An Average Linear Time Algorithm for Web Usage Mining

José Borges

School of Engineering, University of Porto

R. Dr. Roberto Frias, 4200 - Porto, Portugal

jlborges@fe.up.pt

Mark Levene

School of Computer Science and Information Systems

Birkbeck, University of London

Malet Street, London WC1E 7HX, U.K.

mark@dcs.bbk.ac.uk

October 22, 2003

Abstract

In this paper we study the complexity of a data mining algorithm for extracting patterns from user web navigation data that was proposed in previous work [3]. The user web navigation sessions are inferred from log data and modeled as a Markov chain. The chain's higher probability

trails correspond to the preferred trails on the web site. The algorithm implements a depth-first search that scans the Markov chain for the high probability trails. We show that the average behaviour of the algorithm is linear time in the number of web pages accessed.

Keywords. Web usage mining, Markov chains, analysis of algorithms

1 Introduction

Web usage mining is defined as the application of data mining techniques to discover user web navigation patterns in web log data [15]. Log files provide a list of the page requests made to a given web server in which a request is characterised by, at least, the IP address of the machine placing the request, the date and time of the request, and the URL of the page requested. From this information it is possible to reconstruct the user navigation sessions within the web site [1], where a session consists of a sequence of web pages viewed by a user in a given time window. The web site owner can take advantage of web usage mining techniques to gain insights about the user behaviour when visiting the site and use the acquired knowledge to improve the design of the site.

Two distinct directions are, in general, considered in web usage mining research. In the first, the user sessions are mapped onto relational tables and an adapted version of standard data mining techniques, such as mining association rules, is invoked, see for example [11]. In the second approach, in which we situate our research, techniques are developed which can be invoked directly on the log data, see for example [3] or [14].

In [14] the authors propose a novel data mining model specific to analyze log data. A log data mining system is devised to find patterns with predefined characteristics by means of a query language to be used by an expert. Several authors have proposed the use of Markov models to model user requests on the web. Pitkow et al. [12] proposed a longest subsequence model as an alternative to the Markov model. Sarukkai [13] uses Markov models for predicting the next page accessed by the user. Cadez et al. [6] use Markov models for classifying browsing sessions into different categories. Deshpande et al.[7] propose techniques for combining different order Markov models to obtain low state complexity and improved accuracy. Finally, Dongshan and Junyi [8] proposed an hybrid-order tree-like Markov model to predict web page access which provides good scalability and high coverage. Markov models have been shown to be suited to model a collection of navigation records, where higher order models present increased accuracy but with a much larger number of states.

In previous work we have proposed to model a collection of user web navigation sessions as a *Hypertext Probabilistic Grammar* (HPG). The HPG is a self-contained and compact model which is based on the well established theory of probabilistic grammars providing it with a sound foundation for future enhancements such as the study of its statistical properties. In [3] we proposed an algorithm to extract the higher probability trails which correspond to the users preferred navigational trails.

In this paper we provide a formal analysis of the algorithm's complexity, which is a first step in the direction of making the various web usage mining

models proposed in the literature comparable, since the diverse characteristics of the patterns induced by the various models makes such comparison difficult.

The HPG model can alternatively be seen as an absorbing Markov chain [9]. Since the HPG concept is not essential for the algorithm's complexity analysis, herein, for simplicity, we will treat our model in terms of an absorbing Markov chain. For details on the HPG model we refer the reader to [2, 3, 4, 5].

In Section 2 we briefly present the proposed model and the depth-first algorithm for finding patterns in the data. In Section 3 we present the main result of this paper proving that the algorithm has average linear time complexity.

2 Markov Chains to Model User Navigation Sessions

2.1 Building the Model from the Navigation Sessions

In this section we will introduce the proposed Markov model by means of an example. The model is inferred from the collection of sessions. Consider a web site with three pages, $\{A, B, C\}$, and the following collection of user navigation sessions:

$$(1)\ A, B, C, B \quad (2)\ A, C, B, C, C \quad (3)\ B, C, B \quad (4)\ C, A, B.$$

Each navigation session gives a sequence of pages viewed by a user. To each web page visited we create a corresponding state in the model.

Moreover, there are two additional states: a start state S representing the first state of every navigation session and a final state F representing the last

state of every navigation session.

Figure 1 (a) shows the Markov chain modelling just the first session. There is a transition corresponding to each sequence of two pages in the session, a transition from the start state to the first state of the session, and a transition from the last state of the session to the final state. The probability of a transition is estimated by the ratio of the number of times the corresponding sequence of pages was traversed and the number of times the anchor page was visited.

The model is incrementally build by processing the complete set of navigation sessions. Figure 1 (b) shows the model for the entire collection of sessions given in the example.

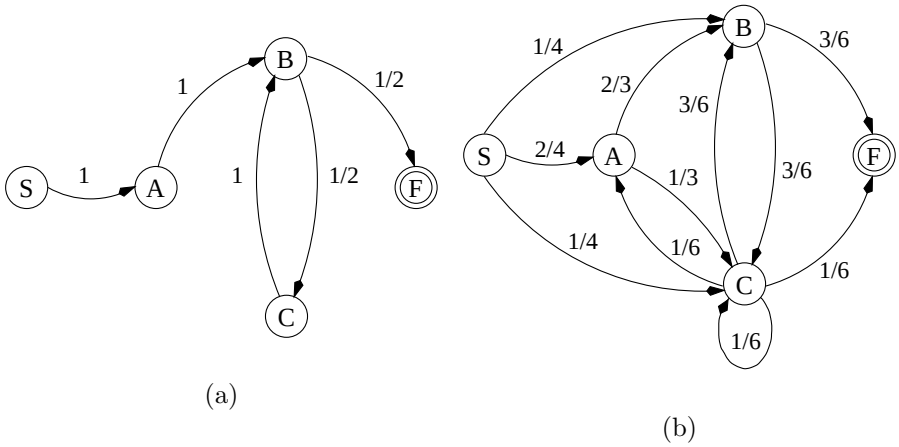

Figure 1: The process of building the Markov model.

Figure 2 shows the Markov chain corresponding to the example represented in Figure 1 (b). The Markov chain is defined by a set of states X, a transition matrix T, and a vector of initial probabilities π. The set of states, X, is composed by the start state, S, the final state, F, and the states that correspond

to the web pages visited. The transition matrix records the transition probabilities which are estimated by the proportion of times the corresponding link was traversed from the anchor.

The initial probability of a state is estimated as the proportion of times the corresponding page was requested by the user. Therefore, according to the model's definition, apart from the final state, all states have a positive initial probability. In [3] we make use of a parameter by which we can tune the model to be between a scenario where the initial probabilities are proportional to the number of times a page has been requested as the first page and a scenario where the probabilities are proportional to the number of times the page has been requested. For this paper we have adopted the latter scenario which enables us to identify sequences of pages which were frequently followed but were not necessarily at the beginning of a user navigation session.

From the method given to infer the model we note that every state in X (i.e., a state present in at least one navigation session) is included in at least one path from state S to state F. Since the final state F does not have out-transitions and it is reachable from every other state, the state F is an absorbing state and, therefore, the Markov chain is an absorbing chain.

As described, the model assumes that the probability of a hypertext link being chosen depends solely on the contents of the page being viewed. Several authors have shown that models that make use of such an assumption are able to provide good accuracy when predicting the next link the user will choose to follow, see for example [13]. In addition, this assumption can be relaxed by

making use of the Ngram concept [3], or the dynamic Markov chain concept [10]. Since the application of the two referred concepts results in a model with the same properties as the model described herein we refer the reader to [3] and [10] for more detail.

$X =$ { A, B, C, F }

$\pi =$ $< 3/15,\quad 6/15,\quad 6/15,\quad 0 >$

$T =$

	A	B	C	F
A	0	2/3	1/3	0
B	0	0	1/2	1/2
C	1/6	3/6	1/6	1/6
F	0	0	0	1

Figure 2: The Markov chain corresponding to the example.

The Markov chain inferred from the log data summarises the user interaction with the web site and the aim is to identify patterns in the navigation behaviour.

Definition 1 (Trail) *We define a* trail *as a finite sequence of states that are accessed in the order of their traversal in the underlying web site.*

According to the model proposed, the probability of a trail is estimated by the product of the initial probability of the first state in the trail and the transition probabilities of the enclosed transitions. For example, the estimated probability of trail A, B, C, F is $3/15 \cdot 2/3 \cdot 1/2 \cdot 1/6 = 6/540$. Note that, a trail induced by the model does not necessarily have to end in the state F. The probability estimated for trail A, B, C, which is $3/15 \cdot 2/3 \cdot 1/2 = 6/90$, gives the probability of A, B, C as a prefix of other trails, and the probability estimated for A, B, C, F gives the probability of a user terminating the session

after following the trail A, B, C.

Definition 2 (Navigation patterns) *A set of* navigation patterns *is defined to be the set of trails whose estimated probability is above a specified cut-point,* $\lambda \in (0, 1)$.

We define the cut-point, λ, to be composed of two distinct thresholds, with $\lambda = \theta\,\delta$, where $\theta \in (0, 1)$ is the *support* threshold and $\delta \in (0, 1)$ the *confidence* threshold. This decomposition is adopted in order to facilitate the specification of the cut-point value. In fact, since the model assumes that every state has a positive initial probability, the values of the probabilities in the vector of initial probabilities are of a much smaller order of magnitude than the values of the transition probabilities. Therefore, when setting the cut-point value we recommend the analyst to view the support threshold as the factor responsible for pruning out the states whose initial probability is low, corresponding to a subset of the web site rarely visited. Similarly, we recommend to view the confidence as the factor responsible for pruning out trails containing transitions with low probability.

One difficulty that arises with models such as the one described herein is how to set the value of the parameters. The idea behind decomposing the cut-point into two components is to provide the analyst with some insight on how to set the parameter's value. For example, if the support threshold is set in a way that takes into account the number of states in the model, i.e. a model with n states having the support set to $\theta = 1/n$, it would mean that only pages which were visited a number of times above the average will be considered as being

initial states of a trail.

Similarly, in order to set the value of the confidence threshold the analyst could take into account the web site branching factor, that is, the average number of out-links per page. For example, if the model has on average t out-links per page the average transition probability is $1/t$. Therefore, if the analyst aims to identify trails composed by transitions whose estimated probability is greater than $1/t$ and with length m, the confidence threshold should be set to $\delta = (1/t)^{(m-1)}$. The factor $(m-1)$ is necessary because the first state in a trail is obtained by a transition from the start state which is taken into account by the support threshold. Two other algorithms that make use of other cut-point definitions in order to identify different types of patterns were proposed in [4] and [5]. Assuming a support threshold of $1/n$, in the context of a set of controlled experiments, it is possible to vary the overall value of the cut-point by varying its confidence component.

2.2 The Algorithm

The algorithm proposed for finding the set of all trails with probability above a specified cut-point consists of a generalisation of a depth-first search, [16]. An exploration tree is built with the start state as its root wherein each branch of the tree is explored until its probability falls below the cut-point.

Definition 3 (Rule) *A branch with probability above the cut-point and with no extensions leading to a longer branch with probability above the cut-point is called a* rule.

While a branch in the tree corresponds to a trail users may follow in the web site, a rule corresponds to a trail that has, according to past behaviour, high probability of being followed by the users. Note that we only include maximal trails in the induced rule-set, RS, where a trail is maximal if it is not a proper prefix of any other trail in the set. All non-maximal trails that have probability above the cut-point are implicitly given by one, or more than one, maximal trail. For example, if a trail X, Y, Z is maximal (i.e., cannot be augmented) the non-maximal trail X, Y also has its probability above the cut-point, however it will not be included in the rule-set because it is implicitly given by the corresponding maximal trail.

We now give the pseudo-code for our depth-first search algorithm. We let X be the set of states, $|X|$ be its cardinality, and $X_i, 1 \leq i \leq |X|$, represent a state. Moreover, wX_i represents a trail being evaluated. For a trail composed by m states, w represents the prefix-trail composed by the first $m-1$ states and X_i represents the m^{th} state which is the tip of the trail being evaluated. We let $p(wX_i)$ represent a trail's probability. The concatenation operation, $wX_i + X_j$, appends the state X_j to the trail wX_i, resulting in the trail wX_iX_j with X_j being its tip. Also, $T_{i,j}$ represents the probability of a transition from state X_i to state X_j, π_i the initial probability of state X_i, and RS represents the set of rules being induced. The transition matrix is implemented as a linked list in such way that each state has its out-transitions represented by a list of the states which according to the user's navigation records are reachable from it in one step. Links that were not traversed and have an estimated probability

of 0 are not kept in the list. The notation, $X_i \rightarrow lst$ represents the access to the first state from the list of states that have a transition from state i, ptr represents a pointer to the state from the list that is currently being handled, and $ptr = ptr \rightarrow next$ assigns to ptr the next state in the list. Finally, X_{ptr} represents the state currently indicated by a pointer.

Algorithm 1.1 (DFSmining(λ))
1. **begin**
2. **for** $i = 1$ **to** $|X|$
3. **if** $\pi_i > \lambda$ **then** Explore(X_i, π_i);
4. **end for**
5. **end.**

Algorithm 1.2 (Explore(wX_i, $p(wX_i)$))
1. **begin**
2. $flag = false$;
3. $ptr = X_i \rightarrow lst$;
4. **while** (ptr != $null$)
5. **if** $p(wX_i) \cdot T_{i,\,ptr} > \lambda$ **then**
6. Explore($wX_i + X_{ptr}$, $p(wX_i) \cdot T_{i,\,ptr}$);
7. $flag = true$;
8. **end if**
9. $ptr = ptr \rightarrow next$;
10. **end while**
11. **if** $flag = false$ **then** $RS = RS \cup \{wX_i\}$;
12. **end.**

Figure 3 illustrates the exploration tree induced when the algorithm with the cut-point set to $\lambda = 0.11$ is applied to the example in Figure 2. A plain line indicates the composition of the maximal trails and a dashed line corresponds to a transition whose inclusion in the trail being explored would lead to a trail with probability below the cut-point.

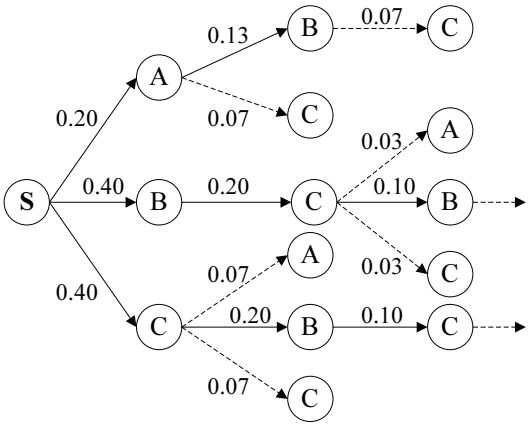

Figure 3: The exploration tree induced by the algorithm when applied to the example on Figure 2 with $\lambda = 0.11$.

2.3 Experimental Evaluation

Experiments were conducted on randomly generated Markov chains in order to assess the performance of the proposed algorithm. For a complete description of the method used to generate the random data and full presentation of experiment results we refer the reader to [2] and [3]. Herein we confine the presentation to the results that lead the authors to the pursuit of a formal complexity analysis of the algorithm's performance.

Figure 4 shows the variation of the number of operations performed with the number of states in the model for different values of the cut-point. The results shown correspond to models with an average number of five out-links per page; we call the average number of out-links per page the model's *branching factor* (or simply BF). We fixed the support threshold to be $\theta = 1/n$ and, therefore,

in the figure the cut-point is indicated by the value of its confidence threshold component. We define an *operation* as the evaluation of a link when constructing the exploration tree. For each configuration of the model 30 runs were performed and the results shown correspond to the average number of operations for 30 runs. We note that similar behaviour was observed on real data sets [3]. These results suggest that the algorithm has average linear time behaviour.

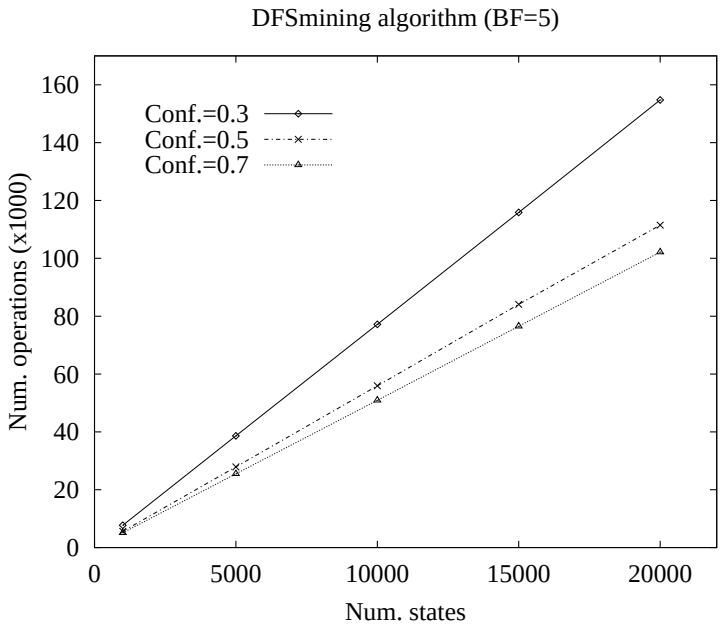

Figure 4: Variation of the number of operations with the model's number of states.

3 Analysis of the Algorithm's Average Complexity

We now give our analysis of the average case complexity for the proposed algorithm. We measure the algorithm's complexity by the number of operations, where an operation is defined as a link evaluation in the exploration tree.

Consider a Markov chain with $n = |X|$ states, t transitions between states having probability greater than 0, and cut-point λ. The model's branching factor, BF, is the average number of out-transitions per state and is given by t/n. Also, we define the length of a trail as the number of states it contains.

Given a model, the average number of trails starting in a particular state and having a given length, Δ, corresponds to the number of distinct branches in a BF-ary tree whose depth is equal to Δ. Thus, since the navigation can begin at any of the n states, we can estimate the average number of trails having a given length, Δ, denoted by $E(\#w_\Delta)$, by the following expression:

$$E(\#w_\Delta) = n \cdot BF^{(\Delta-1)} = n \left(\frac{t}{n}\right)^{(\Delta-1)}.$$

Given that every state has positive initial probability the average initial probability of a state is $1/n$. Similarly, the average transition probability between two states is $1/BF = n/t$. Therefore, the average probability of a trail with length Δ, $E(p(w_\Delta))$, will be

$$E(p(w_\Delta)) = \frac{1}{n} \left(\frac{n}{t}\right)^{(\Delta-1)}.$$

By making $E(p(w_\Delta)) = \lambda$ we can estimate the average trail length for the given cut-point value, Δ_λ, as

$$\Delta_\lambda = \frac{ln(\lambda n)}{ln\left(\frac{n}{t}\right)} + 1.$$

Note that the algorithm augments trails until their probability fall below the cut-point, therefore, on average the probability of the maximal trails inferred is close to λ.

Given that $\lambda = \theta\,\delta$ and assuming $\theta = 1/n$ we have

$$\Delta_\lambda = \frac{ln(\delta)}{-ln\,(BF)} + 1. \tag{1}$$

This last expression shows that for a given confidence threshold and branching factor, the estimated average rule length is constant and independent of the number of states. There is a special case to consider when $BF = 1$, which occurs if all existing trails from S to F are disjoint and with no recurrent states. In this case, $E(p(w_\Delta)) = 1/n$ and Δ_λ is given by the average length of all the trails from S to F. In general, since every state has at least one out-transition $BF \geq 1$, therefore, if $\theta = 1/n$ it follows that $\Delta_\lambda \geq 1$. In addition, for a given δ the average number of trails with length Δ_λ is

$$E(\#w_{\Delta_\lambda}) = n\left(\frac{t}{n}\right)^{(\Delta_\lambda - 1)} = n\left(\frac{t}{n}\right)^{\left(\frac{ln(\delta)}{-ln(t/n)}\right)} = \frac{n}{\delta}, \tag{2}$$

since

$$(\tfrac{t}{n})^{\frac{ln(\delta)}{-ln(t/n)}} = e^{-ln(\delta)} = \tfrac{1}{\delta}.$$

Expression (2) gives the average number of rules for a given cut-point. Intuitively, (2) follows from the fact that $1/\delta$ gives an upper bound on the number of trails we can pack into δ for each of the n states.

We will now determine the average number of operations necessary to induce the rule-set RS for a given cut-point. As before we define an operation as the evaluation of a link. The *for* loop in Algorithm 1.1 performs one operation per state, in a total of n operations. For each state Algorithm 1.2 is invoked.

In Algorithm 1.2 there is a *while* loop that recursively calls the algorithm from line (6). In the worst-case analysis, each recursive call of the algorithm evaluates all the out-transitions from the tip of the trail, and therefore, for each invocation of the while loop the algorithm performs on average BF operations. Moreover, in order to induce a rule-set for a Markov chain with n states, each of the n states needs to have its out-transitions evaluated, therefore, the average rule length, Δ_λ, corresponds to the depth of recursion. Finally, the average number of operations performed, denoted by $E(\#O_n)$, is bounded below and above by

$$\frac{1}{\delta} \leq \frac{E(\#O_n)}{n} \leq \sum_{i=0}^{\lceil \Delta_\lambda \rceil} BF^i = \frac{1 - BF^{\lceil \Delta_\lambda \rceil + 1}}{1 - BF} \tag{3}$$

were $\lceil \Delta_\lambda \rceil$ is the ceiling of Δ_λ.

We are now ready to present the analysis of the average complexity of the algorithm.

Theorem *Given a fixed support, $\theta = O(1/n)$, and confidence, δ, the average number of operations needed to find all the trails having probability above the cut-point, $\lambda = \theta\,\delta$, varies linearly with the number of states in the absorbing Markov Chain.*

Proof. Consider a Markov chain with n states, t transitions between states, $\lambda = \theta\,\delta$ and assume $\theta = 1/n$. From their definitions δ and BF are independent of n. Also, it follows from (1) that Δ_λ is independent of n. Therefore, it follows from (3) that the number of operations depends linearly of n. □

Following the analysis, we can state that the worst-case complexity of the algorithm occurs when the average trail length is maximal. Assuming that T_{max} is the maximal probability of a transition in the Markov chain, for $T_{max} < 1$ we can derive the maximum for the average trail length as

$$\Delta_{T_{max}} = \frac{ln(\lambda n)}{ln(T_{max})} + 1.$$

To obtain the number of operations corresponding to the worst-case just replace Δ_λ by $\Delta_{T_{max}}$ in Equation 3.

We now illustrate the result by means of an example. Consider a model having $n = 5$, $BF = 2$ and $\lambda = \theta\delta = 0.05$ where δ is taken to be 0.25 and θ to be 1/5. The average case of such model consists of a Markov chain in which every state has exactly two out-transitions and every transition has probability 0.5. Thus, the estimate for the average trail length is given by

$$\lceil \Delta_\lambda \rceil = \frac{ln(\delta)}{-ln(BF)} + 1 = 3$$

To induce the rule-set the algorithm constructs the exploration tree represented in Figure 5. In the figure, plain lines indicate links that are part of trails with probability above the cut-point (i.e. rules), dotted lines indicate links whose inclusion in the trail being evaluated would lead to a trail with probability below the cut-point and the lines represented by a dots and dashes represent an exploration tree similar to the one detailed. Finally, the numbers next to the

links indicate the order in which links are evaluated.

Figure 5 indicates the number of operations (link evaluations) performed by the algorithm in order to induce the set of maximal trails. The first link to be evaluated is the transition from the start state to state n_1, which has probability 1/5. Then, transitions are recursively evaluated in a depth-first scheme until the trail's probability falls below the threshold. Since each state has two out-transitions with equal probability the induced trails are composed by just two transitions. Therefore, in order to induce the maximal trails that start from state n_1 we need to perform 15 operations, and 75 operations are needed to induce the complete set of maximal trails.

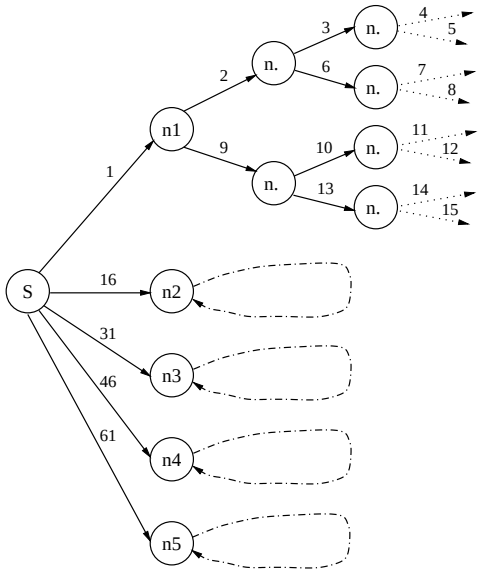

Figure 5: Example of the exploration tree resulting from the algorithm.

The average case analysis for the example gives

$$E(\#O_n) \leq n \ \frac{1-BF^{\lceil \Delta_\lambda \rceil + 1}}{1-BF} = 75$$

which corresponds to five times the sub-tree detailed in Figure 5. Finally, the worst-case analysis depends of the T_{max} value as shown in Table 1.

Table 1: The variation of the worst-case analysis with the T_{max} value.

T_{max}	0.6	0.7	0.8	0.9	1
$\Delta_{T_{max}}$	3.7	4.9	7.2	14.2	∞
$E(\#O_n)$	155	315	2555	327675	∞

4 Concluding Remarks

Several authors have been studying the problem of mining web usage patterns from log data. Patterns inferred from past user navigation behaviour in a site are useful to provide insight on how to improve the web site design's structure and to enable a user to prefetch web pages that he is likely to download. In previous work we have proposed to model users' navigation records, inferred from log data, as a hypertext probabilistic grammar and an algorithm to find the higher probability trails which correspond to the users' preferred web navigation trails. In this paper we present a formal analysis of the algorithm's complexity and show that the algorithm presents on average linear behaviour in the number of web pages accessed. In the literature there are several other web usage mining algorithms, however, comparison is not always possible due to the diversity of the assumptions made in the various models. Providing the average complexity analysis of our algorithm is a step in the direction of making the web different usage mining approaches comparable.

As future work we mention the study of the form of the probability distribution which characterises user navigation behaviour and the effect of such a distribution on the complexity analysis. Also, we aim to explore dynamic Markov chains and state cloning in the context of web usage mining [10]. Finally, we plan to conduct a study to evaluate the usefulness of the web usage patterns to the user and to incorporate relevance measures into the model.

Acknowledgements. The authors would like to thank the referees for several comments and suggestions to improve the paper.

References

[1] Bettina Berent, Bamshad Mobasher, Myra Spiliopoulou, and Jim Wiltshire. Measuring the accuracy of sessionizers for web usage analysis. In *Proceedings of the Web Mining Workshop at the First SIAM International Conference on Data Mining*, pages 7–14, Chicago, April 2001.

[2] José Borges. *A Data Mining Model to Capture User Web Navigation*. PhD thesis, University College London, London Uiversity, 2000.

[3] José Borges and Mark Levene. Data mining of user navigation patterns. In Brij Masand and Myra Spliliopoulou, editors, *Web Usage Analysis and User Profiling*, Lecture Notes in Artificial Intelligence (LNAI 1836), pages 92–111. Springer Verlag, Berlin, 2000.

[4] José Borges and Mark Levene. A fine grained heuristic to capture web navigation patterns. *SIGKDD Explorations*, 2(1):40–50, 2000.

[5] José Borges and Mark Levene. A heuristic to capture longer user web navigation patterns. In *Proceedings of the first International Conference on Electronic Commerce and Web Technologies*, pages 155–164, Greenwich, U.K., September 2000.

[6] I. Cadez, D. Heckerman, C. Meek, P. Smyth, and S. White. Visualization of navigation patterns on a web site using model based clustering. In *Proceedings of the 6th KDD conference*, pages 280–284, 2000.

[7] Mukund Deshpande and George Karypis. Selective markov models for predicting web-page accesses. In *Proc. of the 1st SIAM Data Mining Conference*, April 2001.

[8] Xing Dongshan and Shen Junyi. A new markov model for web access prediction. *Computing in Science & Engineering*, 4(6):34–39, 2002.

[9] John G. Kemeny and J. Laurie Snell. *Finite Markov Chains*. D. Van Nostrand, Princeton, New Jersey, 1960.

[10] Mark Levene and George Loizou. Computing the entropy of user navigation in the web. *International Journal of Information Technology and Decision Making*, 2:459–476, 2003.

[11] Bamshad Mobasher, Honghua Dai, Tao Luo, and Miki Nakagawa. Using sequential and non-sequential patterns for predictive web usage mining

tasks. In *Proceedings of the IEEE International Conference on Data Mining*, pages 669–672, Japan, December 2002.

[12] James Pitkow and Peter Pirolli. Mining longest repeating subsequences to predict world wide web surfing. In *Proc. of the Second Usenix Symposium on Internet Technologies and Systems*, Colorado, USA, October 1999.

[13] R. Sarukkai. Link prediction and path analysis using markov chains. In *Proceedings of the 9th Int. WWW conference*, 2000.

[14] M. Spiliopoulou and C. Pohle. Data mining for measuring and improving the success of web sites. *Data Mining and Knowledge Discovery*, 5:85–114, 2001.

[15] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: Discovery and applications of usage patterns from web data. *SIGKDD Explorations*, 1(2):1–12, January 2000.

[16] Robert Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1(2):146–160, June 1972.