DAWN - A System for Context-based Link Recommendation in Web Navigation

Sebastian Stober and Andreas Nürnberger

Institute for Knowledge and Language Engineering,
School of Computer Science,
Otto-von-Guericke-University Magdeburg, D-39106 Magdeburg, Germany
{stober,nuernb}@iws.cs.uni-magdeburg.de

Abstract. We have developed the system "DAWN" (direction anticipation in web navigation) that learns navigational patterns to help users navigating through the world wide web. In this paper, we present the prediction model and the algorithm for link recommendation of this system. Besides this main focus, we briefly outline the system architecture and further motivate the purpose of such a system and the approach taken. A first evaluation on real-world data gave promising results.

1 Introduction

Users that browse the world wide web often need to choose between multiple options on how to continue navigation. The dynamic nature of the steadily growing and constantly changing web makes this a very hard task. Depending on how well hyperlinks are chosen, it will take users less or more time to finally get to the information they desire. Whilst some users might find this task quite easy, others may get lost, especially if they are facing unfamiliar web content. Although there is most likely no general rule on how to most efficiently navigate to the desired information, there may be certain navigational patterns that can be used as heuristics. Users may unconsciously develop such patterns. A system that watches users navigating the web could learn their navigational patterns and use them to provide navigational support by suggesting the best matching hyperlinks.

This paper introduces a prototype of such a system, named DAWN (direction anticipation in web navigation). Differences to related work are pointed out in Sect. **??**. Sect. **??** gives an overview on the system architecture and addresses model induction and hyperlink recommendation as the two crucial points of the system in detail. In Sect. **??** we present some promising results of a first evaluation of the system with real-world data. Finally, we summarize our work and give an outlook on future developments.

2 Related Work

The general idea to support users browsing the world wide web is not new. Systems that accomplish this task by suggesting hyperlinks or web pages are e.g. dis-

cussed in [?,?,?,?,?]. The system presented here has much in common with these systems: Like Webmate [?], Broadway [?] or Personal Webwatcher [?] it uses an HTTP-proxy to log user actions and to manipulate the requested web pages. Documents are represented according to common practice as term vectors with TF/iDF-weights. Letizia [?] was one of the first systems that used background look-ahead crawling. However, Letizia was designed as a plug-in for the Netscape Navigator 3 and heavily relied on its API whereas the proxy-architecture chosen for DAWN allows the users to use their browsers of choice. Moreover, bandwidth and computational expensive tasks can be performed on the server which reduces the burden on the client machine. In contrast to DAWN, which uses a combination of navigational patterns and document similarities, Webmate, Personal Webwatcher and Letizia are purely content-based systems that rely on document-similarities to decide which web pages are interesting. Broadway relies on case-based reasoning to recommended web pages. Webwatcher [?] uses a collaboration-based approach by asking a user about the desired information and then suggesting links that users with similar information need have followed previously. Furthermore, Webwatcher is a server-side application that is restricted to one website, whereas DAWN works on a client-side proxy and therefore has no such local restriction.

DAWN stores the navigational patterns in a Markov Model. Such models have been successfully used for prediction of HTTP-requests to optimize web-caches [?]. Recently, they have also been applied to modeling of user navigational behavior in the context of adaptive websites [?,?,?] and web-usage mining [?]. However, these are solely server-side applications that are inherently locally confined, i.e. they are restricted to one website. To our knowledge, there have not been any client-side systems that use Markov Models so far. The algorithm for model induction has been derived from the one presented by Borges and Levene in [?] which has been designed to represent a collection of user navigation sessions of a website. Sect. **??** briefly summarizes the algorithm's concept and subsequently discusses our modifications and extensions.

3 System Architecture

An overview of the system architecture is shown in Fig. **??**. The system has been integrated into CARSA, an architecture for the development of context adaptive retrieval systems [?]. All HTTP-request made by the user through the system's HTTP-proxy are recorded and stored in a database (DB). The requested web pages are analyzed and modified prior to forwarding them to the user. This allows tracking of browsing sessions including events such as clicks, form submits and closing of browser windows or the browser itself. From the information stored in the database a model-learner generates a model using the algorithm described in Sect. **??**. It is possible to learn a separate model for each user as well as a global one. Based on a model, candidate pages collected by a web crawler that performs a background look-ahead crawl can be assessed given a user's current history of the most recently accessed web pages. The recommenda-

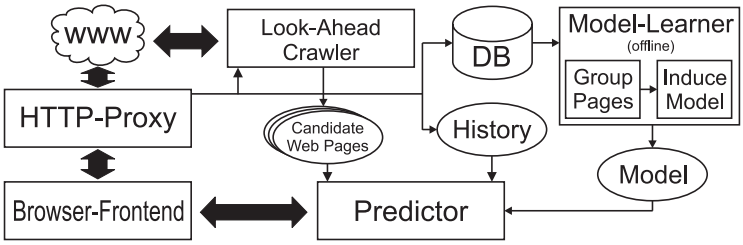


Fig. 1. DAWN: System Overview.

tion algorithm is described in Sect. **??**. Eventually, information about the most probable candidates is displayed in the browser-frontend. The display combines ranking, content and visual information to make it easier for the user to assess the candidates' relevance. Ranking information is provided by sorting the candidates according to the predicted visit probability. Displaying title, URL and keywords gives information about the content and automatically generated thumbnails give a general visual impression of the page layout.

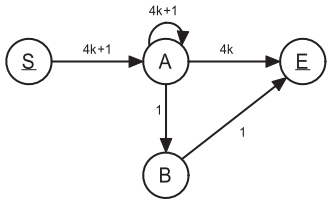
3.1 Model Induction

The algorithm for model induction has been derived from the one proposed by Borges and Levene in [**?**] which has been designed to represent a collection of user navigation sessions of a website. This algorithm iteratively constructs an n^{th}-order Markov Model by making use of the state cloning concept where a state is only duplicated if the n^{th}-order probabilities diverge significantly from the corresponding 1^{st}-order probabilities. Once a state needs cloning, the n^{th}-order probability distributions induced by the different in-paths of the state are clustered. In-paths referring to probability distributions assigned to the same cluster can then be redirected to the same clone of the original state. This reduces the number of clones needed and reduces the model size.

The algorithm by Borges and Levene is intended for server-side application. To be able to use it in a client-side setting as addressed by DAWN, the algorithm had to be extended. Most importantly, we introduced an additional clustering step prior to model induction as an extra abstraction level. In this additional step similar pages are grouped together thus drastically reducing the size of the model's state space and combining different browsing paths into an abstract navigational pattern. As a side effect, cycles of length one may occur when consecutive pages in a browsing path are assigned to the same cluster. These cycles are something the original algorithm cannot deal with. As a result, an n^{th}-order model (with $n \geq 3$) may contain paths of length n that do not exist in the data and thus induce undefined transition probabilities as shown in Fig. **??**. State A is not cloned because for large k all transition probability distributions (table rows) dependent on k converge to $(\frac{1}{2}, 0, \frac{1}{2})$.

We propose the following extension of the algorithm to solve this problem: Prior to induction of an n^{th}-order model from an $(n-1)^{\text{th}}$-order model, the

Browsing Path	Frequency
A	$2k$
A, A	k
A, A, A, A	k
A, A, B	1



(1^{st}-order)	A	B	E
S	1	0	0
A	$\frac{4k+1}{8k+2}$	$\frac{1}{8k+2}$	$\frac{4k}{8k+2}$
B	0	0	1

(2^{nd}-order)	A	B	E
SA	$\frac{2k+1}{4k+1}$	0	$\frac{2k}{4k+1}$
AA	$\frac{2k}{4k+1}$	$\frac{1}{4k+1}$	$\frac{2k}{4k+1}$
AB	0	0	1

(3^{rd}-order)	A	B	E
SAA	$\frac{k}{2k+1}$	$\frac{1}{2k+1}$	$\frac{k}{2k+1}$
AAA	$\frac{k}{2k}$	0	$\frac{k}{2k}$
AAB	0	0	1

Fig. 2. An incorrect 3^{rd}-order model induced by the algorithm presented in [**?**]. The path S, A, B exists in the model but not in the data. (upper left: data, lower left: graph representation of the induced model, right: 1^{st}, 2^{nd} and 3^{rd}-order transition probabilities)

$(n-1)^{\text{th}}$-order model is checked for paths of length n that do not exist in the data. Let $\boldsymbol{l} = (l_{i_1}, \ldots, l_{i_n})$ be such a path and $l_{i_{n-1}}$ the state preceding the last one in the path. Further, let M be the set of all paths in the model that comprise $(n-1)$ states and have $l_{i_{n-1}}$ as the last state. All paths in M must be contained in the data because they were checked in the previous iteration of the algorithm. For the path $\boldsymbol{l'} \in M$ with $\boldsymbol{l'} = (l_{i_1}, \ldots, l_{i_{n-1}})$ the $(n-1)^{\text{th}}$-order modeled transition probability $P(l_{i_n}|\boldsymbol{l'})$ should be 0 because $\boldsymbol{l}$ does not exist in the data. Obviously, this is not the case. Otherwise $\boldsymbol{l}$ would not exist in the $(n-1)^{\text{th}}$-order model. Hence, there must be at least one path $\boldsymbol{m} \in M$ with $P(l_{i_n}|\boldsymbol{m}) > 0$. Thus, M can be partitioned into two non-empty subsets, $M^{(0)} = \{\boldsymbol{m} \in M | P(l_{i_n}|\boldsymbol{m}) = 0\}$ and $M^{(\bar{0})} = M \setminus M^{(0)}$. To fix the problem, the paths in $M^{(0)}$ need to be separated from those in $M^{(\bar{0})}$ by using the method for in-path-separation of the original algorithm. Fig. **??** shows the modified 2^{nd}-order model and the correct 3^{rd}-order model for the example in Fig. **??**.

As another modification of the original algorithm, we replaced the originally proposed K-Means-algorithm for clustering of the transition-probability-vectors by an agglomerative one which always finds the optimal clustering in a single run. Apart from that, several smaller modification were made that mainly address performance issues and are beyond the scope of this paper. They are discussed in detail in [**?**].

3.2 Hyperlink Recommendation

In the preceding section we have presented how a higher-order Markov Model can be induced. Having learned an n^{th}-order Markov Model that represents the navigational patterns from training data, candidate pages can be assessed in the

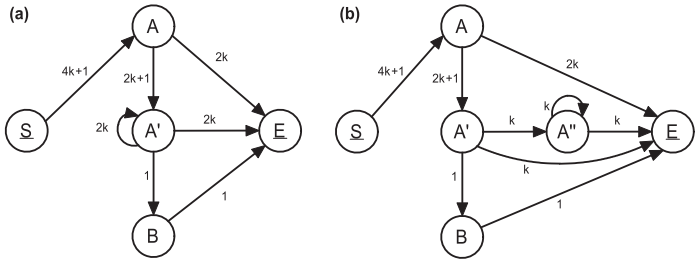


Fig. 3. Modified 2nd-order model (a) and the correct 3rd-order model (b) for the example in Fig. **??**.

context of a user's current history of the last n accessed web pages. For this, a probability distribution for the next state has to be estimated by mapping the history onto the model as shown in Alg. **??**. In the first part of the algorithm (lines 1–21), all navigational paths that are similar to the history are identified, weighted and stored in L. To compute the similarity of a web page and a state, which is a cluster of web pages from the training data that can be represented by its center, the cosine similarity measure is used. The number of states to be regarded as similar is controlled by the similarity threshold λ and bounded by the parameter k. The weight $w_{\boldsymbol{l}}$ of a path $\boldsymbol{l}$ then corresponds to the similarity of the path with the history, which is the minimum of all one-on-one state similarities. Obviously, a navigational path can only be similar to the history, if all of its sub-paths are similar to the corresponding parts of the history. Thus, the set L_d containing all similar paths with $d+1$ states can be constructed by extending all paths contained in L_{d-1}. (Considering that during a browsing session the history is only extended by one web page at a time, the computation of L can be sped up by storing the sets $L_0, \ldots, L_{n-2}$ from the previous call.)

In the second part of the algorithm (lines 22–30), the probability distribution for the next state is computed as the weighted average of the probability distributions induced by all weighted paths in L. For each path $\boldsymbol{l} \in L_d$ each set L_i with $i < d$ contains a suffix of $\boldsymbol{l}$. All these suffixes are taken into account for the computation of the probability distribution, too. At the same time the weight of a suffix cannot be smaller than the weight of the whole path because of the minimum operation that is used to compute the path weight in Alg. **??**, line 14. This leads to an intentional higher weighting of the most recently accessed web pages in the history that is motivated by the assumption that these web pages have a greater impact on the choice of the next hyperlink [**?**].

Once the probability distribution for the next state has been computed, the candidate web pages collected by the crawler can be assessed. For this, each candidate is mapped onto the model's states analogously to the mapping of the elements of the history identifying a set of similar states. The probability of the candidate is the maximum probability of all these states according to the previously estimated probability distribution for the next state. This probability can then be used to rank the candidate pages.

Algorithm 1 Estimation of the probability distribution for the next state.

function EstimateProbDistrib(n^{th}-order model with state space S, n, $\boldsymbol{h}$, λ, k)
  $L \leftarrow \emptyset$
  for depth $d = 0$ to $(\min\{n, |\boldsymbol{h}|\} - 1)$ **do** ▷ identify similar paths
    $L_d \leftarrow \emptyset$
    for all states $s \in S$ **do**
      $w_s \leftarrow sim(\boldsymbol{h}_{[d]}, s)$
    end for
    for the maximal k most similar states s with $w_s \geq \lambda$ **do**
      if $d = 0$ **then**
        $L_0 \leftarrow L_0 \cup (s, w_s)$
      else
        for all weighted paths $(\boldsymbol{l}, w_{\boldsymbol{l}}) \in L_{d-1}$ **do**
          if s predecessor of $\boldsymbol{l}$ **then**
            $\boldsymbol{l}' \leftarrow s, \boldsymbol{l}$
            $w_{\boldsymbol{l}'} \leftarrow \min\{w_s, w_{\boldsymbol{l}}\}$
            $L_d \leftarrow L_d \cup (\boldsymbol{l}', w_{\boldsymbol{l}'})$
          end if
        end for
      end if
    end for
    $L \leftarrow L \cup L_d$
  end for
  for all states $s \in S$ **do** ▷ compute probability distribution for the next state
    $p_s \leftarrow 0$
  end for
  for all weighted path $(\boldsymbol{l}, w_{\boldsymbol{l}}) \in L$ **do**
    for all successors s of $\boldsymbol{l}$ **do**
      $p_s \leftarrow p_s + w_{\boldsymbol{l}} \cdot P(s|\boldsymbol{l})$
    end for
  end for
  $\boldsymbol{p} \leftarrow \left\{ \left(s, \frac{p_s}{\sum_{s \in S} p_s} \right) \middle| s \in S \right\}$
end function

The crucial point is, that the model can be used for prediction, even if, as generally the case, a web page has not been visited during the training period. By mapping the candidate web pages and the elements of the history to multiple similar states of the model instead of using a sharp one-to-one mapping, the system can interpolate between several states in case newly encountered web pages differ significantly from those in the training data.

4 Evaluation

To get a first impression of the system's performance, we have measured the prediction accuracy on server log files. The prediction accuracy is however only an indicator for the systems usefulness. In some cases a user might have followed

a link to a resource that was not useful at all. Correct prediction of such links results in high prediction accuracy but a useful system should not make such predictions. Predictions that do not match the actual link chosen may, on the other hand, still lead to useful resources.

During the clustering step that groups similar pages for model induction and for the computation of the similarities in the prediction process (Alg. **??**, line 5) the web page content, at least in bag-of-words representation, is needed. Thus, simple log files are not sufficient. Either the content of the accessed web pages has to be included in the evaluation data or at least the accessed URLs must not be outdated, i.e. the URLs must still be valid and accessible and the content should not have changed since the access was logged. To our knowledge, there is currently no data set publicly available that meets these requirements. Therefore, we used anonymized log files that contained requests on web pages hosted by the University of Magdeburg recorded during six consecutive days for evalutaion.

The log files were filtered as follows: Any requests with a response status code other than 200 (OK) and 304 (not modified) were removed as well as requests of URLs with non-HTML content. Additionally, several possible denial of service attacks on specific URLs were detected and removed. Afterwards, sessions were identified as, e.g., described in [**?**]. The prediction model was learned from the data of the first five days. For the evaluation of the model the data of the 6^{th} day was used. Table **??** shows the number of sessions, requests and unique URLs in the data.

Table 1. Number of sessions, requests and unique URLs in the data used for training and evaluation.

dataset	sessions	requests	URLs
train	58314	237098	25771
test	13437	60461	5657
total	71751	297559	27877

The requested web pages in the training data were clustered into 250 clusters resulting in an average cluster size of about 100. From the 250 clusters and the sessions extracted from the training data, a 2^{nd}-order Markov Model was induced. This model was used to estimate the probability of all outgoing links for each web page in the test sessions. Table **??** shows the results.

Table 2. Predicted ranks of the actually followed links. The number of candidates (number of different outbound links in a web page) ranged from 1 to 607 with a mean of 10.77.

rank	1^{st}	2^{nd}	3^{rd}	1^{st}–3^{rd}	in 1^{st} third
frequency	20%	5%	5%	31%	45%

In about 31% of all test cases, the candidate that had actually been chosen by the user was amongst the three highest ranked candidates. In these situations the recommendations could have helped the user. However, as mentioned above, this does not necessarily mean that the recommendations would have been useless in the other cases. The real usefulness of the system's recommendations remains to be assessed in a user study.

5 Conclusions and Future Work

In this paper, we have presented a prototype system that provides navigation support for users browsing the world wide web. A first evaluation with real-world data has shown that the system is able to learn navigational patterns to predict a user's browsing path. As future work we plan an evaluation of the system in a user study.

References

1. Lieberman, H.: Letizia: An Agent That Assists Web Browsing. IJCAI, 1995.
2. Joachims, T., Freitag, D., Mitchell, T.: WebWatcher: A Tour Guide for the World Wide Web. In: IJCAI, 1997.
3. Chen, L., Sycara, K.: WebMate: A Personal Agent for Browsing and Searching. In: Proc. 2nd Intl. Conf. on Auton. Agents and Multi Agent Sys., AGENTS '98, 1998.
4. Jaczynski, M., Trousse, B.: Broadway, A Case-Based Browsing Advisor for the Web. In: ECDL '98: Proc. of the 2nd Europ. Conf. on Research and Adv. Technology for Digital Libraries, 1998.
5. Mladenic, D.: Machine learning used by Personal WebWatcher. In: Proc. of ACAI-99 Workshop on Machine Learning and Intelligent Agents, 1999.
6. Sarukkai, R.: Link Prediction and Path Analysis using Markov Chains. Computer Networks, Vol. 33, pp. 337–386, 2000.
7. Anderson, C., Domingos, P. Weld, D.: Relational Markov Models and their Application to adaptive Web Navigation. In: Proc. 8th ACM SIGKDD Intl. Conf. on Knowl. Discovery and Data Mining, 2004.
8. Zhu, J., Hong, J., Hughes, J.: Using Markov Chains for Link Prediction in Adaptive Web Sites. In: Soft-Ware 2002: Comp. in an Imperfect World: 1st Intl. Conf., 2002.
9. Cadez, I., Heckerman, D., Meek, C., Smyth D., White, S.: Model-Based Clustering and Visualization of Navigation Patterns on a Web Site. Data Min. Knowl. Discov., Vol. 7, pp. 399–424, 2003.
10. Borges, J., Levene, M.: Generating Dynamic Higher-Order Markov Models in Web Usage Mining. In: Proc. of the 9th Europ. Conf. on Principles and Practice of Knowl. Discovery in Databases (PKDD 05), 2005.
11. Bade, K., De Luca, E., Nürnberger, A., Stober, S.: CARSA - An Architecture for the Development of Context Adaptive Retrieval Systems. In: Adaptive Multimedia Retrieval: User, Context, and Feedback: 3rd Intl. Workshop, 2005.
12. Stober, S.: Kontextbasierte Web-Navigationsunterstützung mit Markov-Modellen. Diplomarbeit, Otto-von-Guericke-University Magdeburg, Dec. 2005.
13. Schechter, S., Krishnan, M., Smith, M.: Using path profiles to predict http requests. In: Proc. of the 7th intl. conf. on World Wide Web, 1998.
14. Cooley, R., Mobasher, B., Srivastava, J.: Data Preparation for Mining World Wide Web Browsing Patterns. Knowl. and Information Sys., Vol. 1, No. 1, pp. 5–32, 1999.