 6 \) and a confidence threshold of \( \delta = 0.5 \), the geometric cut-point takes a final value of \( \lambda = \frac{1}{n} \cdot 0.5^6 = \frac{1}{n} \cdot 0.0156 \) while the BFS algorithm evaluates all the trails with the cut-point \( \lambda = \frac{1}{n} \cdot 0.5 \).

Figure 5.13 (a) shows the variation in the number of iterations of the IFE heuristic with the grammar's number of states; both versions of the heuristic present a linear behaviour. The branching factor version is more demanding than the geometric for high values of the initial confidence threshold and when the threshold takes values close to \( 1/BF \) the opposite occurs. This last fact is explained by Proposition 5.3 which states that when the confidence threshold is \( \delta < 1/BF \) the geometric cut-point decreases at a faster rate and consequently induces a larger number rules.

According to Proposition 5.2 when \( \delta = 1/BF \) the geometric version is equivalent to the branching version. The synthetic data used in the experiments reported in Fig-
5.4. Inverse Fisheye Heuristic

Figure 5.13 (a) was generated with $BF = 3$, however, the two heuristics present a different number of iterations for $\delta = 1/3 = 0.33$. This occurs because $BF = 3$ is a parameter for the synthetic data generation model which does not correspond exactly to the resulting branching factor of the generated grammar. In fact, the generated grammar has some productions which are assigned probability 0 and, as such, are eliminated, leading to a branching factor marginally smaller than the specified by the parameter.

![Graph showing performance of IFE heuristic with synthetic and real data](image)

(a)

Figure 5.13: The performance of the inverse fisheye heuristic with synthetic data and comparison between the rule-sets obtained with the inverse fisheye and the BFS algorithm for real data.

Figure 5.13 (b) provides a comparison between the size and the ARL of the rule-sets induced by the IFE heuristic with the corresponding rule-sets induced by the BFS algorithm; these results are for real data sets. In an ideal situation we expect the IFE to achieve a smaller rule-set with higher ARL. This is not the case with the branching factor version, where in spite of inducing a rule-set with higher ARL the number of rules obtained is too large. On the other hand, the results obtained with the geometric version are promising, although the number of rules obtained is close to the obtained with the BFS algorithm the ARL is in general larger. The results suggest that the branching version is probably too permissive by making the cut-point decrease at a too fast rate. Both plots on Figure 5.13 correspond to a maximum exploration depth of $\bar{d} = 6$.

Figure 5.14 (a) shows the variation of the number of rules obtained for different
values of the maximum depth of exploration when using the geometric version of the IFE heuristic. In addition, Figure 5.14 (b) shows the variation of the average rule length with the maximum exploration depth. The results show that it is possible to have some control over the number of rules induced by setting the value of both the dynamic cut-point and of the maximum depth of exploration. In fact, an increase in the value of the exploration depth results in a proportional increase on the number of rules induced and the proportionality is stable across the range of the depth of exploration.

Figure 5.14: Evolution of the characteristics of the rule-set obtained with the geometric version of the inverse fisheye heuristic with real data.

Figure 5.15 shows the distribution of the rules' length for a set of randomly generated grammars. As usual with synthetic data, each result corresponds to the average of 30 runs for a specific configuration. The results show that if the initial cut-point is not too low the IFE heuristic induces a smaller number of rules with larger average rule length than the BFS algorithm. For example, the geometric version with \( \delta = 0.4 \) induces some long rules while reducing the rule-set size to almost half of that given by the BFS algorithm. Moreover, although the branching version induces rule-sets with significantly larger average rule length, the size of the rule-set obtained is, in general, too large to be useful.

Figure 5.16 shows the distribution of the rules' length for the real data. For this set of experiments the support threshold was set at \( \theta = 1/n \) and each result corresponds
5.4. Inverse Fisheye Heuristic

Figure 5.15: Distribution of the rules’ length with synthetic data.

Table 5.1: Distribution of the rules’ length with synthetic data.

<table>
<thead>
<tr>
<th>δ</th>
<th>Algorithm</th>
<th>Tot.</th>
<th>Avg.</th>
<th>Rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.3</td>
<td>BFS</td>
<td>1043</td>
<td>1.4</td>
<td>624</td>
</tr>
<tr>
<td>0.3</td>
<td>Geometric</td>
<td>1425</td>
<td>4.5</td>
<td>209</td>
</tr>
<tr>
<td>0.3</td>
<td>Branching</td>
<td>12548</td>
<td>5.9</td>
<td>209</td>
</tr>
<tr>
<td>0.4</td>
<td>BFS</td>
<td>926</td>
<td>1.1</td>
<td>869</td>
</tr>
<tr>
<td>0.4</td>
<td>Geometric</td>
<td>466</td>
<td>1.2</td>
<td>405</td>
</tr>
<tr>
<td>0.4</td>
<td>Branching</td>
<td>2201</td>
<td>5.1</td>
<td>405</td>
</tr>
<tr>
<td>0.5</td>
<td>BFS</td>
<td>882</td>
<td>1.0</td>
<td>869</td>
</tr>
<tr>
<td>0.5</td>
<td>Geometric</td>
<td>465</td>
<td>1.1</td>
<td>451</td>
</tr>
<tr>
<td>0.5</td>
<td>Branching</td>
<td>796</td>
<td>3.1</td>
<td>451</td>
</tr>
<tr>
<td>0.6</td>
<td>BFS</td>
<td>817</td>
<td>1.0</td>
<td>816</td>
</tr>
<tr>
<td>0.6</td>
<td>Geometric</td>
<td>459</td>
<td>1.0</td>
<td>458</td>
</tr>
<tr>
<td>0.6</td>
<td>Branching</td>
<td>503</td>
<td>1.3</td>
<td>458</td>
</tr>
</tbody>
</table>

The analysis of the structure of the HPGs inferred from the real log files revealed that, although the average branching factor (which is 5.2) is similar to the specified for the synthetic data sets, the real data set presents a much higher average standard deviation (which is 9). In addition, in the HPGs inferred from the real data there is an average of 26% states with a single out-link to a state other than the final state $F$. Such states, when included in a trail, increase the length of the trail without decreasing its probability. Since the transition between such states is certain it does not increase the knowledge about the navigation behaviour. In fact, they correspond to trails in which the user faced few navigation options and rules including such states should be analysed critically. This fact helps to explain why the BFS algorithm induces relatively long rules for high values of the cut-point.
### 5.4. Inverse Fisheye Heuristic

#### Figure 5.16: Distribution of the rules' length for real data with $\theta = 1/n$.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>Algorithm</th>
<th>Tot.</th>
<th>Avg.</th>
<th>Rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.4</td>
<td>BFS</td>
<td>1646</td>
<td>2.0</td>
<td>466</td>
</tr>
<tr>
<td>0.4</td>
<td>Geometric</td>
<td>3586</td>
<td>4.5</td>
<td>78</td>
</tr>
<tr>
<td>0.5</td>
<td>BFS</td>
<td>1289</td>
<td>1.9</td>
<td>414</td>
</tr>
<tr>
<td>0.5</td>
<td>Geometric</td>
<td>1623</td>
<td>3.4</td>
<td>131</td>
</tr>
<tr>
<td>0.6</td>
<td>BFS</td>
<td>1090</td>
<td>1.9</td>
<td>373</td>
</tr>
<tr>
<td>0.6</td>
<td>Geometric</td>
<td>1093</td>
<td>2.7</td>
<td>159</td>
</tr>
<tr>
<td>0.7</td>
<td>BFS</td>
<td>933</td>
<td>1.8</td>
<td>335</td>
</tr>
<tr>
<td>0.7</td>
<td>Geometric</td>
<td>871</td>
<td>2.3</td>
<td>188</td>
</tr>
<tr>
<td>0.9</td>
<td>BFS</td>
<td>714</td>
<td>1.78</td>
<td>276</td>
</tr>
<tr>
<td>0.9</td>
<td>Geometric</td>
<td>681</td>
<td>1.87</td>
<td>234</td>
</tr>
<tr>
<td>0.9</td>
<td>Branching</td>
<td>14749</td>
<td>5.8</td>
<td>234</td>
</tr>
<tr>
<td>0.95</td>
<td>BFS</td>
<td>666</td>
<td>1.77</td>
<td>260</td>
</tr>
<tr>
<td>0.95</td>
<td>Geometric</td>
<td>655</td>
<td>1.81</td>
<td>243</td>
</tr>
<tr>
<td>0.95</td>
<td>Branching</td>
<td>13311</td>
<td>5.7</td>
<td>243</td>
</tr>
</tbody>
</table>

#### Figure 5.17: Distribution of the rules' length for real data with $\theta = 2/n$.

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>Algorithm</th>
<th>Tot.</th>
<th>Avg.</th>
<th>Rule length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>0.4</td>
<td>BFS</td>
<td>814</td>
<td>1.8</td>
<td>306</td>
</tr>
<tr>
<td>0.4</td>
<td>Geometric</td>
<td>1213</td>
<td>4.0</td>
<td>56</td>
</tr>
<tr>
<td>0.5</td>
<td>BFS</td>
<td>632</td>
<td>1.8</td>
<td>252</td>
</tr>
<tr>
<td>0.5</td>
<td>Geometric</td>
<td>626</td>
<td>2.9</td>
<td>80</td>
</tr>
<tr>
<td>0.6</td>
<td>BFS</td>
<td>520</td>
<td>1.7</td>
<td>216</td>
</tr>
<tr>
<td>0.6</td>
<td>Geometric</td>
<td>455</td>
<td>2.3</td>
<td>97</td>
</tr>
<tr>
<td>0.7</td>
<td>BFS</td>
<td>445</td>
<td>1.7</td>
<td>195</td>
</tr>
<tr>
<td>0.7</td>
<td>Geometric</td>
<td>384</td>
<td>2.0</td>
<td>113</td>
</tr>
</tbody>
</table>
5.5 Summary

In this chapter we have presented two new heuristics to mine for patterns in web navigation sessions, which are modelled as a hypertext probabilistic grammar. The first heuristic, namely the Fine-grained heuristic, implements an iterative deepening search that incrementally builds the rule-set. A stopping parameter was provided which is shown to give the analyst good control over the number of mined rules. Two versions of the heuristic were proposed which differ in the way the next trail to be expanded is chosen. The best-trail version chooses to expand first the link that will lead to the best new trail; we proved that this version induces, at each stage, the set of trails whose sum of probabilities is maximum for a given depth of exploration. The greedy version chooses to expand first the trail which will have appended the link with highest probability. Experimental results showed that the greedy version is able to induce long rules even for high values of the stopping parameter.

The second heuristic, namely the inverse fisheye heuristic, makes use of a dynamic cut-point which is very strict when evaluating short trails and becomes more permissible as the trails get longer. Two versions for setting the dynamic cut-point were proposed. The branching version takes into account the average branching factor of the underlying hypertext system, and consequently the average link probability, when setting the cut-point value; this version was shown to be, in general, too permissive. The geometric version keeps the cut-point value proportional to the length of the trail being explored. We proved that this version induces a set of rules whose average link probability is greater than or equal to the confidence threshold component of the cut-point.
Chapter 6

Operations on Hypertext Grammars

In this chapter we present a set of binary operations on hypertext probabilistic grammars, whose aim is to provide the analyst with techniques for comparing the contents of two different grammars. The operations we define are sum, union, difference, intersection, and complement. These operations can be useful for the analysis of log files by extending the capabilities of the HPG model. For example, the operations provide the means to assess the evolution over time of grammars inferred from log data or to compare the contents of grammars characterising the behaviour of different classes of users.

6.1 Motivation

In this chapter we propose a set of binary operations for comparing the contents of two hypertext probabilistic grammars. The operations on grammars can be useful both to the analyst who studies the contents of web server log files and to the individual user who has his browser set up to collect a personal log file. The operations we define are sum, union, difference, intersection, and complement.

The operations on grammars are useful for the analysis of web server log files by enabling the analyst to assess the evolution over time of user navigation patterns within the site. For example, the analyst is given the means to incrementally build weekly HPGs and compute the difference between them, in order to assess how the navigation patterns change over time. This analysis can provide some insight as to whether or not a modification on the web site design has achieved the intended results. Further-
more, the operations give the analyst increased flexibility to conduct selective analysis on the log data. For example, if a grammar is inferred from two distinct log files it is not possible, in the general case, to separate its contents into two grammars such that each corresponds exactly to one of the files. But, if a grammar is inferred separately from each of the two log files, the sum of such grammars results in a single grammar representing the information contained in both files. This way, the analyst can analyse the log files either individually or together as a single log file. In addition, the operations give the means to compute the difference between a generated grammar characterising the expected user behaviour when visiting the web site and a grammar inferred from log data that characterises the actual user behaviour. The result of such analysis may provide insight as to whether or not the navigational aspects of the web site design are achieving their purposes. Finally, the operations of difference and intersection provide a measure of the degree of overlapping between grammars. Therefore, such operations can be useful to help identify clusters of grammars with similar contents.

The operations on grammars can also be useful for the individual user in a scenario where personal log files are collected from a group of people with a common interest, such as a research group in a university department. Such log files can be collected by setting up a local proxy recording all web page requests from the browsers of the group's members. In such a scenario an individual benefits from having a personal HPG that keeps track of his favourite trails on the web. In addition, the research group as a whole benefits if users are able to share their browsing experience. By having access to other peoples HPGs the users are enabled to identify the preferred trails of their fellow users. The analysis can be further enhanced by making use of the grammar operations. For example, the intersection between the grammars of two users gives the subset of the web that is common to them. Such a grammar can be further analysed and its high probability trails identified by using the algorithms presented in the previous chapters. The grammar resulting from the union or the sum of two grammars enables the identification of the high probability trails in a grammar that represents the overall users' navigation history. Finally, the difference between two grammars is useful to discover trails that are among the favourites of one user but are unknown to the other user.
6.2 Inferring a HWG from a HPG

The grammar operations proposed in this chapter are defined on hypertext weighted grammars (HWG), see Definition 3.4, rather than on the corresponding HPGs. By making use of HWGs it is possible to overcome the difficulty in accurately computing, for example, the intersection of probabilities, especially in the cases where they don’t correspond to the same number of events. Consider, for example, a page that has a link which was chosen three times out of ten page visits by a given user, i.e., with probability 0.3. In addition, consider that another user chose the same link in two out of fifteen page visits, i.e., with probability 0.13. In this case, we can say that the page was visited at least ten times by any user and the link was traversed at least two times. Similar reasoning is not possible to be carried out with probabilities. Furthermore, in the general case a HPG is derived by the normalisation of a HWG, which means that the characterisation of the corresponding HWG is available.

As will be shown in this section, for every HPG it is possible to determine a HWG whose normalisation originates a HPG equivalent to the original one. This property extends the application of the grammar operations to random HPGs, which are generated according to a specified probability distribution. Therefore, it enables the analyst to compare a grammar representing the actual user behaviour with a generated HPG following a specified probability distribution. For example, it is useful to compare a grammar representing the user behaviour with a grammar that has all links following a uniform probability distribution of being chosen. In this chapter, unless otherwise stated, the word grammar refers to a HWG.

A HWG has the same structure as the corresponding HPG, with the difference that a production is characterised by an integer representing the number of traversals rather than by a probability. (Note that we use the terms link and production interchangeably.) The sum of the weights of the in-links to a state represents the state degree, and is equal to the sum of the weights of the out-links from that state. The state degree gives the number of times the corresponding page was requested and is denoted by \(|A_t|\), where \(A_t\) is a grammar state. The weight of a transitive production gives the number of times the corresponding link was traversed and is denoted by \(|A_tA_j|\). Finally, the degree of
the start state, \(|S|\), is equal to the degree of the final state, \(|F|\), and represents the number of sessions modelled by the grammar. Remember that both \(S\) and \(F\) are artificial states representing the start and finish of a session respectively.

One method to determine a HWG whose normalisation corresponds to a given HPG consists in the following steps:

1. For each state corresponding to a page, \(A_i \in V - \{S, F\}\), write the equation of its degree, that is, an equation stating that the sum of the weights of the in-links is equal to the state degree;
2. Solve the system of equations obtained in step 1 as a function of the degree of the start state, \(|S|\);
3. Determine the minimum value for \(|S|\) which gives an integer solution for both all the states' degree and all the links' weight.

In Figure 6.1 we give an example of a generic HPG. The system of equations for the states' degree is as follows:

\[
\begin{align*}
|A_1| &= a|S| + c|A_2| \\
|A_2| &= (1-a)|S| + b|A_1|
\end{align*}
\]

In Figure 6.2 (a) we give an example of a HPG grammar whose derived HWG is given in Figure 6.2 (b). By solving the system of equations we obtain the degree of...
6.2. Inferring a HWG from a HPG

Figure 6.2: An example of a HPG and its corresponding HWG.

each state as a function of the number of sessions, $|S|$. As it is shown by the following expressions, $|S|$ has to be a multiple of 21 in order for every state degree to be an integer value.

$$
\begin{cases}
|A_1| = \frac{1}{3}|S| + \frac{1}{4}|S| = \frac{10}{21}|S| \\
|A_2| = (1 - \frac{1}{3})|S| + \frac{1}{4}|S| = \frac{18}{21}|S|
\end{cases}
$$

In addition, we have to verify which value of $|S|$ results in having an integer weight assigned to every link. Since all the values of the weights in a HWG are rational, it is always possible to find a value for $|S|$ which results in an integer value for every state degree and link weight.

$$
\begin{align*}
|S A_1| &= \frac{1}{3}|S| \\
|A_1 A_2| &= \frac{1}{4}|A_1| = \frac{1}{31}|S| \\
|A_2 A_1| &= \frac{1}{3}|A_2| = \frac{9}{31}|S| \\
|S A_2| &= \frac{2}{3}|S| \\
|A_1 F| &= \frac{3}{4}|A_1| = \frac{12}{21}|S| \\
|A_2 F| &= \frac{1}{2}|A_2| = \frac{9}{21}|S|
\end{align*}
$$

Figure 6.2 (b) shows the HWG derived from the HPG in Figure 6.2 (a) when setting $|S| = 21$. Following the example we state the following proposition.

**Proposition 6.1.** For any HPG with $n = 2$ states there is a positive and integer solution for the system of equations of the states’ degree.

**Proof.**

1. $0 \leq a, b, c \leq 1$ because they are probabilities;
2. $|S| > 0$ and integer because it represents the number of sessions;
3. $b < 1$ or $c < 1$ in order to have at least one path to $F$, otherwise the HPG is invalid. Therefore, $(1 - cb) > 0$.
4. If $a = 0$ and $c = 0$, from Equation 6.1 we have that $|A_1| = 0$ and $|A_2| = |S|$.

In this case every state has a non-negative degree.
5. IF $a > 0$ or $c > 0$ from Equation 6.1 we have that $|A_1| > 0$ since $a + c > ac$.

6. From 4 and 5 it follows that $|A_1|$ is always non-negative.

7. In addition, from Equation 6.1 we have that $|A_2| \geq 0$ since $|A_1|, |S|, b, (1-a) \geq 0$;

8. Finally, since all numbers are rational and $|A_1|, |A_2|$ are proportional to $|S|$ there is always a large enough $|S|$ such that the degree of states $|A_1|$ and $|A_2|$ are integer values.

Similarly, there is a large enough $|S|$ such that the weight of every link is an integer because the weight of a link is proportional to the degree of anchor state. Therefore, we have that $a|S|, (1-a)|S|, b|A_1|, (1-b)|A_1|, c|A_2|, (1-c)|A_2|$ are all integers. □

For the general case of a grammar with $n$ states we do not have an analytical proof that there is always a solution for the system of equations. However, we provide an intuitive proof that there is a HWG corresponding to every HPG.

**Proposition 6.2.** For any HPG with $n$ states there is always a HWG whose normalisation corresponds to the HPG.

**Intuitive proof.**

1. Generate a collection of $x$ random trails according to the probabilities of the initial HPG;

2. Compute the HWG inferred from the collection of generated trails;

3. Normalise the HWG;

4. By the law of large numbers for Markov chains, if $x$ is large enough the HPG obtained from the collection of random trails follows the same probability distribution as the original HPG. □

This result can be generalised one step further as we consider the concept of a *unbalanced* HWG (or simply uHWG). A uHWG is a HWG that is not balanced, that is, it contains at least one state whose sum of the weights of the in-links is not equal to the sum of the weights of the out-links. A uHWG must have the property of being a reduced grammar, that is, every state is included in a path from the start to the final state.
6.2. Inferring a HWG from a HPG

The uHWG concept is useful because it is easier to generate a random uHWG than a random HWG. For example, if we assign the same weight to every link the resulting grammar may be unbalanced, unless in every state the number of out-links is equal to the number of in-links. Random grammars are essential to study the HPG model behaviour with different configurations, see Chapter 4. In addition, random HWGs or uHWGs can be set to represent specific models of user behaviour, for example, by assigning the same weight to every link in the grammar. Such random models may be useful for the understanding of the actual user behaviour by providing a reference model with which the analyst can compare the user's grammar.

There is always a HWG corresponding to a uHWG. In fact, the normalisation of a uHWG results in a HPG and since there is always a HWG corresponding to every HPG there is also a HWG corresponding to a uHWG. Therefore, a uHWG is valid as a model of user navigation sessions.

Figure 6.3 summarises the relationships between the three classes of grammars considered in this chapter. A plain line denotes a transformation for which an algorithm is available and a dashed line denotes a trivial transformation. Note that, it makes no sense to convert a uHWG to a HWG directly. In such conversion it is essential to maintain the proportions among the weights of the out-links from a state, therefore, it is necessary to consider a HPG as an intermediate step. Finally, the conversions of a HPG into a uHWG and of HWG into a uHWG are trivial, assuming the probabilities are rational.

![Figure 6.3: The relationship between HPGs, HWGs, and uHWGs.](image-url)
6.3 Grammar Equality and Sub-grammars

In this section we present the concepts of \textit{grammar equality} and \textit{sub-grammar}. In the sequel we let $G_1$ and $G_2$ be two HWGs, and for a grammar $G_k$ we let $V_k = \{S, A_1, \ldots, A_n, F\}$ be its set of nonterminal symbols, $\Sigma_k = \{a_i, \ldots, a_n\}$ be its alphabet, and $P_k$ be its set of productions. Moreover, $A_iA_j \in P_k$ represents a link corresponding to a transitive production and $|A_iA_j|_k$ the weight of such link in grammar $G_k$.

Two HWGs are equal if and only if they have exactly the same composition. The formal definition of grammar equality now follows.

\textbf{Definition 6.3 (Grammar equality).} Two HWG are equal, $G_1 = G_2$, if and only if:

1. $V_1 = V_2$;
2. $\Sigma_1 = \Sigma_2$;
3. $P_1 = P_2$;
4. $\forall A_iA_j \in P_1$ and $P_2$ we have that $|A_iA_j|_1 = |A_iA_j|_2$.

A grammar $G_1$ is said to be a sub-grammar of a grammar $G_2$ if and only if all the elements in $G_1$ are also elements of $G_2$, and the weight of a production in $G_1$ is less or equal than the weight of the corresponding production in $G_2$. The formal definition of sub-grammar now follows.

\textbf{Definition 6.4 (Sub-grammar).} A grammar $G_1$ is a sub-grammar of a grammar $G_2$, $G_1 \subseteq G_2$, if and only if:

1. $V_1 \subseteq V_2$;
2. $\Sigma_1 \subseteq \Sigma_2$;
3. $P_1 \subseteq P_2$;
4. $\forall A_iA_j \in P_1$ we have that $|A_iA_j|_1 \leq |A_iA_j|_2$.

Furthermore, a grammar $G_1$ is said to be a \textit{proper sub-grammar} of a grammar $G_2$ if $G_1$ is a sub-grammar of $G_2$ but is not equal to $G_2$, that is:

$$G_1 \subset G_2 \equiv G_1 \subseteq G_2 \text{ and } G_1 \neq G_2.$$
In Figures 6.4 (a), (b) and (c) we give three examples of HWGs where $G(a) \neq G(b)$, $G(c) \neq G(a)$, $G(b) \subseteq G(c)$, $G(c) \not\subseteq G(a)$, and $G(c) \not\subseteq G(b)$.

Note that the following properties hold with every HWGs:

1. $G_1 \subseteq G_1$, that is, a grammar is a sub-grammar of itself;
2. $G_1 = G_2 \equiv G_1 \subseteq G_2$ and $G_2 \subseteq G_1$;
3. Given three grammars, $G_1$, $G_2$ and $G_3$ we have that $G_1 \subseteq G_2$ and $G_2 \subseteq G_3 \Rightarrow G_1 \subseteq G_3$, that is, the sub-grammar property is transitive.

### 6.4 Primitive Methods

We now present two primitive methods on HWGs which provide the foundations for the definition of the grammar operations. When deriving the grammar resulting from the application of an operation on two grammars there is a need of methods to remove and insert link traversals into a grammar. One essential characteristic of such methods is that after a link is inserted or removed the resulting grammar must continue to be a valid HWG. Remember that in a HWG every state is included in a path from the start state to the final state, and the sum of the weights of the in-links to a state is equal to the sum of the weights of the out-links from that same state.

The first method consists in incrementing the weight of a grammar's production. Consider the example of the HWG in Figure 6.5 (a), such grammar can be inferred from the following navigation sessions:

\begin{align*}
(1) \ & S A_1 A_2 A_3 A_1 F \\
(2) \ & S A_2 A_3 A_1 A_2 F.
\end{align*}
If a new link traversal is inserted into the grammar, and the weight of the corresponding production incremented, several different interpretations are possible. One interpretation can be that the new link traversal corresponds to a complete new session, another interpretation can be that the new traversal is a prefix or a suffix of an existing session. For example, when the weight of link $A_1A_2$ in the grammar of Figure 6.5 (a) is incremented it can be interpreted as if the two initial sessions were concatenated, that is, the first session is a prefix of the new traversal and the second session a suffix of the new traversal. In this case, the two sessions are merged into a single longer session and the corresponding HWG is given in Figure 6.5 (b). Note that there are now three traversals from state $A_1$ to state $A_2$, and link $A_1F$ was deleted because the session which was terminating in $A_1$ is now considered to continue to $A_2$. Moreover, link $SA_2$ was deleted since the session which was starting in $A_2$ is now considered to be a prolongation of the other session.

![Figure 6.5: Two alternative interpretations when adding a link traversal to a HWG.](image)

An alternative interpretation for the above situation is to consider that the new link traversal corresponds to a new session. Figure 6.5 (c) gives the grammar corresponding to this case. Note that the weight of link $SA_1$ was incremented to represent the start of the new session and link $A_2F$ was incremented to represent the finish of the session. Other similar interpretations can be adopted. In this chapter’s figures the links represented in bold style are those whose weight was altered but which remained in the
grammar; those links whose weight was reduced to zero are not represented.

To formally define the primitive method for incrementing the weight of a link we adopt the interpretation that a new link traversal is a new session. Therefore, when incrementing the weight of the link $A_iA_j$ both $SA_i$ and $A_jF$ have their weights incremented by the same amount.

Note that both interpretations described above guarantee that states $A_i, A_j, S$ and $F$ are kept balanced. However, only assuming that a new traversal corresponds to a new session guarantees to keep every state included in a path from the state $S$ to the state $F$, because all existing paths from $S$ to $F$ are maintained and an alternative new path is introduced. Merging existing sessions can lead to the elimination of essential transitions from $S$ or to $F$.

In Algorithm 6.1 we present the pseudo-code for incrementing the weight of a link $A_iA_j$ by a factor of $g$. As usual, for a state $A_i$ we let $|A_i|$ be the state degree, $|SA_i|$ be the number of transitions from the start state, and $|A_iF|$ be the number of transitions to the final state. Moreover, we let $|A_iA_j|$ be the weight of a transitive production.

Algorithm 6.1: The pseudo-code for the method to increment the weight of a link in a HWG.

```
IncrementLinkWeight (A_iA_j, g)
1. begin
2.   for i = 1 to g
3.     |A_iA_j| = |A_iA_j| + 1;
4.     |SA_i| = |SA_i| + 1;
5.     |A_jF| = |A_jF| + 1;
6.   end for
7. end.
```

The second primitive method consists in decrementing the weight of a link. As with the previous method, when a link traversal is removed from a HWG several interpretations are possible. For example, when the weight of link $A_1A_2$ in the grammar of Figure 6.6 (a) is decremented it can be interpreted as if such a traversal belongs to the beginning of the first navigation session. In addition, it can be considered that the re-
remaining part of the session can be appended to the end of the second session and form a single longer session. In this case, the resulting grammar is the one shown in Figure 6.6 (b). Note that link $SA_1$, which corresponded to the start of the first session, was eliminated as well as the link $A_2F$, which corresponded to the end of the second session.

![Figure 6.6](image)

Figure 6.6: Two alternative interpretations when removing a link traversal from a HWG.

An alternative interpretation is to consider that when a traversal of link $A_1A_2$ is removed from the HWG the first session is subdivided into two shorter sessions. One session composed of a single state $SA_1F$, and the second composed by the remaining part of the original session, $SA_2A_3A_1F$. This interpretation results in the grammar of Figure 6.6 (c).

In this work we adopt the interpretation that whenever a link traversal is removed from a grammar an existing navigation session is split into two smaller sessions. Therefore, when decrementing the weight of link $A_iA_j$ both $A_iF$ and $SA_j$ have their weights incremented by the same amount (as in Figure 6.6 (c)). Note that the weight of a link is always greater than or equal to zero, therefore, a link cannot have its weight decremented by a factor greater than its current weight.

As with the previous primitive method, this method keeps every state balanced and included in at least one path from $S$ to $F$. The same would not be always true if a different interpretation was adopted. In Algorithm 6.2 we present the pseudo-code for
the method to decrement the weight of a link by a factor of \( g \).

\textbf{DecrementLinkWeight} (\( A_iA_j, \ g \))

1. \textbf{begin}
2. \quad \textbf{for} \( i = 1 \) \textbf{to} \( g \)
3. \quad \quad \textbf{if} \ |A_iA_j| > 0 \textbf{ then}
4. \quad \quad \quad |A_iA_j| = |A_iA_j| - 1;
5. \quad \quad \quad |A_iF| = |A_iF| + 1;
6. \quad \quad \quad |SA_j| = |SA_j| + 1;
7. \quad \quad \textbf{end if}
8. \quad \textbf{end for}
9. \textbf{end}.

Algorithm 6.2: The pseudo-code for the method to decrement the weight of a link in a HWG.

6.5 Binary Operations on Grammars

In this section we present a collection of the binary operations on HWGs. The operations we define are sum, union, difference, intersection, and complement. For each operation we provide its definition and the pseudo-code for the algorithm to obtain the resulting grammar.

The computation of the output grammar from the application of a binary operation on two input grammars consists of a two stage approach. In the first stage the reachability relation of the output grammar is constructed, assigning to each transitive production the weight corresponding to the operation being implemented. (Recall that the reachability relation of a grammar, \( \mathcal{R} \), is defined by its set of transitive productions.) Only the primitive operations defined in Section 6.4 are used in the first stage in order to guarantee that every instance of the grammar being transformed is a valid HWG. In the second stage of the process, the degree of each state of the grammar resulting from the first stage is corrected, in order to get the appropriate value. A state \( A_i \) can have its degree increased by incrementing the weight of productions \( SA_i \) and \( A_iF \) by a constant factor, and to reduce the degree of a state the same productions have their weight decremented. Increasing the degree of a state corresponds to inserting into the gram-
mar a session composed of a single page visit. Note that the interpretations adopted in the definitions of the primitive methods tend to over-evaluate the degree of a state and, therefore, justifies the need of a stage aimed at correcting the states’ degree.

The computation of the reachability relation is accurate with every operation. In fact, the reachability relation of the output grammar follows the definitions for the union, intersection, and difference of finite automata, see (Martin, 1997). The union and sum of HWGs produce the same reachability relation and the difference between the operations is the weights attributed to the productions. In a HWG there is a one-to-one mapping between the set of nonterminals and the set of terminal symbols. Therefore, the output grammar of a binary operation has the same states and productions that exist in the input grammars. Note that with a standard finite automaton, for example, the union of two grammars with \( n_1 \) and \( n_2 \) states respectively results in a grammar with \( n_1 + n_2 \) states \( (n_1 \times n_2 \) in the intersection).

The weights of the transitive productions are estimated following the theory of operations on bags, see (Nissanke, 1999). A bag is a collection of objects where multiple occurrences of a given object are allowed. Given two bags, the cardinality of an element in their union is given by its maximal cardinality in the two bags. The cardinality of an element in the intersection of two bags is given by the minimum cardinality in the two bags. In the sum and difference the cardinality is computed by the sum and difference of the cardinalities in the two bags respectively.

The computation of the degree of a state in the grammar resulting from an operation follows the same principles. In both the sum and union of two input HWGs the computation of the degree of a state in the output grammar is well defined. However, with the difference and intersection the computation of the degree of a state is not obvious and some assumptions are necessary. These assumptions will only affect the probability of starting and finishing a navigation session at a given page. Note that the start probability of a page in a HPG is a measure which can be weighted when performing the normalisation of a HWG by making use of the \( \alpha \) parameter (see Definition 3.6).

Each binary operation results in a unique output grammar given any two input grammars. In the sequel we let \( G_1 \) and \( G_2 \) be two input HWGs and \( G \) be the output
6.5. Binary Operations on Grammars

The method \( G_k.copy() \) returns a copy of a grammar \( G_k \), \( \min(x, y) \) returns the minimum of the two values \( x \) and \( y \), and \( \max(x, y) \) returns the maximum of the same values. Moreover, \( A_i \) is a state in \( G \), \( (A_i)_k \) corresponds to an instance of the same state in \( G_k \), and \( (A_iA_j)_k \) represents an instance of production \( A_iA_j \) in \( G_k \). Moreover, we let \( \overline{A_i} \) be the sum of the weights of the transitive productions whose left-hand side is \( A_i \), and \( A_i \) be the sum of the weights of the transitive productions whose right-hand side is \( A_i \); \( A_i = \overline{A_i} \) holds.

The sum of two grammars

The definition of the sum of two grammars now follows.

**Definition 6.5 (Grammar sum).** The sum of two HWGs, \( G = G_1 + G_2 \), is defined as the grammar containing those states and productions belonging to either grammar,

\[
A_i \in G \text{ if and only if } (A_i)_1 \in G_1 \text{ or } (A_i)_2 \in G_2 \text{ and }
A_iA_j \in G \text{ if and only if } (A_iA_j)_1 \in G_1 \text{ or } (A_iA_j)_2 \in G_2.
\]

The weight of a transitive production is given by the sum of the weights of the corresponding link in \( G_1 \) and \( G_2 \),

\[
|A_iA_j| = |A_iA_j|_1 + |A_iA_j|_2.
\]

The degree of a state is given by the sum of the degrees of the corresponding states in the original grammars,

\[
|A_i| = |A_i|_1 + |A_i|_2.
\]

Figure 6.7 gives two HWGs which are used as a running example throughout this section to illustrate the operations on grammars.

Algorithm 6.3 gives the pseudo-code for the method to compute the sum of two grammars and Figure 6.8 illustrates the sum of the two input grammars given in the running example of Figure 6.7. The algorithm starts with a copy of one of the input grammars, in this case \( G_1 \), in which all the link traversals from the other input grammar, \( G_2 \), are inserted. Figure 6.8 (a) shows grammar \( G \) which consists of a copy of \( G_1 \) in which \( |A_1A_2|_2 \) is inserted. In Figure 6.8 (b) traversal \( |A_2A_3|_2 \) is inserted in \( G \) and in (c) both \( |A_3A_2|_2 \) and \( |A_1A_3|_2 \) are inserted in \( G \). Finally, in Figure 6.8 (d) the degrees of states \( A_2 \) and \( A_3 \) are corrected in order to be \( |A_2| = |A_2|_1 + |A_2|_2 = 6 \) and
Figure 6.7: Example of two HWGs to illustrate the operations.

\[ |A_3| = |A_3|_1 + |A_3|_2 = 6 \]

and the output grammar representing the sum of the two input grammars is finally obtained.

![Diagram of G1 and G2](image)

Figure 6.8: Example of the computation of the sum of two HWGs.

One interesting property of the sum operation is that the sum of two input grammars, each inferred from a different log file, is equal to a grammar inferred from the information contained in the two log files. This property holds because the weight of a link in a HWG represents the number of its traversals given by the log file, and the degree of a state represents the number of times a request of the corresponding page occurs in the log file. Since the sum of two HWGs is defined as both the sum of the
6.5. Binary Operations on Grammars

**GrammarSum** \((G_1, G_2)\)

1. begin
2. \( G = G_1.copy(); \)
3. for every \((A_i, A_j) \in G_2\)
4. \( G.IncrementLinkWeight(A_i, A_j, |A_i| + |A_j|); \)
5. end for
6. for every \(A_i \in G\)
7. if \(|A_i| > |A_i|_1 + |A_i|_2\) then
8. \( G.DecrementLinkWeight(SA_i, |A_i| - (|A_i|_1 + |A_i|_2)); \)
9. else if \(|A_i| < |A_i|_1 + |A_i|_2\) then
10. \( G.IncrementLinkWeight(SA_i, (|A_i|_1 + |A_i|_2) - |A_i|); \)
11. end if
12. end for
13. end.

Algorithm 6.3: The pseudo-code for the method to compute the sum of two HWGs.

weights of the links and the sum of the degrees of the states in the two input grammars, the result is equal to a grammar inferred from the information contained in both log files. We will now state this property as a proposition.

**Proposition 6.6.** Given two grammars, \(HPG_1\) inferred from a log file \(f_1\) and \(HPG_2\) inferred from a log file \(f_2\), the sum of the two grammars \(HPG_1 + HPG_2\) is equal to a grammar inferred from the information contained in a file resulting from combining \(f_1\) with \(f_2\).

This property gives the analyst the ability to store log data information in different HWGs and compute their sum whenever the aim is to analyse a HPG representing all the navigation information. Note that the opposite is not possible, that is, once a grammar is inferred from several log files it is not possible, in general, to split the HWG into several HWGs each corresponding to exactly one of the log files.

With the sum operation the reachability relation of the output grammar represents the subset of the web modelled by either grammar. The weight of a production is given by the total number of times the corresponding link was traversed by the users. Therefore, such operation is useful, for example, to analyse the overall navigation behaviour of a group of users, each user having a HWG characterising his preferences.
The union of two grammars

The grammar union is similar to the grammar sum. In fact, the reachability relation of the output grammar given by the union is equal to the given by the sum. The difference between the two operations is the weights assigned to the productions. With the union the weight of a transitive production represents the maximum number of times a user traversed the corresponding link. In addition, the degree of a state is given by the maximum value between the degrees of the corresponding states in the input grammars.

The definition of the union of two grammars now follows.

**Definition 6.7 (Grammar union).** The union of two HWG, $G = G_1 \cup G_2$, is defined as the grammar containing those states and transitive productions that belong to either grammar,

$$A_i \in G \text{ if and only if } (A_i)_1 \in G_1 \text{ or } (A_i)_2 \in G_2$$

$$A_iA_j \in G \text{ if and only if } (A_iA_j)_1 \in G_1 \text{ or } (A_iA_j)_2 \in G_2.$$  

The weight of a transitive production is defined as the maximum number of times the corresponding link was traversed by any of the users,

$$|A_iA_j| = \max(|A_iA_j|_1, |A_iA_j|_2).$$

The degree of a state in the output grammar is equal to the maximum degree of the degrees of the corresponding states in the input grammars, unless a larger value is imposed by the weights of the transitive productions of the output grammar,

$$|A_i| = \max\left(\max(|A_i|_1, |A_i|_2), \max(\overline{A_iA_i})\right).$$

Algorithm 6.4 gives the pseudo-code for the method to compute the union of two grammars and Figure 6.9 illustrates the computation of the union of the input grammars given in the running example of Figure 6.7. The algorithm starts by assigning a copy of $G_1$ to $G$, in which link traversals are inserted until each production in $R$ is assigned the required weight. Figure 6.9 (a) shows $G$ after $|A_1A_3|_2$ is inserted in it. In Figure 6.9 (b) traversal $|A_3A_2|_2$ is inserted in $G$. At this stage the transitive productions have the required weights but the degree of each state has to be verified. The degree of state $A_1$ has the required value, which is $|A_1| = \max(\max(2,3), \max(1,4)) = 4$. However, states $A_2$ and $A_3$ must have their degrees corrected to $|A_2| = \max(\max(4,2), \max(3,3)) = 4$.
and \(|A_3| = \max(\max(3, 3), \max(5, 2)) = 5\). These last two updates are shown in Figures 6.9 (c) and (d) respectively. State \(A_3\) is an example of a state which has its degree imposed by the weights of the transitive productions. In fact, the sum of the weights of the in-links to state \(A_3\) is 5, which is above the degree of the corresponding state on both input grammars.

![Figure 6.9: Example of the computation of the union of two HWGs.](image)

**Algorithm 6.4**: The pseudo-code for the method to compute the union of two HWGs.

```
 GrammarUnion \((G_1, G_2)\)
 1. begin
  2. \(G = G_1..copy()\);
  3. for every \((A_i, A_j) \in G_2\)
  4. \(G.IncrementLinkWeight(A_i, A_j, \max(0, |A_i \cdot A_j| - |A_i|))\);
  5. end for
  6. for every \(A_i \in G\)
  7. if \(|A_i| > \max(|A_i|, |A_i|)\) then
  8. \(G.DecrementLinkWeight(SA_i, |A_i| - \max(|A_i|, |A_i|))\);
  9. else if \(|A_i| < \max(|A_i|, |A_i|)\) then
 10. \(G.IncrementLinkWeight(SA_i, \max(|A_i|, |A_i|) - |A_i|)\);
 11. end if
 12. end for
 13. end.
```

The output grammar obtained by the union operation is similar to the given by the
sum operation. However, while in the sum the weight of a production gives the total number of times the corresponding link was traversed by the users, in the union the weight of a link gives the maximum number of times the link was traversed by the users. The definition of this operation gives the analyst enhanced flexibility when analysing the overall behaviour of a group of users, where each class of users is characterised by its own HWG.

**The difference between two grammars**

The output of the *difference* between two grammars, \( G_1 - G_2 \), gives the subset of the web which was more heavily traversed by the user represented by \( G_1 \) than by the user represented by \( G_2 \). The weight of a transitive production in the output grammar is given by the difference of the weights of the corresponding productions in \( G_1 \) and \( G_2 \). The definition of the difference between two grammars now follows.

**Definition 6.8 (Grammar difference).** The difference between two HWGs, \( G = G_1 - G_2 \), is the grammar with states

\[
A_i \in G \text{ if and only if } (A_i)_1 \in G_1 \text{ and productions } \\
A_iA_j \in G \text{ if and only if } (A_iA_j)_1 \in G_1.
\]

The weight of a transitive production is \( |A_iA_j| = |A_iA_j|_1 - |A_iA_j|_2 \).

The degree of a state is \( |A_i| = |A_i|_1 \) if \( A_i > 0 \) or \( A_i > 0 \), or \( |A_i| = 0 \) if \( A_i = A_i = 0 \), that is, if a state in \( G \) is included in at least one transitive production its degree is equal to the degree of the corresponding state in the input grammar \( G_1 \), otherwise the state has degree zero.

Algorithm 6.5 gives the pseudo-code for the method to compute the difference between two grammars. With this algorithm only the states which are to be eliminated need to have their degree corrected. In fact, the primitive method to decrement the weight of a link maintains the degree of the states unchanged, see Algorithm 6.2, and the degree of a state in the output grammar is equal to the degree of the corresponding state in \( G_1 \), which is the grammar assigned to \( G \) in the first step of the method.

Figure 6.10 shows how the difference between the grammars in the running example given in Figure 6.7 is obtained. In order to compute \( G = G_1 - G_2 \) the algorithm
starts by assigning a copy of $G_1$ to $G$. Then, the link traversals in $G_2$ that correspond to links which exist in $G_1$ are removed from the grammar $G$. Figure 6.10 (a) shows $G_1$ after traversal $|A_1A_2|_2$ is removed and Figure 6.10 (b) shows the grammar after traversal $|A_2A_3|_2$ is removed. Figures 6.10 (c) and (d) illustrate the computation of $G = G_2 - G_1$.

Figure 6.10: Two examples of the computation of the difference between two HWGs.

**Difference** $(G_1, G_2)$

1. begin
2. $G = G_1.copy();$
3. for every $A_iA_j \in G$
4. $G.DecrementLinkWeight(A_iA_j, \min(|A_iA_j||, |A_iA_j|_2));$
5. end for
6. for every $A_i \in G$
7. if $A_i = A_i = 0$ then
8. $|SA_i| = 0;$
9. $|A_iF| = 0;$
10. end if
11. end for
12. end.

Algorithm 6.5: The pseudo-code for the method to compute the difference between two HWGs.

One can think of the output grammar $G = G_1 - G_2$ given by Definition 6.8 as corresponding to the grammar $G_1$ minus the link traversals of grammar $G_2$. An alternative
interpretation would be to define the difference to be $G_1$ minus both the link traversals and page visits in $G_2$. However, with such interpretation there is no guarantee of obtaining a valid HWG as a result of the operation. Figure 6.11 shows an example where this stricter definition of difference results in an invalid grammar. In fact, in the grammar $G(c)$, which corresponds to $G(a) - G(b)$, both states $A_3$ and $A_4$ are balanced but are not reachable from $S$ or able to reach $F$. In such cases the grammar is not reduced. A non-reduced grammar can have cycles with probability one and, therefore, generate infinite strings even for high values of the cut-point. Note that if we normalise a HWG having $\alpha > 0$ all states are reachable from $S$ in the corresponding HPG, and a cycle such as the $A_3 A_4 A_3$ in Figure 6.11 (c) can be reached. Grammar $G(d)$ corresponds to the output obtained when the Definition 6.8 is used to compute the difference $G(a) - G(b)$. The difference is the degree attributed to states $A_2$ and $A_3$ where each state is kept with the degree it has in grammar $G(a)$.

Figure 6.11: An example of the drawbacks of adopting a strict definition for the computation of the difference between two HWGs.

Another alternative interpretation for the difference between two grammars, $G = G_1 - G_2$, is to consider that a link is in the output grammar, $G$, only if the corresponding link is in $G_1$ but not in $G_2$. This definition is also stricter than the one proposed
in Definition 6.8. If this alternative definition is adopted, a link with weight one in \( G_2 \) is not included in the output grammar even if the corresponding link in \( G_1 \) has a very high weight. Definition 6.8 is more flexible, and its output represents a grammar whose higher probability strings identify the trails which were significantly more traversed by the user represented by grammar \( G_1 \) than by the user represented by \( G_2 \).

The operation to compute the difference between HWGs is useful to compare the contents of two grammars. For example, two individual users are given the means to compare their navigation history, characterised by HWGs, and identify which parts of the web are unknown to one of them. Such an exercise can be invaluable in helping the user to find subsets of the web with relevant contents.

**The intersection of two grammars**

The reachability relation of the intersection between two grammars, \( G = G_1 \cap G_2 \), contains only those links which exist in both \( G_1 \) and \( G_2 \). The weight of one such link represents the minimum number of times the corresponding hypertext link was traversed by a user. The definition of the intersection of two grammars now follows.

**Definition 6.9 (Grammar intersection).** The output of the intersection of two input HWGs, \( G = G_1 \cap G_2 \), is defined as the grammar containing only the states and transitive productions that belong to both input grammars

\[
A_i \in G \text{ if and only if } (A_i)_1 \in G_1 \text{ and } (A_i)_2 \in G_2, \text{ and }
\]

\[
A_i A_j \in G \text{ if and only if } (A_i A_j)_1 \in G_1 \text{ and } (A_i A_j)_2 \in G_2.
\]

The weight of a transitive production is defined as the minimum number of times the corresponding link was traversed by any of the users

\[
|A_i A_j| = \min(|A_i A_j|_1, |A_i A_j|_2).
\]

The degree of a state is equal to the maximum degree between the degrees of the corresponding states in the original grammars

\[
|A_i| = \max(|A_i|_1, |A_i|_2).
\]

In Definition 6.9 we are proposing a non-strict interpretation for the intersection of two grammars in what concerns to the computation of the degree of a state. In fact, the degree of a state in the output grammar is set to be equal to the maximum degree between the corresponding states in the input grammars.
Figure 6.12 shows an example wherein the adoption of a strict definition for the intersection of $G(a)$ and $G(b)$ results in $G(c)$, which is an invalid HWG. This is an example where the strict definition for the intersection is uninteresting since the two grammars have an overlapping in the sessions modelled, the only difference being the start and final state. The strict definition would give an empty grammar in this case.

In addition, the example shows that it is not always obvious which value should have the degree of a state. One possible solution in this example is to adopt either $G(a)$ or $G(b)$ as the solution, but that would make the operation noncommutative. A useful solution is the one given in Figure 6.12 (d) which corresponds to the output grammar obtained by Definition 6.9. In this case the degree of a state is equal to the maximum degree between the corresponding states in the two input grammars. Note that, the reachability relation of the output grammar is well defined and we only adopt a non-strict interpretation for the intersection when computing the degree of the states.

![Figure 6.12: An example of the consequences of adopting a strict definition for the computation of the intersection between two HWGs.](image)

Algorithm 6.6 gives the pseudo-code for the method to compute the intersection of two grammars and Figure 6.13 illustrates the computation of the intersection of the two input grammars given in the running example of Figure 6.7. Figure 6.13 (a) gives grammar $G$ after being assigned a copy of $G_1$ and having the weight of link $A_1A_2$ decremented by one. This way the link is given the required weight which is $\min(|A_1A_2|_1, |A_1A_2|_2) = 1$. Figure 6.13 (b) shows grammar $G$ after the weight of link $A_2A_3$ was decremented by a factor of two. Thereafter, (c) gives the grammar after link $A_3A_1$ is removed. At this stage the transitive productions have the desired weight but
we still need to check the degree of every state. In Figure 6.13 (d) the degree of state $A_1$ is incremented by one in order to have the weight $\min(|A_1|_1, |A_1|_2) = 3$ and the result of the intersection of the two grammars is obtained.

The intersection operation is useful to identify common characteristics in different grammars. For example, two grammars whose output intersection is similar to the contents of the input grammars can be said to be similar. Therefore, the intersection can serve as the basis for a clustering technique aimed at identifying clusters of similar grammars in a large collection of grammars. In addition, the intersection operation can be used by the analyst of web server log files to identify the subset of the web site which is among the interests of every class of users, where each class is characterised by a HWG. Such knowledge is useful for the improvement of the web site design.

**The complement of a grammar**

Finally, we give the definition of *complement* in the context of HWGs. In set theory the complement of a given set is the difference of that set to the universal set. The universal set contains all the elements relevant to the problem in hands.

If a HWG represents the user interaction within a web site the universal set, $U$, can be considered to be a grammar representing the complete web site. The weights of such grammar can be set according to the expected user behaviour when visiting the site. In this case the complement of the grammar representing the user behaviour would help
Algorithm 6.6: The pseudo-code for the method to compute the intersection of two HWGs.

to identify deviations between expected and the actual user behaviour. In cases where a HWG represents a user personal history when browsing the web, the complement could be used, for example, to compare the user knowledge of the web with the subset of the web indexed by a search engine.

**Definition 6.10 (Grammar complement).** The complement of a HWG, \( G = \neg G_1 \), is given by the difference between a grammar representing the known subset of the web, \( U \), and the grammar \( G_1 \)

\[
G = U - G_1.
\]

As final remarks note that all these operations are equally applicable when using the \( N \) gram model. In addition, since all the grammar transformations are performed with the primitive methods defined in Section 6.4 all operations are closed, in the sense that applying the operations to two HWGs results in a HWG.

### 6.6 Summary

In this chapter we have defined a set of operations on hypertext probabilistic grammars (HPG). The operations enhance the HPG model analysis capabilities by allowing the

**GrammarIntersection** \((G_1, G_2)\)

1. begin
2. \( G = G_1.copy() \);
3. for every \((A_i, A_j \in G \text{ AND } |A_i, A_j| > 0)\)
4. \( G.\text{DecrementLinkWeight}(A_i, A_j, \max(0, |A_i, A_j| - |A_i, A_j|_2)) \);
5. end for
6. for every \( A_i \in G \)
7. if \(|A_i| > \max(|A_i|_1, |A_i|_2)\) then
8. \( G.\text{DecrementLinkWeight}(SA_i, |A_i| - \max(|A_i|_1, |A_i|_2)) \);
9. else if \(|A_i| < \max(|A_i|_1, |A_i|_2)\) then
10. \( G.\text{IncrementLinkWeight}(SA_i, \max(|A_i|_1, |A_i|_2) - |A_i|) \);
11. end if
12. end for
13. end.
analyst to compare the contents of two different grammars. The operations are defined on hypertext weighted grammars (HWG) and we showed that for every HPG there is always a HWG whose normalisation corresponds to the given HPG. The operations defined are: union, sum, intersection, difference, and complement. Moreover, we showed that the sum of two grammars, each inferred from a different log file, is equal to a grammar incrementally inferred from the two log files.
Chapter 7

Summary and Conclusions

7.1 Summary of the Thesis

In this thesis we have proposed a novel data mining model to capture user web navigation patterns. We model the user navigation sessions as a hypertext probabilistic grammar (HPG) whose higher probability generated strings correspond to the user preferred trails on the web. We supply an algorithm to exhaustively compute all the trails with probability above a specified cut-point, where the set of such trails is called a rule-set. The algorithm is shown to scale linearly with the number of states in the model. In addition, we propose two heuristics that give the analyst enhanced control over the length and probability of the trails included in the rule-set. The first heuristic performs an iterative deepening search and incrementally builds the rule-set by first exploring trails whose probability is higher. A stopping parameter is provided which gives the analyst control over the number and quality of the rules mined. The second heuristic makes use of a dynamic cut-point which imposes a very strict evaluation criterion for short trails and becomes more permissible as the trails get longer. With this heuristic the analyst can build a relatively small set of long rules. We prove that this heuristic returns a rule-set whose trails have average link probability above the initial cut-point. In addition, we define a set of binary operations for HPGs that enables the analyst to compare the structure of different HPGs. Such operations are useful for assessing the evolution of the user navigation patterns and for comparing the knowledge different users or groups of users have about the web.
7.1. Summary of the Thesis

The HPG model has the potential of being successful in the fast moving field of web technologies. In fact, the HPG definition is based on the well established theory of probabilistic regular grammars and Markov chains which gives it a solid foundation. A HPG is built from a collection of user navigation sessions, which can be inferred from information collected by the web server or by the user browser (via a local proxy). Therefore, the model can be relatively independent of the available browsing technologies, file transfer protocols, or of the languages available to format pages to be displayed. The proposed model is simple and intuitive, provides a compact way of storing log data, is incremental, extensible, and is equipped with efficient algorithms to analyse its contents.

A model such as the HPG is better suited to handle the navigation information of relatively stable web sites. For example, pages with dynamic contents are not accurately modelled by a single state representing all its different configurations. In addition, in order to have log data representing real user behaviour some caution is needed in the pre-processing stage. In fact, user agents that request in advance a collection of the pages from a web site will make a lot of misleading page requests; the same can be said about crawlers used by search engines.

The use of a Markov model is sometimes criticised by arguing that it does not represent exactly the actual user navigation sessions. Although such assumption is not realistic in some situations, see Section 2.3.3.3, we view it as an advantage of the model since, in general, the probability of a long trail being followed often in exactly the same manner is low. Also, the contents of a page recently viewed should have more influence in the choice of the next link to follow than a page viewed in the early steps of the session.

One limitation of this work is the restricted analysis performed regarding the semantics of the rules induced. However, in order for such analysis to be meaningful the analyst has to be aware of the contents and business objectives of the web site or of the profile of the user who provided the log data. We are planning to build a tool to collect personal navigation data which will enable us to conduct meaningful experiments. A tool that records individual user web navigation sessions can be implemented as a
browser plug-in or as a local proxy and will give the means to build personal HPGs. The user can then be confronted with the induced rules and asked to assess the usefulness of the model as a personal navigation assistant tool. Also, in order to evaluate the semantic quality of the rules induced by a HPG used as a web server log analysis tool, the co-operation of the web designer is important since he is the person able to assess which rules are interesting.

7.2 Directions for Future Research

There are several directions to be explored in future research, which are now listed.

- Conduct a set of experiments aimed at evaluating the semantics of the mined rules - We are planning to build a tool to be connected to the web browser in order to collect the user's navigation history. With such tool we will be able to perform controlled experiments by both selecting a group of users with focused interests and controlling the quality of the data collected. Knowing the user goals when navigating the web and working in co-operation with him will allow us to assess the usefulness of the rules obtained.

- Extend the model in a way that enables the algorithms to take into account the pages' contents when mining the rules - Each web page in a HPG can have its contents analysed in order to quantify its relevance to a set of keywords. Being able to determine the relevance of a page to a query will make it possible to identify the high probability trails that are also relevant to that query.

- Conduct a set of experiments to evaluate the grammar operations defined in Chapter 6 - Such experiments can be performed by collecting the personal navigation history of a group of users with a common interest. After inferring a HPG for each individual user the usefulness in practice of the grammar operations can be assessed. In particular we are interested in assessing the applicability of the operations as a method to find clusters of grammars with similar content.

- Conduct a set of controlled experiments to evaluate the HPG's usefulness to the end user - One experiment could consist in assigning a task to a group of users.
The task could be filling a shopping basket with a collection of products available in the site. Moreover, the users' performance in the completion of the task could be evaluated by, for example, the time spent, the number of pages viewed and the number of unrelated pages downloaded. After concluding the task, the feedback given by the users in conjunction with the analysis of the their navigation information with the HPG model would be used as guidelines to improve the web site design. Then, a different group of users would be asked to perform the same or a similar task and the comparison of the performance of the two groups would be a measure of the model’s usefulness.

- Further study the use of information theoretic measures to describe the statistical properties of the model - Intuition suggests that there is a relation between a grammar’s entropy and the characteristics of the rules mined from it. Such a relation can be useful to gain some insight in the determination of the cut-point. In Section 4.4 we studied the correlation between the entropy and the number of mined rules by making use of the concept of posterior grammar. However, more studies are needed in order to fully understand the use of the entropy as an estimator of the rules characteristics. One particular aspect to consider is how to take into account the cut-point when computing the entropy of a HPG since the definition of a grammar’s entropy does not take the cut-point into account.

In addition, the concept of relative entropy, which is a measure of the distance between two probability distributions, has the potential to be useful as a measure of the similarity between HPGs.

- Further study the trade-off between the model complexity and its accuracy when using the Ngram concept - A better understanding is needed on the conditions for the applicability of the $\chi^2$ test, see Section 3.3, and to determine how much data is needed in order to have high confidence in the results.

- Study the compression rate of log data that the HPG model can achieve and compare the results with other methods used to store user navigation sessions.
7.2. Directions for Future Research

In the following section we elaborate on the problem of incorporating relevance measures into the model.

7.2.1 Incorporating Relevance Measures in HPG

In a HPG a page is characterised by a probability proportional to the number of times it was visited, and a link is characterised by a probability proportional to the number of times its was traversed. In this section we discuss how the HPG model can be enhanced in order to take into account the page's contents. If the model takes into account the contents of web pages it is possible to identify the high probability trails which are composed by pages relevant to a given topic.

Assume that each web page is described by a vector in which the value of each item corresponds to a keyword and denotes the utility of viewing the page given that keyword, (Baeza-Yates and Ribeiro-Neto, 1999). Such vector is called the vector of relevances. One additional item in that vector can represent the average time the users spent viewing the page, normalised in order to take into account the page size. A HPG which has its pages characterised by relevance vectors will be called an enhanced hypertext probabilistic grammar (or simply eHPG).

Having characterised an eHPG it is possible to find the user's high probability navigation trails which are also relevant to a given query. Such trails can be mined with an algorithm such as the breadth-first search (BFS), see Section 3.5, which has the cut-point definition extended in order to take into account the relevance measure. To that effect, we define the enhanced cut-point which is composed of three different thresholds: the probability threshold, the time threshold, and the term-relevance threshold. The probability threshold corresponds to the cut-point of a HPG, the time threshold is a lower bound for the viewing time of the pages in a rule, and the term-relevance threshold indicates the value above which the analyst wants a page to be considered relevant to a query term.

Both the time and relevance thresholds can have a strict or a non-strict interpretation. The strict interpretation imposes that every page composing a rule has to pass the threshold test and the non-strict interpretation imposes that the average relevance of the pages composing a rule has to pass the threshold. Therefore, the non-strict ver-
7.3 Applications of the HPG model

The language generated by an eHPG with an enhance cut-point is composed of all the trails that meet the criteria of user navigation preference, term relevance and viewing time. The model can be further enhanced in a similar way in order to include other criteria for the selection of pages.

7.3 Applications of the HPG model

There are several applications for the HPG model which we now discuss in some detail.

- **Provide guidelines to the optimisation of a web site structure:** Knowing the users preferred trails within a web site helps the web site designer to understand the users' behaviour. Such knowledge is essential in the process of tuning the web site structure to meet its business objectives. Such objectives can be, for example, to personalise web pages, to increase the average time a user spends in the web site, or to introduce new pages in places which make them highly visible. If the designer aims at increasing the average time a user spends on the site he can create links between pages of the trails that are most frequently accessed. In addition, if he wants to introduce a new product he can include links to its relevant pages.
from a popular trail that includes related products. Knowing the popular trails can also help to identify the most common points where users terminate their sessions so that the content on those points could be improved. The rules given by a HPG are also useful in planning online advertising.

- **Adaptive web sites:** HPGs can work as models of the users' preferences in the implementation of an adaptive web site. When a user visits the site a HPG is chosen to be his model and the pages shown to him are adapted accordingly. In a simple implementation the links are ordered according to the probabilities of being chosen. A more advanced implementation adapts a page in a way that provides entry points for high probability and relevant trails, whose score is given next to the links corresponding to entry points.

- **Search engines:** Enhance search engines technology. A HPG can be useful to enhance the random surfer concept used by the Google search engine (see Section 2.3.2). In fact, the knowledge of the probabilities with which the links in a page are chosen can be used to attribute degrees of importance to the different links. The subsets of the web with no available user navigation information could continue to be assumed as having a uniform distribution.

  Another aspect of the search engine technology that can be improved is the way the query results are presented to a user. An improved way of doing that could consist in presenting to the user an ordered set of high probability trails relevant to the submitted query, instead of simply presenting an ordered set of pages.

- **Web personal assistant:** A HPG can be seen as a personal tool that stores the user's web navigation history. In such context the HPG works as a memory aid and as an enhanced bookmarking facility. When using a HPG the user is able to identify his preferred trails for a given topic and share that information with his peers. The operations on grammars enable the user to compare his knowledge of the web with other users, and by doing so identify important trails that are unknown to him.
7.4. Final Remarks

With the constant growth of the web, users are getting frustrated in their search for useful information. As such, there is room for new tools and techniques to help both the user and the web site designer to enhance the web navigation experience. In this thesis we have proposed a formal model that captures user web navigation patterns. The model can provide the foundation for the development of new tools aimed at improving the navigation experience. In fact, the model can be used to improve search engine technology, to help build web visualisation tools, as user models in the implementation of adaptive web sites, or as the core of a personal web navigation assistant. The HPG model is a solid model, capable of coping with the fast moving field of web technologies since it is based on a sound theoretic foundation.
Appendix A

Details of the HPG Implementation

In this Appendix we give details of the implementation of the hypertext probabilistic grammar (HPG) model. All code has been programmed in Java (Morrison, 1997) on a Unix Sun Solaris platform and Jdbc (Reese, 1997) was used to connect the application to an Oracle database which was used to store the data. Altogether, more than 8000 lines of code were written (including comments). The code is organised into the classes given in Figure A.1.

<table>
<thead>
<tr>
<th>Class</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HPG</td>
<td>Main class that implements the HPG data structures and methods.</td>
</tr>
<tr>
<td>HPGproduction</td>
<td>Data structure containing the elements of a production rule.</td>
</tr>
<tr>
<td>HPGstate</td>
<td>Data structure containing the elements of a grammar state.</td>
</tr>
<tr>
<td>RuleSet</td>
<td>Class that implements the data structure and methods to manage a set of rules induced from a HPG.</td>
</tr>
<tr>
<td>Rule</td>
<td>Data structure containing the elements of a rule.</td>
</tr>
<tr>
<td>JDBCfunctions</td>
<td>Class containing a collection of methods that implement the interaction with the Oracle database.</td>
</tr>
<tr>
<td>LogFile</td>
<td>Class containing a collection of methods to process log files.</td>
</tr>
</tbody>
</table>

Figure A.1: A list of the classes in the HPG model implementation.

We will now describe in some detail the more relevant classes. A HPG is implemented using a adjacency linked list representation. The set of grammar states is represented by an array of elements of the type HPGstate. Moreover, for each state a linked list composed of instances of the class HPGproduction keeps a list of all the productions having that state on their left-hand side. A list of the more important methods in the HPG class is given in Figure A.2.
<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFS()</td>
<td>Implements the breadth-first search data mining algorithm, see Section 3.4.</td>
</tr>
<tr>
<td>DFS()</td>
<td>Implements the depth-first search data mining algorithm, see Section 3.4.</td>
</tr>
<tr>
<td>FineGrained()</td>
<td>Implements the Fine-Grained heuristic, see Section 5.3.</td>
</tr>
<tr>
<td>InverseFisheye()</td>
<td>Implements the Inverse Fisheye heuristic, see Section 5.4.</td>
</tr>
<tr>
<td>IterativeDeepening()</td>
<td>Implements the Iterative Deepening heuristic, see Section 5.2.</td>
</tr>
<tr>
<td>CreateGrammar()</td>
<td>Infers a grammar from a log file, see Section 3.4.</td>
</tr>
<tr>
<td>UpdateGrammar()</td>
<td>Updates an existing grammar with information from a new log file, see Section 3.4.</td>
</tr>
<tr>
<td>RandomGrammar()</td>
<td>Creates a random grammar, see Section 4.1.</td>
</tr>
<tr>
<td>PosteriorGrammar()</td>
<td>Infers a grammar from a set of rules, see Section 4.4.</td>
</tr>
<tr>
<td>LoadGrammar()</td>
<td>Loads a grammar that is stored in a database table.</td>
</tr>
<tr>
<td>NormaliseGrammar()</td>
<td>Computes the HPG corresponding to a given hypertext weighted grammar, see Definition 3.6.</td>
</tr>
<tr>
<td>SouleEntropy()</td>
<td>Computes the entropy of a HPG while seeing it as an absorbing Markov chain, see Section 3.6.</td>
</tr>
<tr>
<td>EntropyRate()</td>
<td>Computes the entropy of a HPG while seeing it as an irreducible Markov chain, see Section 3.6.</td>
</tr>
<tr>
<td>CopyGrammar()</td>
<td>Creates a new copy of the current grammar.</td>
</tr>
<tr>
<td>InverseGrammar()</td>
<td>Computes the inverse of the transition matrix corresponding to the current grammar.</td>
</tr>
<tr>
<td>InsertProduction()</td>
<td>Inserts a new production or updates the weights of an existing production in the grammar.</td>
</tr>
<tr>
<td>InsertState()</td>
<td>Inserts a new state or updates the weights of an existing state in the grammar.</td>
</tr>
<tr>
<td>Print()</td>
<td>Prints the contents of a grammar on the screen.</td>
</tr>
<tr>
<td>QuiSquareTest()</td>
<td>Computes the $\chi^2$ test for the grammar.</td>
</tr>
<tr>
<td>SaveGrammar()</td>
<td>Saves the grammar in a database table.</td>
</tr>
<tr>
<td>SaveRules()</td>
<td>Saves a set of induced rules in a database table.</td>
</tr>
</tbody>
</table>

Figure A.2: The more important methods in the HPG class.
Similarly, the RuleSet class is implemented using an linked list representation. The RuleSet possesses an array of pointers to elements of the type Rule where each pointer is the head of a linked list containing the set of induced rules which start with the same state. The size of the array is equal to the number of grammar states. A list of the methods in the RuleSet class is given in Figure A.3.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Push()</td>
<td>Inserts a new rule.</td>
</tr>
<tr>
<td>Peek()</td>
<td>Returns a copy of the first rule in a specified row.</td>
</tr>
<tr>
<td>Pop()</td>
<td>Removes the first rule from a specified row.</td>
</tr>
<tr>
<td>Empty()</td>
<td>Returns true if the rule-set is empty and false otherwise.</td>
</tr>
<tr>
<td>Print()</td>
<td>Prints the contents of the rule-set on the screen.</td>
</tr>
</tbody>
</table>

**Figure A.3: The methods in the RuleSet class.**

A rule is stored as a text string in which the codes of the states composing the rule are separated by commas. A list of the methods in the Rule class is given in Figure A.4.

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Length()</td>
<td>Returns the length of a rule.</td>
</tr>
<tr>
<td>FirstState()</td>
<td>Returns the first state of a rule.</td>
</tr>
<tr>
<td>LastState()</td>
<td>Returns the last state of a rule.</td>
</tr>
<tr>
<td>AppendState()</td>
<td>Appends a new state into the tail of a rule.</td>
</tr>
<tr>
<td>Copy()</td>
<td>Returns a copy of a rule.</td>
</tr>
</tbody>
</table>

**Figure A.4: The methods in the Rule class.**

In Figure A.5 we give the Entity-Relationship diagram (Levene and Loizou, 1999b) for the database which stores the data of the experiments. In the diagram it is shown what are the main attributes of each class. Note that all the parameters used in the generation of a random HPG are stored, these parameters are:

- the number of states (NumStates),
- the average branching factor (BFparam),
- the seed of the random number generator used to generate the grammar skeleton (SeedSkeleton),
- the parameter of the Poisson distribution used in the generation of the links' weights (PoissonParam), and
- the seed used in the random generation of the links weights (SeedWeights).

By storing the seeds of the random number generators we are able to reproduce the experiments. Moreover, the use of a different seed for the skeleton and for the links' weights enables to generate several different HPGs for a given skeleton. We have used the random number generator available in the Java standard libraries.
Figure A.5: The Entity-Relationship diagram for the database used in the experiments.
Bibliography


liopoulos, M., editors, Web Usage Mining. To appear in Lecture Notes in Artificial
Intelligence (LNAI 1836). Springer Verlag, Berlin.

Brusilovsky, P., Kobsa, A., and Vassileva, J., editors, Adaptive Hypertext and Hy­

enabled knowledge discovery process. In Proceedings of ninth International
Database Conference, pages 13–27, Hong Kong.


Press, San Diego, California.

Cadez, I., Heckerman, D., Meek, C., Smyth, P., and White, S. (2000). Visualization of
navigation patterns on a web site using model based clustering. In Proceedings of
the sixth ACM SIGKDD International Conference on Knowledge Discovery and
Data Mining, Boston, Massachusetts. To appear.


rations, 1(2):1–11.

Chakrabarti, S., Dom, B. E., Gibson, D., Kleinberg, J., Kumar, R., Raghavan, P., Ra­
jagopalan, S., and Tomkins, A. S. (1999a). Mining the link structure of the world

approach to topic-specific web resource discovery. In Proceedings of the eighth


In *Proceedings of the tenth ACM Conference on Hypertext and Hypermedia: Returning to Our Diverse Roots*, pages 133–134, Darmstadt, Germany.


Index

Symbols

\( \alpha \), 70
\( \chi^2 \), 81
\( \delta \), 73
\( \varepsilon \), 51
\( \eta_k \), 120, 125
\( \lambda \), 72
\( \lambda^p \), 136
\( \lambda_\sigma \), 135
\( \mathcal{R} \), 24, 65
\( S \), 24
\( |w| \), 51
\( \tau \), 121, 125
\( \theta \), 73
\( w \), 51

A

absorbing state, 57
alphabet, 51
ambiguous grammar, 52
ambiguous language, 53
automaton, 55

B

balanced state, 70
best-trail criterion, 125
branching factor, 73
branching factor cut-point, 136

C

candidate rule, 88
candidate sub-trail, 124
candidate trail, 88
chi-square test, 81
closed language, 53
conditional entropy, 62
confidence threshold, 73
consistent grammar, 54
context-free grammar, 51
context-free language, 51
cookies, 40, 49
cut-point, 60, 72

d.f., 83
degrees of freedom, 83
derivation, 51
length, 52
probability, 54
derivational entropy, 92
DFA, 55
dynamic cut-point, 135
Index

E
empty string, 51
entropy, 61
entropy rate, 93
ergodic set, 57
ergodic state, 57
exploration tree, 119, 123

F
FG, 123
final production, 52, 67
final state, 66
final string, 51
fine-grained, 123
finite automata, 55

G
geometric cut-point, 135
grammar
ambiguos, 52
complement, 172
context-free, 51
difference, 166
entropy, 92
equality, 154
intersection, 169
reduced, 53
regular, 51
skeleton, 54
sub-grammar, 154
sum, 161
unambiguous, 52
union, 164
greedy criterion, 125

H
HFA, 59
HG, 66
history depth, 80
HPA, 60
HPG, 69, 71
HTML, 29
HWG, 69, 149
hypertext database, 24, 64
hypertext finite automata, 59
hypertext grammar, 66
unambiguous, 68
hypertext language, 68
hypertext probabilistic automata, 60
hypertext probabilistic grammar, 71
reduced, 72
hypertext probabilistic language, 74
hypertext system, 19, 64
hypertext trail, 65
hypertext weighted grammar, 69, 149
reduced, 70

I
ID, 119
IFE, 135
inadmissible expansion, 88
induced grammar, 75
inverse fisheye, 135
iteration, 89
<table>
<thead>
<tr>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>iterative deepening, 119</td>
</tr>
<tr>
<td>J</td>
</tr>
<tr>
<td>joint entropy, 61</td>
</tr>
<tr>
<td>L</td>
</tr>
<tr>
<td>language, 51</td>
</tr>
<tr>
<td>ambiguous, 53</td>
</tr>
<tr>
<td>regular, 52</td>
</tr>
<tr>
<td>unambiguous, 53</td>
</tr>
<tr>
<td>M</td>
</tr>
<tr>
<td>Markov chain, 57</td>
</tr>
<tr>
<td>absorbing, 57</td>
</tr>
<tr>
<td>aperiodic, 57</td>
</tr>
<tr>
<td>irreducible, 57</td>
</tr>
<tr>
<td>periodic, 57</td>
</tr>
<tr>
<td>maximal rule-set, 119</td>
</tr>
<tr>
<td>maximal string, 74</td>
</tr>
<tr>
<td>maximal trail, 74, 88, 120</td>
</tr>
<tr>
<td>Memex, 19</td>
</tr>
<tr>
<td>N</td>
</tr>
<tr>
<td>navigation problem, 23</td>
</tr>
<tr>
<td>navigation session, 68</td>
</tr>
<tr>
<td>NFA, 55</td>
</tr>
<tr>
<td>(N)gram, 79</td>
</tr>
<tr>
<td>nondeterministic finite automata, 55</td>
</tr>
<tr>
<td>nonterminal, 66</td>
</tr>
<tr>
<td>normalisation, 70</td>
</tr>
<tr>
<td>O</td>
</tr>
<tr>
<td>one-step derivation, 51</td>
</tr>
<tr>
<td>P</td>
</tr>
<tr>
<td>parent grammar, 115</td>
</tr>
<tr>
<td>Poisson distribution, 97</td>
</tr>
<tr>
<td>posterior grammar, 115</td>
</tr>
<tr>
<td>primitive methods, 155</td>
</tr>
<tr>
<td>probabilistic grammar, 53</td>
</tr>
<tr>
<td>probabilistic language, 53</td>
</tr>
<tr>
<td>probabilistic regular grammar, 54</td>
</tr>
<tr>
<td>production, 51, 66</td>
</tr>
<tr>
<td>production numbering, 68</td>
</tr>
<tr>
<td>proper sub-grammar, 154</td>
</tr>
<tr>
<td>R</td>
</tr>
<tr>
<td>random surfer, 36</td>
</tr>
<tr>
<td>reachability relation, 24, 65, 159</td>
</tr>
<tr>
<td>reduced grammar, 53, 70</td>
</tr>
<tr>
<td>regular grammar, 51</td>
</tr>
<tr>
<td>regular language, 52</td>
</tr>
<tr>
<td>rule, 74, 88</td>
</tr>
<tr>
<td>rule-set, 74, 121</td>
</tr>
<tr>
<td>S</td>
</tr>
<tr>
<td>sentential entropy, 92</td>
</tr>
<tr>
<td>sentential form, 51</td>
</tr>
<tr>
<td>session, 68</td>
</tr>
<tr>
<td>skeleton, see grammar skeleton</td>
</tr>
<tr>
<td>Soule entropy, 93</td>
</tr>
<tr>
<td>start production, 67</td>
</tr>
<tr>
<td>start state, 66</td>
</tr>
<tr>
<td>start symbol, 51</td>
</tr>
<tr>
<td>state</td>
</tr>
<tr>
<td>balanced, 70</td>
</tr>
</tbody>
</table>
Index

state degree, 70, 149
state entropy, 93
stopping criterion, 121, 125
stopping parameter, 120, 125
string, 51
    derivation, 51
    length, 51
    probability, 54
sub-grammar, 154
sub-rule, 74
sub-trail, 120
support threshold, 73

T
terminal, 66
trail, 65
    admissible expansion, 119
    candidate, 119, 124
    candidate sub-trail, 124
    inadmissible expansion, 119
    maximal, 119, 120, 124
    sub-trail, 120
transient set, 57
transient state, 57
transition diagram, 56
transitive production, 52, 67

U
unambiguous grammar, 52
unambiguous language, 53
unbalanced HWG, 152
user navigation session, 68

X
Xanadu, 20
XML, 29

Z
Zipf distribution, 97