Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering 59 (2006) 270–291
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Using HMM to learn user browsing patterns for focused Web crawling

Hongyu Liu [a,*], Jeannette Janssen [b], Evangelos Milios [a]

[a] Faculty of Computer Science, Dalhousie University, 6050 University Avenue, Halifax, NS, Canada B3H 1W5
[b] Department of Mathematics and Statistics, Dalhousie University, 6050 University Avenue, Halifax, NS, Canada B3H 1W5

Received 18 January 2006; received in revised form 18 January 2006; accepted 18 January 2006
Available online 10 March 2006

Abstract

A focused crawler is designed to traverse the Web to gather documents on a specific topic. It can be used to build domain-specific Web search portals and online personalized search tools. To estimate the relevance of a newly seen URL, it must use information gleaned from previously crawled page sequences.

In this paper, we present a new approach for prediction of the links leading to relevant pages based on a Hidden Markov Model (HMM). The system consists of three stages: user data collection, user modelling via sequential pattern learning, and focused crawling. In particular, we first collect the Web pages visited during a user browsing session. These pages are clustered, and the link structure among pages from different clusters is then used to learn page sequences that are likely to lead to target pages. The learning is performed using HMM. During crawling, the priority of links to follow is based on a learned estimate of how likely the page is to lead to a target page. We compare the performance with Context-Graph crawling and Best-First crawling. Our experiments demonstrate that this approach performs better than Context-Graph crawling and Best-First crawling.
© 2006 Elsevier B.V. All rights reserved.

Keywords: Focused crawling; Web searching; Relevance modelling; User modelling; Pattern learning; Hidden Markov models; World Wide Web; Web Graph

1. Introduction

With the exponential growth of information on the World Wide Web, there is great demand for efficient and effective approaches to organize and retrieve the information available [1–4]. The imbalance between the rapid growth of the Web and the limited network and storage resources poses many challenges for generic crawlers and search engines. For example, search engines may find it hard to refresh their index frequently enough to keep up with the pace of the Web changing. As well, a topic-oriented keyword search is more and more likely to yield thousands of relevant results only based on keywords specified in the query, which

* Corresponding author.
E-mail addresses: hongyu@cs.dal.ca (H. Liu), janssen@mathstat.dal.ca (J. Janssen), eem@cs.dal.ca (E. Milios).

0169-023X/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.datak.2006.01.012

means that a user still needs to go through a selection and browsing process before finding the pages most relevant to her. On the other hand, popular topical pages cannot fit every user's interests equally well. Therefore, the generic search engines will become less appealing to a user with specific information needs that do not fall into the most popular categories. Hence there will be a market for topic-specific search tools such as Web search portals that index pages on a particular topic, or personalized search tools that search the Web for pages that fit a particular user profile.

1.1. Focused crawling

The success of topic-specific search tools depends on the ability to locate topic-specific pages on the Web while using limited storage and network resources. This can be achieved if the Web is explored by means of a *focused crawler*. A focused crawler is a crawler that is designed to traverse a subset of the Web for gathering only documents on a specific topic, instead of searching the whole Web exhaustively.

In order to find pages of a particular type or on a particular topic, focused crawlers, which were first introduced by Chakrabarti et al. [5], aim to identify links that are likely to lead to target documents, and avoid links to off-topic branches. Regular crawlers use a Breadth-First search strategy in order to download as many pages as possible, while focused crawlers selectively choose links that quickly lead to target pages. As a result, with a small investment of network resources, a focused crawler is expected to collect high quality domain-specific documents from the Web with respectable coverage at a rapid rate.

The challenge in designing a focused crawler is to predict which links lead to target pages, since the Web is noisy and multi-topic. Sometimes, relevant pages link to other relevant ones. However, it is very common that pages which are not on topic lead to topical pages indirectly. According to a study [6], most relevant pages are separated by irrelevant pages, the number of which ranges from at least 1, to a maximum of 12, commonly 5. A common approach to focused crawling is to use information gleaned from previously crawled pages to estimate the relevance of a newly seen URL. The effectiveness of the crawler depends on the accuracy of this estimation process.

1.2. Literature review

A variety of methods for focused crawling have been developed. The term focused crawler was first coined by Chakrabarti in 1999, however, the concept of prioritizing unvisited URLs on the crawl frontier for specific searching goals is not new, and Fish-Search [7] by De Bra et al. (1994) and Shark-Search [8] by Hersovici et al. (1998) were some of the earliest algorithms for crawling for pages with keywords specified in the query. In Fish-Search, the Web is crawled by a team of crawlers, which are viewed as a school of fish. If the "fish" finds a relevant page based on keywords specified in the query, it continues looking by following more links from that page. If the page is not relevant, its child links receive a low preferential value. Shark-Search is a modification of Fish-search which differs in two ways: a child inherits a discounted value of the score of its parent, and this score is combined with a value based on the anchor text that occurs around the link in the Web page.

The focused crawler introduced in [5] uses a canonical topic taxonomy. The user specifies a number of starting points, and browses from these pages. The user matches pages to the best categories while browsing. This categorization is used to train a classifier which makes relevance judgements on a document to the topic. A distiller then determines visit priorities based on hub-authority connectivity analysis.

Intelligent crawling with arbitrary predicates is described in [9]. The method involves looking for specific features in a page to rank the candidate links. These features include page content, URL names of referred Web page, and the nature of the parent and sibling pages. It is a generic framework in that it allows the user to specify the relevant criteria. Also, the system has the ability of self-learning, i.e. to collect statistical information during the crawl and adjust the weight of these features to capture the dominant individual factor at that moment. Similar methods can be found in [10–12]. These methods have in common that they use a baseline best-first focused crawling strategy combined with different heuristics based on local features extracted from the parent page, such as similarity to a query, in-degree, PageRank, and relevance feedback. This reflects the belief that in the Web graph, relevant pages are likely to link to other relevant pages.

Other methods used to capture path information leading to targets include reinforcement learning [13], genetic programming [14], and Context Graph algorithms [15]. In [13], an algorithm was presented for learning a mapping performed by Naive Bayes text classifiers from the text surrounding a hyperlink to a value function of sum of rewards, the estimate of the number of relevant pages that can be found as the result of following that hyperlink.

A genetic programming algorithm is used in [14] to explore the space of potential strategies and evolve good strategies based on the text and link structure of the referring pages. The strategies produce a rank function which is a weighted sum of several scores such as hub, authority and SVM scores of parent pages going back k generations.

The Context Graph method, proposed by Diligenti et al. [15] uses backlinks to estimate the link distance from a page to target pages. Their method starts from a set of seed documents, follows backlinks to a certain layer, and then builds up a context graph for the seed pages. A classifier is constructed for each layer using the Naïve Bayes algorithm. As a new document is found, it is classified into a layer. Documents classified into layers closer to the target are crawled first. The experiments showed that this approach maintained a higher level of relevance in the retrieved Web pages. However, the assumption that all pages in a certain layer from a target document belong to the same topic described by a set of terms does not always hold.

A complete framework to evaluate different crawling strategies is described in [16–18]. An application of extending digital libraries with a focused crawler can be found in [19].

1.3. Contribution and outline of the paper

Unlike all the systems above, our objective is to capture the hierarchical semantic linkage structure of topics. Typical examples of such topical hierarchies can be found in the Open Directory (ODP) or Yahoo directory. They are built by humans, and do not reflect the actual Web. However, in the real Web, such topical hierarchies structure do exist in a less visible way. For example, although different university Web sites may have different style and content presentations on the surface, the *Homepage* → *Department* → *People* → *Faculty* is a very common hierarchical semantic linkage structure underneath. In other words, off-topic, but semantically related, pages may often reliably lead to relevant pages. For example, a *University* page leads to a *Department* page, which leads to a *People* page, *Faculty* list page, *Research* and *Publications*. When looking for research publications on a specific topic, the crawler may have to traverse pages that are irrelevant to the topic, before it reaches highly relevant ones.

Our approach is to use a combination of semantic content analysis and link structure of paths leading to targets by learning from the user's browsing patterns and to emulate them to find more relevant pages. We believe that sequential pattern learning from user's browsing behavior on specific topics is beneficial because humans are very good at discriminating between links based on a variety of clues in a page. Semantic content analysis is performed by a clustering algorithm. Documents are grouped semantically to build a concept graph, from which the learning and prediction are done from both content and linkage by training an HMM to recognize sequences of topics that lead to relevant pages. The use of finite-state Markov models helps represent useful context including not only text content, but also linkage relations. For example, in the path $p1 \rightarrow p2 \rightarrow p3 \rightarrow w$, the prediction of the link $p3 \rightarrow w$ is based on the observable text content of w combined with features from its ancestor sequence $p1$, $p2$, and $p3$.

Our approach is unique in the following important ways. One is the way we use Hidden Markov Models for focused crawling. We think of a focused crawler as a random surfer, over an underlying Markov chain of hidden states, defined by the number of hops away from targets, from which the actual topics of the document are generated. When a new document is seen, the prediction is to estimate how many links this document is away from a target. After training, our system may have learned patterns like "University pages are more likely to lead to Research papers than Sports pages". Note that this is different from other systems in that we capture semantic associative relations along paths leading to target pages rather than physical link distances to make predictions. Another contribution is modelling the semantic structure of the Web by observing the user's behavior on a small training data set and application of this model to guide the actual Web crawler in order to find the relevant documents. Experiments show that the proposed crawler outperforms Best-First and Context-Graph crawling. The learned model is general, and its parameters are adapted to different users searching

different topics, so it is broadly applicable. It can be used to build topic-specific portals and it can help individual users perform personalized online search.

One possible application of our focused crawler is as part of a personalized search tool. Current search engines are highly effective in a keyword search for a specific, well-defined type of Web page. However, in many instances a user will have an information need that is less easily expressed with a keyword search. In this case, the user often needs to follow two or three levels of hyperlinks before finding the pages she is looking for. Our focused crawler will be able to help for such search instances. From observing the user's browsing behavior, the focused crawler will learn the nature of the relevant target pages, and recognize the pathways leading to such pages. The crawler will then go online to find more pages of this kind, and can present the URL's of such pages whenever the user next opens her browser.

Another application is the construction of topic-specific Web portals. Often such portals are maintained by professional societies and other not-for-profit organizations, who usually have limited resources at their disposal. Using a focused crawler will lead to faster updates and more complete coverage of the topic. The results of our experiments demonstrate that our browsing-based focused crawler gives results superior to focused crawlers based on other principles. To obtain the power of our approach, the cooperation of a number of experts is required, who will donate some of their time to participate in a focused browsing session for pages on the required topic in order to train the crawler. Alternatively, our crawler could learn from a general crawler's history, or from the general linked neighborhood of a number of pre-identified target pages.

A third application is to have the focused crawler interact with a regular search engine in an iterative manner. Once the focused crawler has been learned, more starting pages can be identified by querying a search engine using the most common keywords in the cluster, to find pages similar to the pages in each cluster. Paths from the new starting pages that are found leading to pages of interest are then added to the training set and the focused crawler is retrained.

The rest of the paper is organized as follows. Section 2 describes the overview of our focused crawling system and how to collect page sequences user browsed to build a concept graph. Pattern learning and the use of HMM for focused crawling are described in Section 3. Section 4 describes the detailed crawling algorithm. Experimental results are shown in Section 5. Conclusions and discussion are in Section 6.

2. System overview

Consider the way a user searches for information on the Web. Usually the surfer has an idea (topic) in mind before starting search. Then she will search for relevant pages by entering topic keywords into a search engine like Google or Yahoo. Keyword search typically results in thousands of hits. She will then select some results returned by the search engine and start browsing there. While browsing, she selectively follows links from page to page to identify what she really wants. If she reaches a page she is interested in, she may take time to read, print or save it. The process continues by choosing another link, or going back to the list of search engine results to follow other returned results, or typing new keywords to start a refined topic search. In short, the user refines keyword-based search by combining searches with further browsing to fulfill her topic-specific information need. What the user really seeks goes beyond lexical keyword search.

The Web graph is noisy and multi-topic. Off-topic pages often reliably lead to highly relevant pages. Due to the small world nature of the Web [16], links may lead to unrelated topics within an extremely short radius. At the same time, there exist long paths and large topical subgraphs where topical coherence persists. Webmasters usually follow some general rules to organize the pages of a Web site semantically. For example, university pages point to department pages, then to pages of faculty members; they rarely point to sports canoeing pages. In other words, dominant semantic topic hierarchies exist. User surfing behavior is likely to follow an intuitive understanding of such semantic topic hierarchies. The decision by the user to follow or ignore a specific link is based on a variety of clues on the page and humans are very good at making such decisions.

The above observations suggest that the context of the hyperlinks the user follows reveals the user's information need. If we can detect such sequential patterns hidden in the surfer's topic-specific browsing, we may be able to build an effective focused crawler.

Focused crawlers can only use information from previously crawled page sequences to estimate the relevance of a newly seen URL. Therefore, good performance relies on powerful modelling of context as well

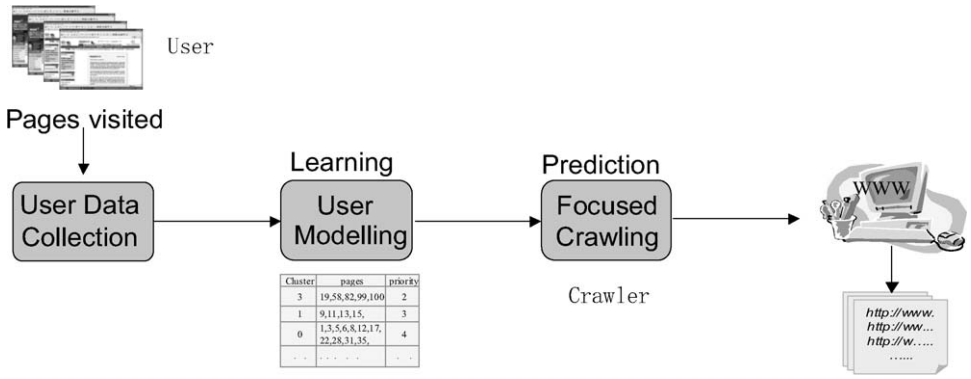

Fig. 1. System architecture: User Data Collection, User Modelling via Pattern Learning, and Focused Crawling.

as the current observable document. Probabilistic models, such as HMM, offer a good balance between simplicity and expressiveness of context.

The system consists of three components: User Data Collection, User Modelling via Pattern Learning, and Focused Crawling, as shown in Fig. 1. The following sections describe each of the components.

2.1. User Data Collection

In the User Data Collection stage, we collect the sequences of pages visited by the user during her topic-specific browsing. The user starts browsing with a given topic in mind and is intended to stay on the topic. If the user considers the current page relevant, she can click the ***Useful*** button which is added to each page (Fig. 2a), and the page will become an annotated target page. In order to represent the user browsing pattern, we construct a Web graph (Fig. 2b). A Web graph is a directed graph whose nodes correspond to pages on the Web, and whose arcs correspond to links between these pages. To construct a Web graph, each browsed page is represented by a node, and all hyperlinks between pages are added as edges between the nodes (Fig. 2b). When a user requests a Web page by typing a URL in the location bar, choosing a URL from a bookmark, or following a link to reach a new Web page, a new node will be created and all the hyperlinks between it and existing nodes will be added into the Web graph as edges. For example, if the user follows the hyperlink in page 2 to access page 3 in Fig. 2, node 3 is created and edges from $2 \rightarrow 3$ is added. Each node is associated with a browsed HTML page with a unique URL, and an edge is established if a referring document has a hyperlink pointing to a referred document. Nodes corresponding to targets are marked as double-bordered white nodes in the Web graph, for instance nodes 4 and 5. In the case of Web pages with frames, when a frame page is being requested, all of its children pages will be automatically downloaded by the browser. Therefore, instead of recording the frame page, all of its pages will be recorded in the Web graph.

Sometimes, the user may not make the best decision and follow a longer path to reach a target page. In this case, we store other possible links between nodes. Continuing the example above, if the user follows a hyperlink in page 2 to access page 3, and if page 1 links to page 3, then a link from node 1 to node 3 will be added to the Web graph as well.

The Web graph thus captures the link structure of the pages visited by the user. The way the user browses reflects the content and linkage structure leading to her targets, which is the information we try to capture. For example, path $1 \rightarrow 2 \rightarrow 4$ in Fig. 2(b). Sometimes, the user may follow some links to unrelated pages and go 'back' from them, for example, using the "Back" button on the Web browser. In this case, the page information, but not path information, is still stored in the graph. For example, the user follows the path $1 \rightarrow 2 \rightarrow 3$ in Fig. 2(b), then finds that page 3 is totally unrelated, so she goes 'back' to page 2. In this case, page 3 is still saved as part of the training data, but the move from page 3 to page 2 does not translate into a link between these pages. In other words, the Web graph records the actual hyperlinks between pages user followed, rather than user click streams. The reasoning is that the focused crawler will eventually crawl the real Web, so the Web graph should reflect the actual Web linkage structure. If the user follows some hyperlinks that eventually

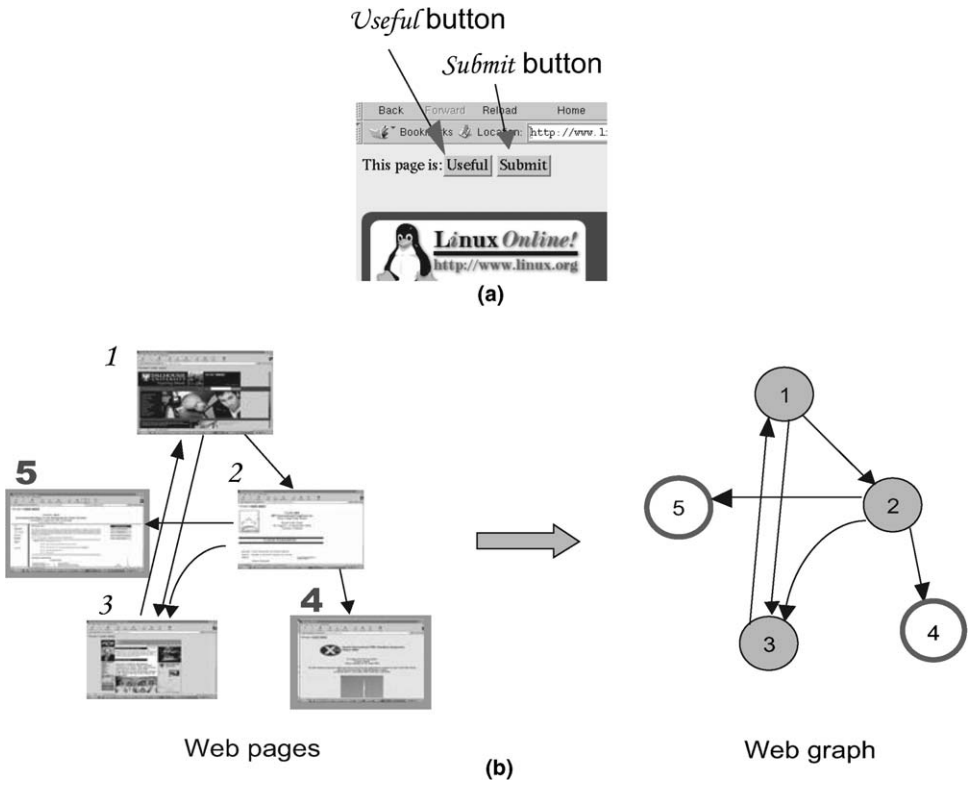

Fig. 2. User data collection. Double-bordered white nodes represent target pages: (a) Useful and Submit buttons, (b) user visited pages form Web graph.

leading to targets, these pages are considered as positive linkage examples. On the contrary, if the user follows hyperlinks that could not reach any targets, normally the user will go 'back', in this case, the pages will be treated as negative examples. However, the action of user going back is not stored, because it is not part of the linkage structure between pages.

The reasons for considering only visited pages in defining a Web graph is to reject noisy hyperlinks that may mislead the focused crawler, and to extract the dominant structure the user follows towards her topic. Furthermore, in addition to sequences of pages which the user visited, recording links between nodes in the Web graph reflects linkage structures in the real world, where focused crawling will take place. We argue that the behaviors of the user during topic-specific browsing reveal the user's real information needs.

After completing the browsing session, the user can then submit the entire session by clicking the ***Submit*** button on each browsed page (Fig. 2a). The entire Web graph consisting of all user visited pages, together with the user's annotated target information, will be saved as training data.

In our implementation, we use the Algorithmic Solutions Software LEDA package [20] to save the graph with a hash table for URLs and their corresponding document ID numbers.

2.2. Concept graph

To capture the semantic relations from the user's browsing session, we first categorize the Web pages visited by the user into different types. To do this, we first use Latent Semantic Indexing (LSI) [21,22] to identify semantic content, then apply a clustering algorithm, then use the cluster information and the user Web graph to build a concept graph. Finally a HMM is built to learn sequential patterns leading to targets from the concept graph.

2.2.1. LSI—identification of semantic content

A topic the user browses on can be characterized by the keywords contained in Web pages on the topic. Different words may be used to describe the same concepts in different pages, so we bring context into consideration by representing documents using LSI. The documents in the Web graph are processed to extract the semantic relationships among words in the collection. Latent Semantic Indexing (LSI) [21,22] captures the semantic relationships among different words on the basis of co-occurrences in the document collection. It uses the singular value decomposition (SVD) to reduce the dimensions of the term-document space. Its effectiveness has been demonstrated empirically in many information retrieval applications leading to increased average retrieval precision. The main drawback of LSI is that the computation cost for large document collections is high. However, in our system, the user visited pages form a small collection with limited topic categories, thus LSI performance limitations are less of an issue.

In detail, we implement document representation using the following steps:

(1) Form the dictionary of the collection by extracting all meaningful content words; We perform stemming, convert all characters to lower case, and truncate words with more than 20 characters.
(2) For each document and term from the dictionary, calculate its normalized TF–IDF (Term Frequency, Inverse Document Frequency) score.
(3) Construct the term-document matrix using the TF–IDF scores as entries.
(4) Apply the SVD algorithm to obtain a low dimensional LSI space representation for the document collection.

The normalization in the weighting step 2 is a scaling step designed to keep large documents with many keywords from overwhelming smaller documents. The normalized term weight for term i in document k is

$$W_{ik} = \frac{tf_{ik}\left(\log_2 \frac{N}{\mathrm{d}f_i} + 1\right)}{\sqrt{\sum_{k=1}^{T} (tf_{ik})^2 \left(\log_2 \frac{N}{\mathrm{d}f_i} + 1\right)^2}}, \tag{1}$$

where tf_{ik} is the frequency of term i in document k; N is the number of documents in the collection; $\mathrm{d}f_i$ is the number of documents in which term i appears and T is the number of terms in the collection.

In step 4, singular value decomposition breaks the term-document matrix A down into the product of three matrices,

$$A = U\Sigma V^{\mathrm{T}}, \tag{2}$$

where the columns of matrices U and V are the left and right singular vectors, respectively, and Σ is a diagonal matrix of the singular values of A sorted in decreasing order. The closest rank-k approximation to the original matrix A is constructed from the k-largest singular values. The number of reduced dimensions k and the associated singular values and matrices U_k and V_k are determined by means of an iterative process using Lanczos algorithm [23,24] with tolerance of the approximation 10^{-6}. The implementation provided in [21] is used. Within the reduced space, semantically related terms and documents presumably lie near each other so that clusters of terms and documents are more readily identified.

2.2.2. Clustering

After obtaining the low-dimensional LSI representation, pages are grouped together form semantically coherent clusters and a concept graph is created. In order to do this, a clustering algorithm is applied.

An important question is how to find the right number of clusters in the data. Therefore, we apply the X-means algorithm [25], which extends k-means with efficient estimation of the number of clusters within a given interval. We first normalize the reduced LSI document vectors, then we cluster all documents, except the pages marked as target by the user, using the X-means algorithm, where the number of clusters is in the interval [3,8]. We felt that the interval [3,8] is a good compromise between flexibility in the number of clusters and constraining that number to a range that is neither trivially small (under 3) or excessively fragmented (over 8). All user

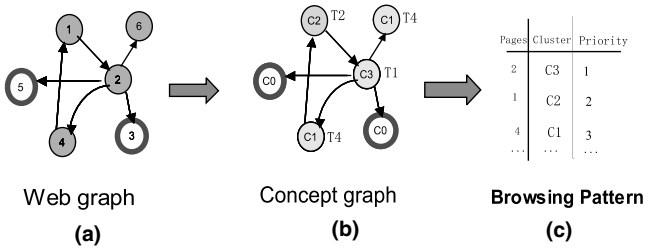

Fig. 3. User modelling via sequential pattern learning. C_i is the label of cluster i, T_j is the estimated hidden state.

marked target pages form a separate target cluster (cluster 0). They are not part of the clustering process. By clustering documents we try to capture the semantic relation between different Web pages (clusters) leading to the target pages (cluster 0).

We might use topic categorization instead of clustering based on pre-existing categories available online in the Open Directory Project (ODP)[1] or Yahoo![2] However, compared to user visited pages produced by topic-specific browsing, the categories in the ODP are too general. Furthermore, the ODP hierarchy does not reflect real-life Web linkage structure, and standard categorization in this way lacks page context information. Therefore, applying clustering within the context of user's browsing is reasonable than using standard categorization. In fact, our goal is to be able to capture the concept associative relation between different types of documents, i.e., document type A is more likely to lead to targets than document type B, rather than what exact categories they belong to.

After clustering, the associative relationships between groups are captured into a *concept graph* $G = (V, E)$, where each node is assigned to C_i, the label of the cluster it belongs to, as shown in Fig. 3(b). In the concept graph, we do not merge different nodes with the same cluster number, since doing that would cause linkage information between nodes to become lost.

3. User modelling

We model focused crawling in the following way: we assume that there is an underlying Markov chain of hidden states, from which the actual topics of the documents are generated. Each state is defined by the distance, in number of hops, of the current page to a target page. In other words, hidden states represent the closeness of a page to a target, and the actual document is the observation that is probabilistically generated. Given a sequence of observations, we predict the next state the page will be in based on the observations so far. During the crawling, those pages with lower state subscript such as T_1 will be assigned higher visit priorities than those with higher state subscript such as T_2, and those pages with the same state are then sorted by the prediction scores, as shown in Fig. 3(c). The pattern learning procedure involves the estimation of the parameters of the underlying probabilistic model (HMM), and details on HMM will be discussed in Section 3.2.

A HMM [26] is defined by a finite set of states $S = \{s_1, s_2, \ldots, s_n\}$ and a finite set of observations $O = \{o_1, o_2, \ldots, o_m\}$ associated with two conditional probability distributions $P(s_j|s_i)$ and $P(o_k|s_j)$. There is also an initial state distribution $P_0(s)$. HMMs, widely used in speech-recognition and information extraction, provide superior pattern recognition capabilities for sequential patterns. HMMs are useful when one can think of underlying unobservable events probabilistically generating surface events, that are observable.

3.1. Structure of the HMM for focused crawling

In the HMM used for focused crawling, the hidden states reflect the topology of the Web graph with respect to the target pages, while the observations reflect the content of the Web pages. The model aims to estimate the

[1] http://dmoz.org.
[2] http://www.yahoo.com.

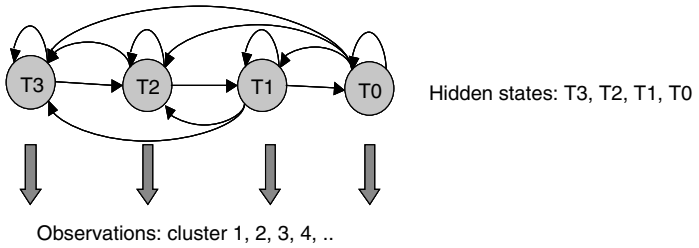

Fig. 4. The structure of a Hidden Markov Model with four hidden states.

likelihood of topics leading to a target topic directly or indirectly. The structure of the Hidden Markov Model is shown in Fig. 4.

Let n be the number of hidden states. The key quantities associated with the model are the hidden states, observations, and the parameters (initial probability distribution, transition and emission probabilities).

- Hidden states: $S = \{T_{n-1}, T_{n-2}, \ldots, T_1, T_0\}$
  - The focused crawler is assigned to be in state T_i if the current page is i hops away from a target. The state T_{n-1} represent "$n-1$" or more hops to a target page. The number of hidden states n is a parameter of the model.
- Observations: $O = \{1, 2, 3, 4, \ldots, x\}$
  - Cluster number $(1 \ldots, x)$ of an observed Web page. The number of clusters x is determined by the X-means algorithm.
- Set of HMM parameters θ:
  - Initial probability distribution

    $\pi = \{P(T_0), P(T_1), P(T_2), \ldots, P(T_{n-1})\}$.

    The probabilities of being $0, 1, \ldots, n-1$ hops away from a target page at the start of the process. Due to lack of prior information, we use the uniform distribution $\pi = \{1/n, 1/n, \ldots, 1/n\}$.
  - Matrix of transition probabilities: $A = [a_{ij}]_{n \times n}$, where a_{ij} = probability of being in state T_j at time $t+1$ given that the system is in state T_i at time t.
  - Matrix of emission probabilities: $B = [b_{ij}]_{n \times x}$, where b_{ij} = probability that the current page belongs to cluster j if the system is in state T_i.

3.2. Parameter estimation

Once the state-transition structure is defined, the model parameters are the state transition and the emission probabilities. We use annotated training data, i.e. the concept graph with the identified target pages (see Section 2.2). We "label" all nodes in the concept graph as $T_0, T_1, T_2, \ldots, T_{n-1}$ in a Breadth-First search out of the set of target pages (T_0) as shown in Fig. 5.

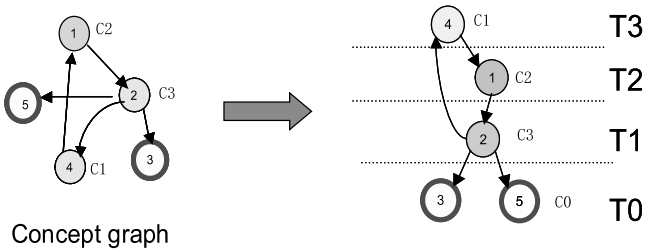

Fig. 5. Parameter estimation of HMM.

Probabilities are estimated by maximum likelihood as ratios of counts, as follows:

$$a_{ij} = \frac{|L_{ij}|}{\sum_{k=0}^{n-1} |L_{kj}|}, \tag{3}$$

where $L_{ij} = \{v \in Ti, w \in Tj : (v, w) \in E\}$

$$b_{ij} = \frac{|N_{ij}|}{\sum_{k=1}^{x} |N_{kj}|}, \tag{4}$$

where $N_{ij} = \{C_i : C_i \in Tj\}$.

3.3. Efficient inference

The task during the crawling phase is to associate a priority value with each URL that is being added to the queue. Inference using the HMM model is performed for this task. Given a downloaded Web page w_t, the URLs associated with its outgoing links are inserted into the queue and they all receive the same priority value, computed on the basis of the prediction of the state of the children of w_t.

The estimated distribution of the state of w_t, $P(S_t|O_{1,\ldots,t})$, is a probability distribution over all possible state values $T_0, T_1, T_2, \ldots, T_{n-1}$, given observations $O_{1,\ldots,t}$ and the distribution at step $t-1$, corresponding to the parent w_{t-1} of w_t. This estimation problem is known as filtering. Parent w_{t-1} of w_t is the Web page containing the URL of w_t, which was inserted earlier into the queue. The sequence $1, \ldots, t$ refers to the visited pages along the shortest path from the seed page to the current page. The sequence does not necessarily reflect the temporal order in which pages have been visited.

Based on $P(S_t|O_{1,\ldots,t})$, a prediction step is carried out to estimate the distribution of the state of the children $t+1$, which have not been seen yet, and therefore there is no observation O_{t+1} associated with them.

The probability distribution $P(S_{t+1}|O_{1,\ldots,t})$ of the state at the next step $t+1$ given observations $O_{1,\ldots,t}$ can be calculated using the following equations:

$$P(S_{t+1}|O_{1,\ldots,t}) = P(S_{t+1}|S_t)P(S_t|O_{1,\ldots,t}), \tag{5}$$

$$P(S_t|O_{1,\ldots,t}) = P(O_t|S_t)P(S_t|S_{t-1})P(S_{t-1}|O_{1,\ldots,t-1}), \tag{6}$$

where $P(S_{t+1}|S_t)$ are the transition probabilities, and $P(O_t|S_t)$ are the emission probabilities of the model.

For the prediction step, we associate values $\alpha(s_j, t)$, $j = 1, \ldots, n$, with each visited page. Forward value $\alpha(s_j, t)$ is the probability that the system is in state s_j at time t, based on all observations made thus far. Hence the values $\alpha(s_j, t)$ are the calculated values of $P(S_t|O_{1,\ldots,t})$. Given the values $\alpha(s_j, t-1)$, $j = 1, \ldots, n$ of the parent page, we can calculate the values $\alpha(s_j, t)$ using the following recursion, derived from Eq. (6):

$$\alpha(s_j, t) = b_{jk_t} \sum_{s_i} \alpha(s_i, t-1) a_{ij}, \tag{7}$$

where a_{ij} is the transition probability of state T_i to T_j from matrix A and b_{jk_t} is the emission probability of O_{k_t} from state T_j from matrix B. All the $\alpha(s_j, t)$ values for all $j = 1, \ldots, n$ are stored for each item of the priority queue.

From $\alpha(s_j, t)$, we can calculate the prediction for the future state S_{t+1} using Eq. (5). Let $\alpha'(s_j, t+1)$ be the probability that the system will be in state s_j at the next time step, given the observations thus far.

$$\alpha'(s_j, t+1) = \sum_{s_i} \alpha(s_i, t) a_{ij} \tag{8}$$

The prediction for the next step involves only the transition probabilities because there is no additional evidence.

4. Focused crawling

After the learning phase, the system is ready to start focused crawling. An overview of the crawling algorithm is shown in Fig. 6. The crawler utilizes a queue, which is initialized with the starting URL of the crawl, and keeps all candidate URLs ordered by their visit priority value. We use a timeout of 10 s for Web downloads and filter out all but pages with text/html content. The crawler downloads the page pointed to by the URL at the head of the queue, calculates its reduced dimensionality (LSI) representation, and extracts all the outlinks. Then all child pages are downloaded and classified using the K-Nearest Neighbor algorithm into one of the clusters. The prediction of future state is calculated for each parent–child pair based on the current observations and corresponding HMM parameters, and the visit priority values are assigned accordingly. The crawling algorithm with efficient inference using HMM is described in detail in Section 4.3.

Since the queue is always sorted according to the visit priority value associated with each URL, we expect that URLs at the head of the queue will locate targets more rapidly.

We also set a relevance threshold γ for determining if a Web page is relevant to the user's interests. If its maximal Cosine similarity to the target set is greater than γ, the URL is stored and presented to the user as a relevant page.

4.1. Calculation of the priority

The visit priority is determined based on the estimated future state distribution $P(S_{t+1}|O_{1,\ldots,t})$, defining a vector key $(P(T_0), P(T_1), P(T_2), \ldots, P(T_{n-1}))$, where n is the number of hidden states.

If there are two or more items of approximately equal value in the first key, the key data items are ordered in decreasing order according to the second data item to break the tie. In our system, we use a threshold $\varepsilon = 0.001$ to define the equality of two key data items, i.e., for two state distributions $X[P(T_0), P(T_1), P(T_2), \ldots, P(T_n)]$ and $Y[P(T_0), P(T_1), P(T_2), \ldots, P(T_n)]$, if $|X[P(T_0)] - Y[P(T_0)]| < \varepsilon$, the sorting process would use the second key data items pair $X[P(T_1)]$ and $Y[P(T_1)]$.

4.2. Priority queue data structure

The priority queue contains the URLs of pages w_t to be visited sorted by the priority of the parent page w_{t-1} of page w_t, defined as the page through which the URL of w_t was placed on the queue.

Each queue element consists of

- the URL of page w_t,
- cluster C_{t-1} of the parent page w_{t-1} of page w_t,
- the visit *priority* of w_t,
- probabilities $\alpha(j, t-1)$ that page w_{t-1} is in hidden state j, for all j, capturing $P(S_{t-1}|O_{1,\ldots,t-1})$.

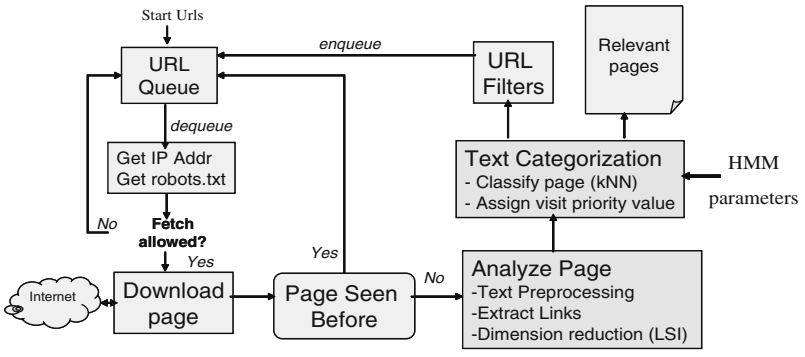

Fig. 6. Flow chart of focused crawling.

Algorithm Focused_Crawler(HMM, γ, n)
$urlQueue$:= {Seed URLs};
WHILE(not(termination)) DO
w_t := dequeue head of $urlQueue$;
extract from w_t its URL, parent cluster C_{t-1}, and
$\alpha(j, t-1)$ for all possible states j;
Download contents of w_t;
Parse and classify page w_t to cluster C_t;
IF $cos(w_t, TargetSet) > \gamma$, THEN store w_t as relevant;
Calculate $\alpha(i,t)$ and $priority$ for w_t's children
IF w_t is start Url,
THEN $\alpha(i,t) = \pi(i)\, b_{iC_t}$;
ELSE $\alpha(i,t) = \sum_j (\alpha(j,t)\, a_{ji}) b_{iC_t}$;
calculate prediction $[P(T_0), P(T_1), P(T_2), ..., P(T_n)]$;
calculate the visit *priority*;
FOR EACH outlink w_{t+1} of w_t with url (has the same priority)
$urlQueueEntry := (priority,\ url,\ C_t,\ \alpha(i,t))$;
Enqueue ($urlQueueEntry$); // entries sorted by $priority$

Fig. 7. Pseudocode of crawling algorithm.

4.3. The algorithm

The basic crawling algorithm is summarized in Fig. 7. Each Web page w_t is associated with information about its parent in the form of cluster number C_{t-1} and partial probabilities $\alpha(j, t-1)$. If page w_t is a start URL, its $\alpha(.,.)$ will be calculated using the initial matrix, otherwise, it will be calculated based on information of its parent and current observations.

5. Evaluation

5.1. Performance metrics

It is important that the focused crawler return as many relevant pages as possible while minimizing the portion of irrelevant ones. The standard measures used to evaluate the performance of a crawler are *Precision* and *Recall*.

To evaluate the effectiveness of our approach, we use *precision*, which is the *percentage* of the Web pages crawled that are relevant. Since in general there are multiple target pages marked by the user as interesting, the relevance assessment of a page p we are using is based on maximal Cosine similarity to the set of target pages T with a confidence threshold γ. That is, if $\max_{t \in T} \cos(p,t) \geqslant \gamma$ then p is considered as relevant.

The *recall* of the standard evaluation measure is the ratio of the relevant pages found by the crawler to all relevant pages on the entire Web. It is not possible to get the exact total number of relevant pages on the Web so *recall* cannot be directly measured. However, it is reasonable for the purpose of comparing different retrieval algorithms on a document corpus to estimate *recall* using all relevant pages returned by all versions of crawls strategies and start URLs.

The ability of the crawler to remain focused on topical Web pages during crawling can also be measured by the average relevance of the downloaded documents [16,17]. Average Similarity is the accumulated similarity over the number of the crawled pages. The ideal crawler will always remains focused on the topic and have the maximum possible similarity all the time. This relevance measure also has been used in [16–18,15]. In our system, since the user may mark multiple targets, if the query document is close to any one of the targets, it is considered as matching the user interests. The query document is compared to all the target documents,

and the highest similarity value is used to calculate Average Similarity, which we called the *Maximum Average Similarity* σ

$$\sigma = \max_{t \in T} \frac{\sum_{p \epsilon S} \cos(p,t)}{|S|}, \tag{9}$$

where T is the set of target pages found, S is the set of pages crawled, and

$$\cos(p,t) = \frac{\sum_{k \epsilon p \cap t} w_{pk} \cdot w_{tk}}{\sqrt{\sum_{k \epsilon p} w_{pk}^2 \sum_{k \epsilon t} w_{tk}^2}}$$

is the standard Cosine similarity function, and w_{dk} is the TF–IDF weight of reduced vector term k in document d.

5.2. Experiments

To collect training data, we selected topics from the Open Directory Project (ODP)[3] categories, which are neither too broad nor too narrow, such as Linux. Users are graduate students from computer science for each topic and they were asked to search for Web pages they are interested in related to the given topic. We also compare our focused crawler with a Best-First Search (BFS) crawler and a Context-Graph (CGS) crawler.

5.2.1. Algorithms for comparison

We now briefly describe the focused crawling algorithms against which we compare our focused crawler.

(1) Best-First Search crawler (BFS): This crawler assigns priorities to all children of the current page using standard lexical cosine similarity between the set of target pages and the content of the current page. Since we have multiple targets, the visit priority order of a URL here is based on maximal Cosine similarity, and the priority queue is sorted by this value.
(2) Context-Graph crawler (CGS): We implemented the focused crawler which is based on the Context Graph method [15], in which Naïve Bayes classifiers are built for each layer, and a category "other". We use the likelihood with a threshold of 0.5 to assign weakly matching pages to "other". When a new page is seen, it will be classified to the winning layer j it belongs to and its visit priority order is sorted by the link distance to layer 0, that is, the closer the layer j is to layer 0, the higher its visit priority. Pages in the same layer are sorted by the likelihood score. For fair comparison of the learning performance, during the focused crawling, we did not use the extra back-crawling feature which obtains and includes all immediate parents of every relevant page into the queue when it is discovered.

5.2.2. Training data

We now describe the details of collecting training data for the three focused crawlers being compared.

(1) *Building a Web graph using user visited pages for HMM crawling*: Data is collected from a Data Collection session as described in Section 2.1. Targets are all pages the user marked as relevant during browsing. To collect training data, the user is given a topic such as *Hockey* and is asked to search for Web pages she is interested in related to this topic. As we mentioned before, our approach is to capture the sequential patterns leading to targets from user's browsing and emulate them on the real Web crawling. To capture better the topical hierarchies for training, we use ODP and search engines to help collect effective training data of the user browsing behaviors when following from general to specific topics. The user is asked to start browsing from 1 or 2 levels above the given topic in the ODP or from results returned by Google using keywords on topics 1 or 2 levels above the desired topic in ODP. Note that ODP was just used

[3] http://dmoz.org/.

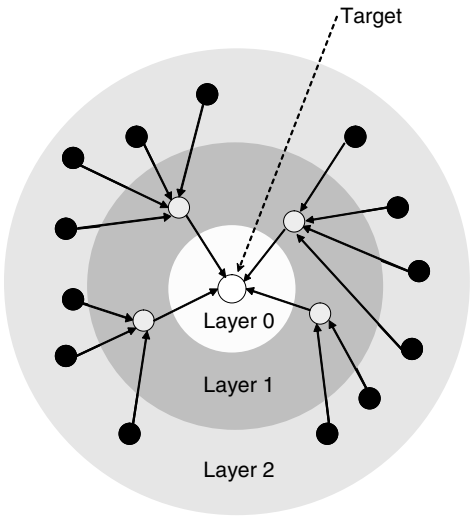

Fig. 8. Build context graph. Each circle represents one layer. Targets form layer 0, and layer i contains all the parents of the nodes in layer $i-1$.

to determine starting pages, while user browsing was performed on the actual Web. For example, for the topic *Hockey*, she may start from the *Sports* level URLs in the ODP, or type *Sports* keywords in Google and start from some of the returned URLs. As a result, we will be able to capture for training hierarchical pages sequences. We found that not sufficiently many of such sequences were captured when the user started browsing from pages too close to the target pages (e.g., if she started browsing from pages on *Hockey* in our example).

During browsing, the user clicks the *Useful* button to mark relevant Web pages, and clicks the *Submit* button to save the training data anytime.

(2) *Building a Context Graph using backlinks for Context-Graph crawling*: Data is collected using the Google [27] backlink service, starting from target pages and proceeding up to 4 layers with a maximum of 300 pages in a single layer. Layer i includes all nodes reaching targets by i hops. As shown in Fig. 8, each circle indicates one layer, target pages form layer 0, and layer i contains all the parents of the nodes in layer $i-1$. We do not check for links between nodes in the same layer and links between nodes in different layers except neighboring layers, as shown in Fig. 8.

(3) *Building a Web graph using all nodes from the Context Graph for HMM crawling*: Combination of the above two. We use all nodes collected in the context graph to build a Web graph which includes all links between the nodes, not just the ones going to the next layer.

For HMM crawling, we performed two sets of experiments using two kinds of training data (no. 1 and 3 above) for comparison. The motivation is to observe the effect on the performance of using data obtained from topic-specific browsing against using the complete Web graph.

5.3. Results

The number of relevant pages plotted against the number of pages downloaded for the topic *Linux* with relevance threshold 0.7 is shown in Fig. 9(a), and the average similarity over all downloaded pages on the same topic is shown in Fig. 9(b). In the following figures, in addition to the crawling methods we mentioned above (HMM, CGS, and BFS), we also include noLSI method which is HMM crawling without LSI dimensionality reduction. It is used to demonstrate the impact of LSI on the performance and will be explained later.

The system collected a total of 200 user browsed Web pages including 30 marked target pages. For the CGS crawler, we used the same target pages as layer 0 and constructed the context graph of depth 4. The results show that our system performs significantly better than Context-Graph crawling and Best-First crawling.

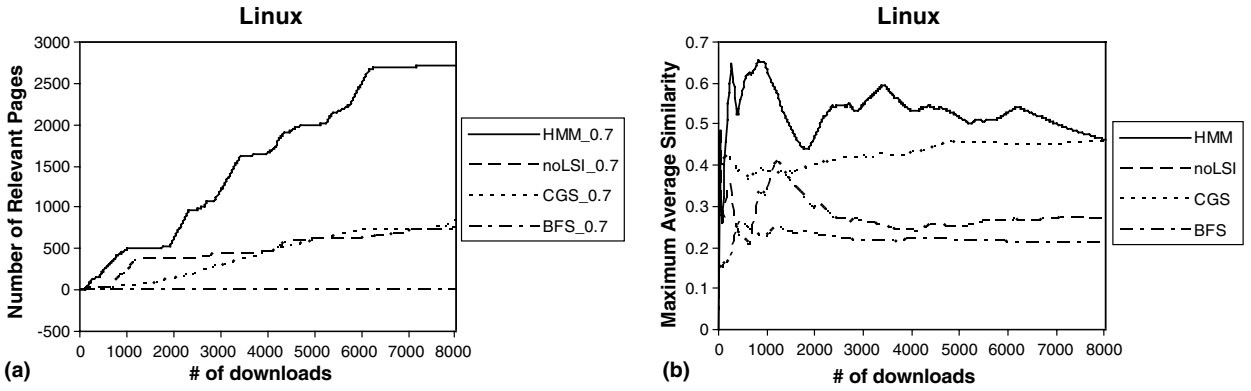

Fig. 9. Topic *Linux*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) average maximal similarities to the set of target pages of all downloaded pages.

In this case, Best-First crawling performs very poorly and when we examine closely Fig. 9(b), at the very earlier stage, Best-First crawling pursued the links that appear the most promising (with higher similarity score) at the expense of large longer term loss, whereas our focused crawler explored suboptimal links that eventually lead to larger longer term gains. A typical example during the crawls is that starting from http://www.comptechdoc.org, our system follows links quickly leading to http://www.comptechdoc.org/os/linux/manual and crawls a lot of links during the same domains, whereas Best-First crawler picks the branch leading to http://www.iceteks.com/forums/ and crawls many links under the branches that are far away from relevant pages. The URL visit order in Context-Graph crawling is based on the classification score on each layer, so the followed links include http://www.comptechdoc.org/os/linux/, http://www.comptechdoc.org/os/windows/, http://www.comptechdoc.org/independent/.

A different topic, *Call for Papers* is shown in Fig. 10(a) and (b). Again our system still showed very good performance compared to the other two. But in this case the Best-First crawler performs much better than Context-Graph crawler overall. The system collected a total of 153 user browsed Web pages and 4 marked target pages with the threshold $\gamma = 0.7$. The basic principle behind the Best-First crawler is linkage locality of the Web, that is the assumption that Web pages on a given topic are more likely to link to those on the same topic. The Context-Graph crawler captures physical layers leading to targets, however, if common hierarchies do not exist for each layer due to mixing links (cross links between topics) in the Web graph, it performs poorly, which reflects our arguments before. In contrast, our model takes both content and linkage

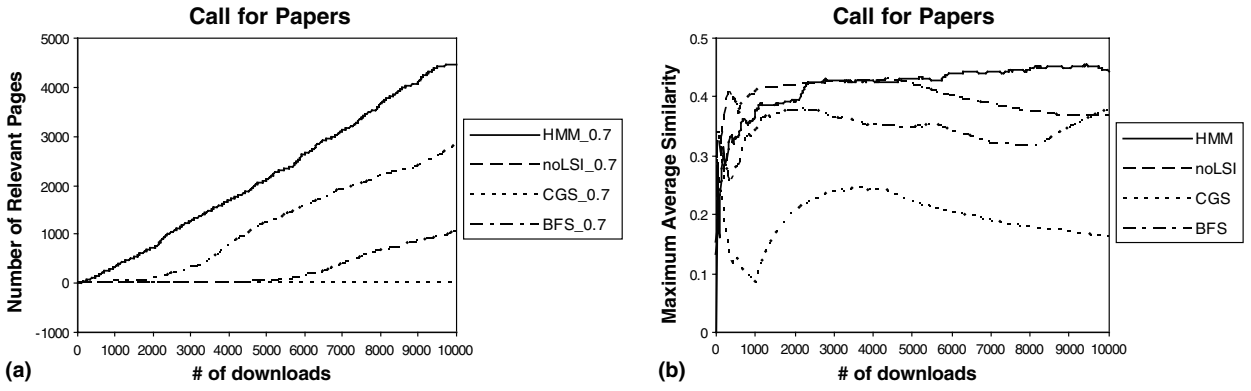

Fig. 10. Topic *Call for Papers*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) average maximal similarities to the set of target pages of all downloaded pages.

structure into account during the learning procedure in order to learn semantic relation to targets, and is shown here to work well on both cases.

Other topics, *Biking* and *Hockey*, are shown in Figs. 11 and 12. For the topic *Biking*, the system collected a total of 697 user browsed Web pages and only 1 marked target pages. Since the target page information is very specific, consequently all methods return much fewer relevant pages and less values on maximum average similarity, as shown in Fig. 11, so we lower the threshold by 0.1 for comparison. In this case, HMM crawler still performs the best for the number of relevant pages (Fig. 11a). The Context-Graph crawler initially beats HMM crawler on the maximum average similarity, but the number of relevant pages received is dramatically lower, indicating that at the earlier stage of crawl, the Context-Graph crawler gets high quality pages, but much fewer of them (Fig. 11b). In the long run, HMM crawler returns more relevant pages and shows better performance.

On the different topic *Hockey* the Best-First crawler shows very good performance compared to HMM crawler (Fig. 12). Both of them returned a large number of relevant pages with high maximum average similarities. The system collected a total of 201 user visited pages and 30 marked targets. As we mentioned before, Best-First crawler always pursues the links that have the highest cosine similarity, so if the crawler enters a large topical subgraph where topical coherence persists, it normally performs very well. The drawback in some cases is that the crawler then tends to get stuck within the subgraph or one Web site, while other crawlers more

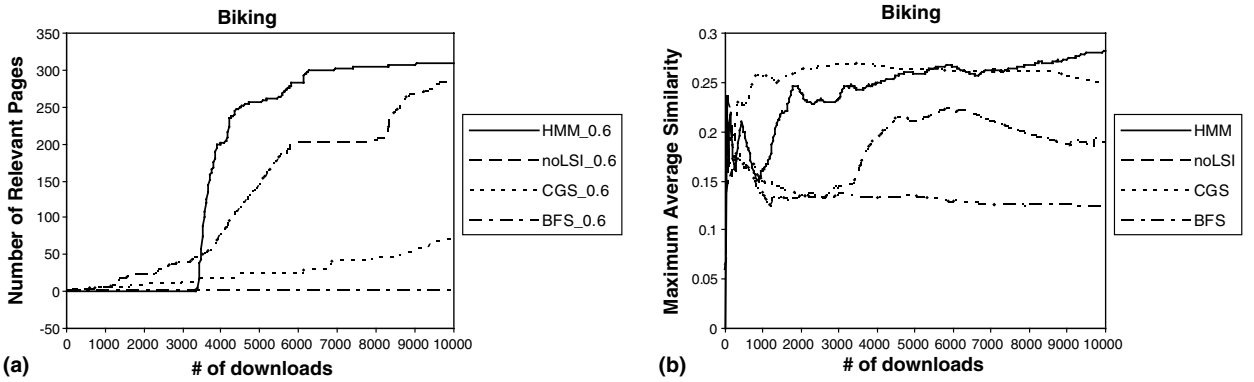

Fig. 11. Topic *Biking*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.6, (b) average maximal similarities to the set of target pages of all downloaded pages.

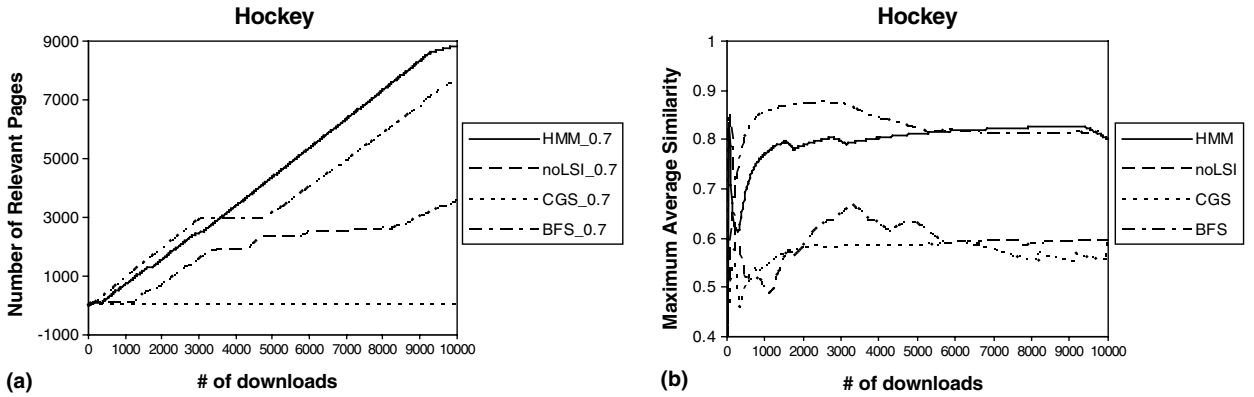

Fig. 12. Topic *Hockey*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) average maximal similarities to the set of target pages of all downloaded pages.

easily leave the subgraph, resulting in a small proportion of relevant pages immediately, but reaching to more diverse target pages in the long run.

It is also worth mentioning that in the original work on the Context Graph method [15], it was demonstrated that the linkage locality of the Web does boost the performance when including the back-crawling feature: when a page is discovered as a relevant page, its immediate parents are obtained from the search engine and added to the crawling queue automatically. We omitted this feature in order to obtain a fair comparison, since the other two methods do not have access to the feature. Apparently our system could be further enriched by combining such features.

Next we examine the impact of LSI on the performance. As we mentioned before, LSI is useful in situations where traditional lexical information retrieval approaches fail. By reducing the dimensionality of the term-document space, the underlying semantic relationships between documents are revealed, and much of the noise is eliminated. Web pages contains more noisy data than regular text documents, since they may contains irrelevant information such as advertisements. We proposed to use LSI to minimize the word-based retrieval problems. To compare the performance without using LSI, we use only the regular term-document matrix for X-means clustering as described in Section 2.2.2. The results, as shown in Figs. 9–16, demonstrated that LSI does significantly boost the performance almost all cases, except for the topic *Biking* with threshold 0.7 as shown in Fig. 16(a).

Since cosine similarity with a relevance threshold can be sensitive to the value of the threshold, we further investigate the effect of different threshold values. As shown in Figs. 9(a), 13, 10(a), and 14, our system shows

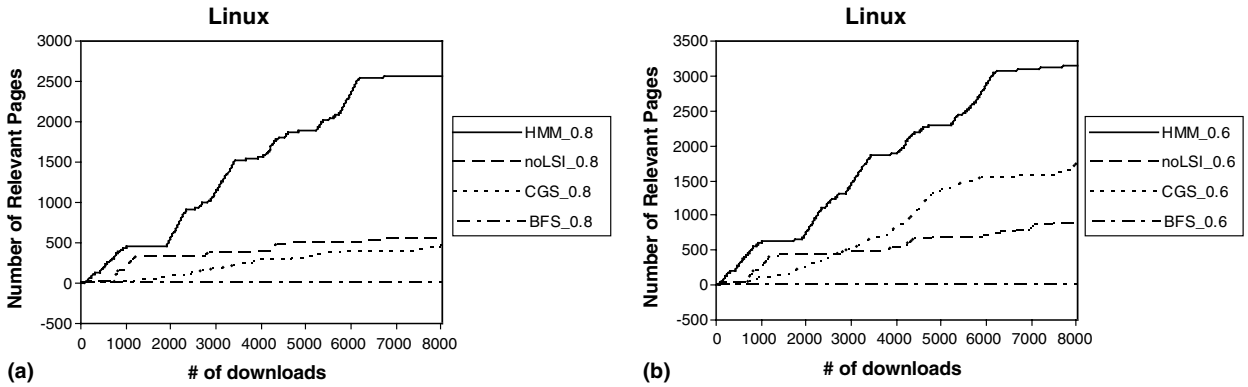

Fig. 13. The effect of different relevance threshold values on topic *Linux*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.8, (b) the number of relevant pages within the set of downloaded pages with threshold 0.6.

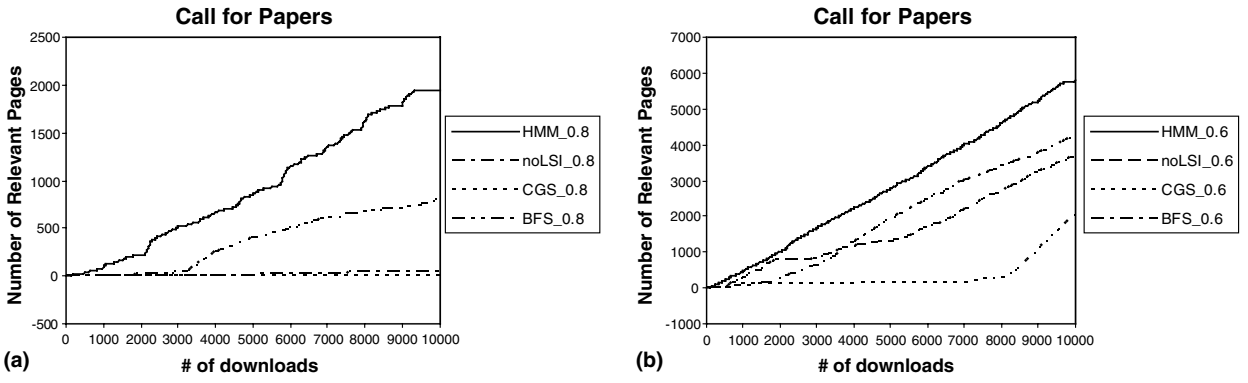

Fig. 14. The effect of different relevance threshold values on topic *Call for Papers*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.8, (b) the number of relevant pages within the set of downloaded pages with threshold 0.6.

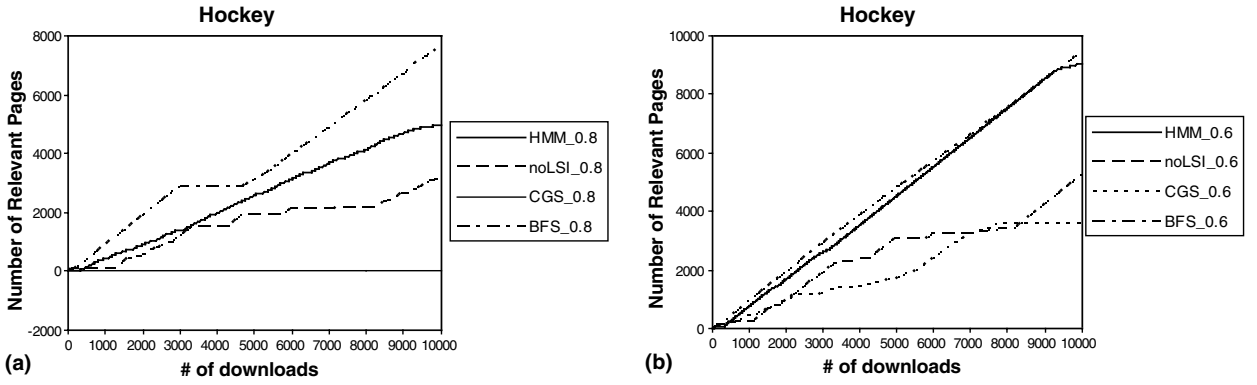

Fig. 15. The effect of different relevance threshold values on topic *Hockey*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.8, (b) the number of relevant pages within the set of downloaded pages with threshold 0.6.

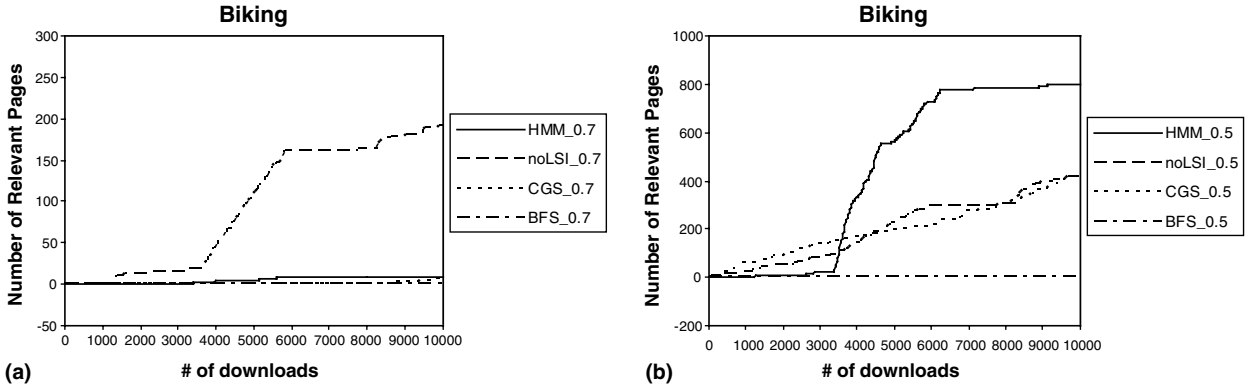

Fig. 16. The effect of different relevance threshold values on topic *Biking*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) the number of relevant pages within the set of downloaded pages with threshold 0.5.

robust results on both topic *Linux* and topic *Call for Papers* respect to change of the threshold. In fact, the better performance of HMM crawler is achieved, the higher threshold is used. For the topic *Hockey*, we saw that in Fig. 12, Best-First crawler performs almost matched HMM crawler for threshold 0.7. We observe similar behavior with threshold 0.6 and 0.8: for threshold 0.6, the behaviors of HMM and Best-First crawlers are almost identical, while for threshold 0.8, Best-First method is even better than HMM crawler (Fig. 15). On topic *Biking*, the HMM crawler showed stable behaviors on threshold 0.5 (Fig. 16b) and 0.6 (Fig. 11a), and extreme sensitivity to the choice of threshold of 0.7 (Fig. 16a). For threshold 0.7, the HMM crawler performs poorly and is outperformed by noLSI crawler. We observe that maximum average similarity is very low (see Fig. 11b) and the number of relevant pages is small for all crawlers, indicating that none of the crawlers performs well, because there is only one target page whose content is also too specific. Therefore, in this case, a higher threshold amplifies the sensitivity.

Next we observe the effect of the training data on the learning performance. We compared the performance on two types of training data described in Section 5.2.2 for HMM-learning: HMM using user data (HMM), and HMM using the context graph with all links added (HMM_alldata) as described in Section 5.2.2 (3), shown in Figs. 17 and 18. The results show that using user visited pages as training data performs best. As we expected, building the Web graph from the user's browsing pattern not only extracts the dominant link structure leading to target pages, but also incorporates the judgment of the user about the nature of the Web pages. We found that constructing the Web graph using all nodes can bring too many noisy links into the learning process, resulting in bad performance.

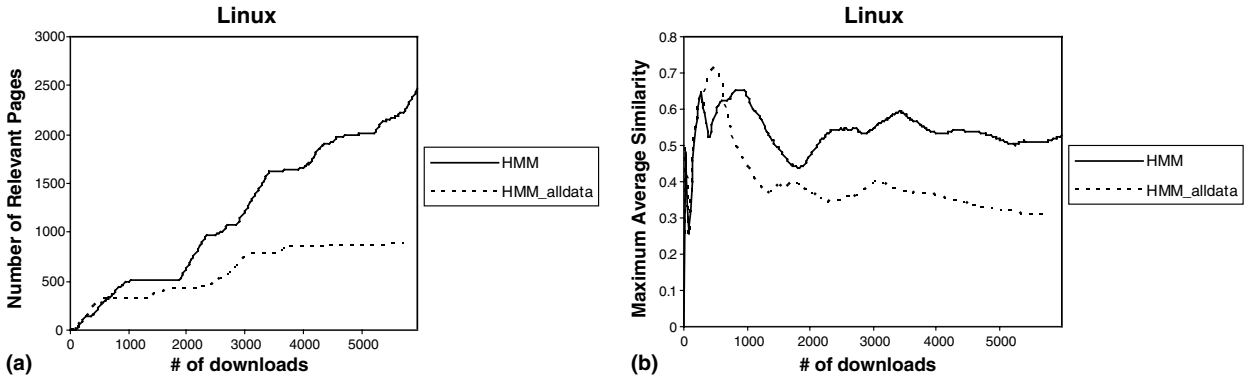

Fig. 17. Comparison of HMM crawler using different training data on topic *Linux*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data no. 1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data no. 3).

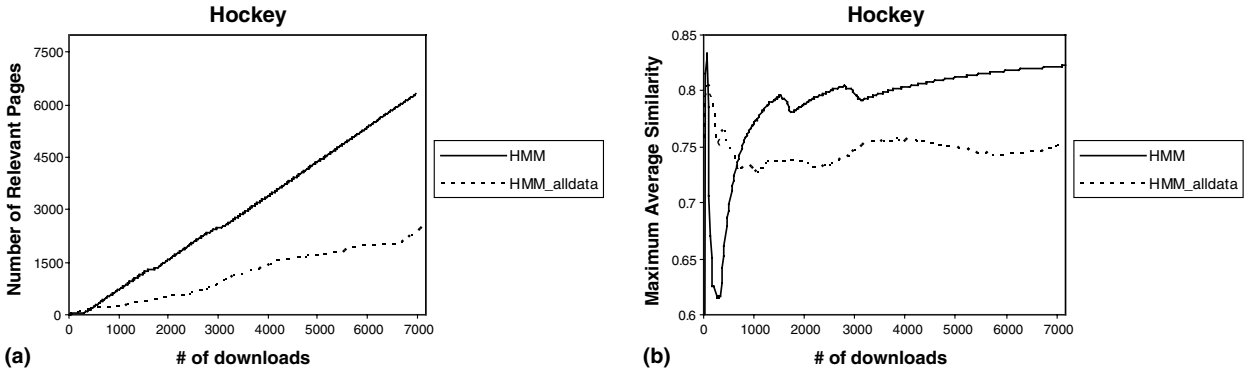

Fig. 18. Comparison of HMM crawler using different training data on topic *hockey*: (a) the number of relevant pages within the set of downloaded pages with threshold 0.7, (b) average maximal similarities to the set of target pages of all downloaded pages. HMM: HMM-learning with the Web graph using user visited pages (training data no. 1); HMM_alldata: HMM-learning with the Web graph using all nodes from Context Graph (training data no. 3).

Table 1
Recall evaluation

	topic *Linux* (%)	topic *Hockey* (%)	topic *Biking* (%)	topic *Call for Papers* (%)
HMM	61.8	44.2	46.3	53.3
noLSI	19.3	17.6	42.7	12.6
CGS	18.7	0.4	10.8	0.5
BFS	0.2	37.8	0.2	33.6

Finally we estimate the recall using all relevant pages returned by all different crawling strategies and start URLs. This is under the assumption that all relevant pages would be found within the combined output of all crawls. The recall scores for all crawls with threshold 0.7 are shown in Table 1.

6. Conclusions and discussion

Our contribution is to use HMM to model the link structure and content of documents leading to target pages. Our system is unique in several respects. We think of a focused crawler as a random surfer, over an

underlying Markov chain of hidden states, defined by the number of hops away from targets, from which the actual topics of the document are generated. The prediction is based on hidden semantic content and linkage hierarchy leading to targets instead of only physical link distance. Our experiments indicate that our system locates relevant pages quickly and is superior to standard focused crawlers.

Our user learning model is versatile, and potentially suitable for different applications. The system does not rely on other search engines and the parameters can be tailored to different users and topics. The proposed approach can potentially lead to focused crawlers that retrieve the information which is the most relevant to the user's interests. The system could be applied to collect user browsing data during daytime, and crawl relevant pages for the user during night time. The model can be used to build topic specific portals, while individual users can rely on it for personal online search.

We are exploring several directions in ongoing work. The proposed way of capturing and exploiting the user's personalized interests can potentially lead to more effective focused crawlers. Experiments showed that learning from user visited training data works better than using the complete Web graph of all nodes. It can be improved by including more and longer path sequences into the training data. In the current user browsing session, to obtain the hierarchy of topics, the user has to browse from the upper level of topical hierarchy and the paths leading to targets are short. It will be nice if the user just starts from Google with the topic keywords as usual, and the system automatically builds the upper topical hierarchy connected with user's browsing data. To make the User Modelling session more practical, we could combine backlinks and user visited pages to better present the topical hierarchy. The user will browse towards her topic in a usual way, typing keywords in Google to start browsing and marking targets as before. When the user marks a target, the system will automatically trace back the path to top page, to collect all backlink sequences of these top pages. The Web graph can be constructed from the top pages using Breadth-First search to targets, or built from the targets using the backlink service of Google API. These top pages are either Google returned results or URLs the user types in the URL bar. We expect that this combination can enhance user's browsing with topical hierarchy structure without much burden on the user. Furthermore, target pages can be updated automatically during the crawl.

The system can be extended for multiple topics. During surfing, the user marks target pages she is interested in without limiting herself to one topic. Training data can be collected from a number of separate browsing sessions. In this case, the Web graph can be comprised of several loosely connected separate subgraphs (topics). Accordingly, each subgraph can be clustered separately and patterns can be learned individually for a topic or across topics.

A major area we are investigating is the extension of the feature space. In addition to semantic content, many useful kinds of data such as anchor texts, title/meta data, URL token, and user's query keywords could be included into the learning model. Words "linux", and "hockey" in anchor texts and URLs such as http://proicehockey.about.com and http://justlinux.com/ are important features and provide potential information. We are conducting further improvements and currently exploring more sophisticated models, such as conditional random fields (CRFs) [28,29] to richly integrate more features. Including these into the current learning model is expected to boost the performance.

Acknowledgements

The authors gratefully acknowledge the financial support from the MITACS Network of Centres of Excellence, IT Interactive Services Inc., and the Natural Sciences and Engineering Research Council of Canada.

References

[1] P. Lyman, H. Varian, J. Dunn, A. Strygin, K. Swearingen, How much information? 2003. Available from: <http://www.sims.berkeley.edu/research/projects/how-much-info-2003>. Link checked on March 10, 2006.

[2] R. Zakon, Hobbes' internet timeline v7.0. Available from: <http://www.zakon.org/robert/internet/timeline>. Link checked on March 10, 2006.

[3] Search engine sizes. Available from: <http://searchenginewatch.com/reports/article.php/2156481>. Link checked on March 10, 2006.

[4] Site position and coverage. Available from: <http://www.silurian.com/sitepos/coverage.htm>. Link checked on March 10, 2006.

[5] S. Chakrabarti, M. van den Berg, B. Dom, Focused crawling: a new approach to topic-specific Web resource discovery, in: Proceedings of the 8th International WWW Conference, Toronto, Canada, 1999.
[6] D. Bergmark, C. Lagoze, A. Sbityakov, Focused crawls, tunneling, and digital libraries, in: Proceedings of the 6th European Conference on Digital Libraries, Rome, Italy, 2002.
[7] P.D. Bra, R. Post, Information retrieval in the World Wide Web: making client-base searching feasible, in: Proceedings of the 1st International WWW Conference, Geneva, Switzerland, 1994.
[8] M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, S. Ur, The Shark-search algorithm—an application: tailored Web site mapping, in: Proceedings of the 7th International WWW Conference, Brisbane, Australia, 1998.
[9] C. Aggarwal, F. Al-Garawi, P. Yu, Intelligent crawling on the World Wide Web with arbitrary predicates, in: Proceedings of the 10th International WWW Conference, Hong Kong, 2001.
[10] S. Chakrabarti, K. Punera, M. Subramanyam, Accelerated focused crawling through online relevance feedback, in: Proceedings of the 11th International WWW Conference, Hawaii, USA, 1999.
[11] J. Cho, H. Garcia-Molina, L. Page, Efficient crawling through URL ordering, in: Proceedings of the 7th World Wide Web Conference, Brisbane, Australia, 1998.
[12] K. Stamatakis, V. Karkaletsis, G. Paliouras, J. Horlock, et al., Domain-specific Web site identification: the CROSSMARC focused Web crawler, in: Proceedings of the 2nd International Workshop on Web Document Analysis (WDA2003), Edinburgh, UK, 2003.
[13] J. Rennie, A. McCallum, Using reinforcement learning to spider the Web efficiently, in: Proceedings of the 16th International Conference on Machine Learning (ICML-99), Bled, Slovenia, 1999.
[14] J. Johnson, K. Tsioutsiouliklis, C.L. Giles, Evolving strategies for focused Web crawling, in: Proceedings of the 20th International Conference on Machine Learning (ICML-2003), Washington, DC, USA, 2003.
[15] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, M. Gori, Focused crawling using context graphs, in: Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000), Cairo, Egypt, 2000.
[16] F. Menczer, G. Pant, P. Srinivasan, M. Ruiz, Evaluating topic-driven Web crawlers, in: Proceedings of the 24th Annual International ACM/SIGIR Conference, New Orleans, USA, 2001.
[17] F. Menczer, G. Pant, P. Srinivasan, Topical Web crawlers: evaluating adaptive algorithms, ACM TOIT 4 (4) (2004) 378–419.
[18] P. Srinivasan, F. Menczer, G. Pant, A general evaluation framework for topical crawlers, Information Retrieval 8 (3) (2005) 417–447.
[19] G. Pant, K. Tsioutsiouliklis, J. Johnson, C. Giles, Panorama: extending digital libraries with topical crawlers, in: Proceedings of ACM/IEEE Joint Conference on Digital Libraries (JCDL 2004), Tucson, Arizona, June 2004, pp. 142–150.
[20] Algorithmic Solutions. Available from: <http://www.algorithmic-solutions.com/enledabeschreibung.htm>. Link checked on March 10, 2006.
[21] M.W. Berry, LSI: Latent Semantic Indexing Web Site. Available from: <http://www.cs.utk.edu/~lsi/>. Link checked on March 10, 2006.
[22] S.C. Deerwester, S.T. Dumais, T.K. Landauer, G.W. Furnas, R.A. Harshman, Indexing by latent semantic analysis, Journal of the American Society of Information Science 41 (6) (1990) 391–407.
[23] M.W. Berry, Large scale singular value computations, International Journal of Supercomputer Applications 6 (1992) 13–49.
[24] M.W. Berry et al., SVDPACKC: Version 1.0 User's Guide, Technical Report CS-93-194, University of Tennessee, Knoxville, TN, October 1993.
[25] D. Pelleg, A. Moore, X-means: extending *K*-means with efficient estimation of the number of clusters, in: Proceedings of the 17th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, 2000.
[26] L.R. Rabiner, A tutorial on hidden Markov model and selected applications in speech recognition, Proceedings of the IEEE 77 (2) (1989) 257–285.
[27] Google. Available from: <http://www.google.com>. Link checked on March 10, 2006.
[28] J. Lafferty, A. McCallum, F. Pereira, Conditional random fields: probabilistic models for segmenting and labeling sequence data, in: Proceedings of the International Conference on Machine Learning (ICML-2001), Williams College, USA, 2001.
[29] D. Pinto, A. McCallum, X. Wei, W.B. Croft, Table extraction using conditional random fields, in: Proceedings of the 26th Annual International ACM SIGIR Conference, Toronto, Canada, 2003.

Hongyu Liu received her B.S. degree in Computer Science from Capital Normal University, China. She enjoyed her teaching experience for seven years and she also co-authored two books published by Xueyuan Press in China. Now she is pursuing her Ph.D. degree in Dalhousie University, Halifax, Canada. Her current research interests include machine learning, information retrieval, Web mining, and personalization.

Jeannette Janssen is an associate professor in the department of Mathematics and Statistics at Dalhousie University. Her current research interests are in stochastic models for real-life complex networks, polyhedral methods applied to combinatorial problems, and graph colouring.

Evangelos Milios received a diploma in Electrical Engineering from the National Technical University of Athens, Greece, and Master's and Ph.D. degrees in Electrical Engineering and Computer Science from the Massachusetts Institute of Technology, Cambridge, Massachusetts. While at M.I.T., he was a member of Digital Signal Processing group and he worked on acoustic signal interpretation at the M.I.T. Lincoln Laboratory. After spending five years doing research on shape analysis, sensor-based robotics, and signal processing in the Department of Computer Science, University of Toronto, he joined York University as an Associate Professor. Since July of 1998 he has been with the Faculty of Computer Science, Dalhousie University, Halifax, Nova Scotia, where he is Professor. He was Director of the Graduate Program (1999–2002). He is a Senior Member of the IEEE. He served as a member of the ACM Dissertation Award committee (1990–1992) and a member of the AAAI/SIGART Doctoral Consortium Committee (1997–2001). He has published on the processing, interpretation and use of visual and range signals for landmark-based navigation and map construction in single- and multi-agent robotics. His current research activity is centered on mining Networked Information Spaces, combining text and link information.