Probabilistic models for focused web crawling

Liu, Hongyu; Milios, Evangelos

Computational Intelligence, 2010-12-01

Probabilistic Models for Focused Web Crawling

Hongyu Liu, Evangelos Milios

October 9, 2010

Abstract

A focused crawler is an efficient tool used 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. Focused crawlers can only use information obtained from previously crawled pages to estimate the relevance of a newly seen URL. Therefore, good performance depends on powerful modeling of context as well as the quality of the current observations. To address this challenge, we propose capturing sequential patterns along paths leading to targets based on probabilistic models. We model the process of crawling by a walk along an underlying chain of hidden states, defined by hop distance from target pages, from which the actual topics of the documents are observed. When a new document is seen, prediction amounts to estimating the distance of this document from a target. Within this framework, we propose two probabilistic models for focused crawling, Maximum Entropy Markov Model (MEMM) and Linear-chain Conditional Random Field (CRF). With MEMM, we exploit multiple overlapping features, such as anchor text, to represent useful context and form a chain of local classifier models. With CRF, a form of undirected graphical models, we focus on obtaining global optimal solutions along the sequences by taking advantage not only of text content, but also of linkage relations. We conclude with an experimental validation and comparison with focused crawling based on Best-First Search (BFS), Hidden Markov Model (HMM), and Context-graph Search (CGS).

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. Nowadays, almost everyone uses search engines to find information on the Web. At the back end of a search engine, a generic crawler downloads Web pages that it deems worth including in the search engine, and makes them available to an indexer. The rapid growth and dynamic nature of the Web and the limitations of network and storage resources pose many challenges to generic crawlers. It is generally believed that search engines only index a small fraction of the Web [52, 45]. Due to this limitation, topic-specific search tools have developed, supported by topic-specific, or

focused, crawlers, which collect only pages relevant to specific topics. An application of extending digital libraries with a focused crawler can be found in [36].

The success of topic-specific search tools depends on their ability to locate topic-specific pages on the Web while using limited storage and network resources. Sometimes, relevant pages link to other relevant ones. However, it is very common that pages which are not on topic lead to topical pages. 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. The common approach to focused crawling is to use information obtained from previously crawled pages to estimate the relevance of a newly seen URL. The effectiveness of the focused crawler depends on the accuracy of this estimation process.

In the majority of focused crawlers in the literature, although different techniques and heuristics to guide focused crawling were used, the underlying paradigm is best-first search strategy, that is, to train a learner with *local* features collected about relevant nodes from the immediate vicinity of a hyperlink $u \rightarrow v$ (i.e., the parent pages and sibling pages). This is based on the hypothesis that in the Web graph, Web pages on a given topic are more likely to link to those on the same topic (Linkage Locality), and if a Web page points to certain Web pages on a given topic, then it is more likely to point to other pages on the same topic (Sibling Locality). In other words, topics appear clustered in the Web graph [12, 29].

A number of focused crawlers reported in the literature has relied on the text context of the links in visited pages to set the visit priority of the links. InfoSpiders [30] is a collection of autonomous adaptive goal-driven crawler agents which search for pages relevant to the topic, using evolving query keyword vectors and Neural Networks with Q-learning to decide which links to follow. An agent estimates the visit priority of the links in the current page by considering the text surrounding those links. A complete framework to evaluate several crawling strategies is described in [32, 31, 47]. In these studies it was found that Best-First crawling gives competitive performance to more sophisticated strategies. Aggarwal *et al.* proposed an "intelligent crawling" framework [1]. The method involves looking for specific features in a page to rank the candidate links. The visit priority value of the candidate link is calculated by a linear combination of these weighted features. More recent work [34, 35] systematically studied the use of different classification algorithms to guide topical crawlers based on text content and link-related contexts such as anchor text. It was found that a crawler using a combination of anchor text and the entire parent page performs significantly better than a crawler that depends on just one of those cues. The introduction of web page genre in focused crawling is proposed in [13], where it is demonstrated that for a specific genre of pages significant improvement in precision and recall can be obtained. The success of the method clearly depends on the ability to define the characteristics of the genre of interest. A hierarchical taxonomy of topics in both English and Chinese is used for cross-language focused crawling for topics in the taxonomy, using page content, anchor text, URL addresses and link types in [10].

Several research projects have attempted to introduce link structure into focused crawlers. PageRank is a good ordering metric in generic crawling. However it performs poorly when the task is to locate pages that are relevant to a particular topic

or query [32]. Topic-specific PageRank has been proposed as a way to incorporate the user's context in the ranking of the search results [18]. However, the topics must be predefined, based on resources such as the Open Directory Project (ODP)[1]. A user query is classified into one of the predefined topics, and ranking is done based on the pre-computed PageRank for the topic. The space of potential focused crawling strategies is explored in [20]. Good strategies evolve based on the text and link structure of the referring pages. The strategies produce a rank function which is a weighted sum of several scoring functions about a page u, such as Hub score, Authority score, Link community scores (in-links and out-links), and SVM classification scores of the ancestor pages. A Genetic algorithm is used to evolve the weights from training data.

Recent efforts to combine text and link analysis for focused crawling are described in [3, 5]. In [3], the term-document matrix for document representation is expanded to include as additional "terms" (rows) already visited documents, to which the represented document is linked. The represented documents (columns) include the seed corpus of relevant documents, the already fetched documents, and documents on the crawl frontier. Latent Semantic Indexing is applied to the expanded term-document matrix, to define a low-dimensional space in which the similarity between documents in the crawl frontier and the driving query is measured. Issues with this approach include the computational cost of updating the expanded term-document matrix, and the scalability of the method, requiring ways to limit the growth of the matrix as the crawl progresses. The method was evaluated only on bounded corpora (WebKB and Cora). In [5], the approach of [14] is extended by making the *link classifier* of [4], which is trained to assign a score to a link that corresponds to the distance from a target page that link leads to, adaptive during the crawl. The link classifier relies on features consisting of the words in the neighbourhood of links (anchor and text around the link), and the URL. The advantage of our approach is the explicit capture of sequential patterns. The introduction of a link classifier into our approach has potential as a promising future research direction.

The focused crawlers that are closest related to the ones proposed in this article, employ link structure aiming to capture link paths leading to target pages, that are longer than one hop. In [40], crawlers are modelled as autonomous agents that learn to choose optimal actions to achieve their goal, in a reinforcement learning framework. The reward for a hyperlink traversal is the cumulative estimate of the number of relevant pages starting from the current page that can be found as the result of following this hyperlink, calculated recursively by the immediate reward value from the hyperlink traversal, plus the discounted estimated reward from the successor page. The Context Graph method [14] uses the text of a page to estimate its link distance to target pages. For training, the method collects paths leading to given relevant pages, following backlinks to a certain distance to build up a context graph for the goal page, where a layer is defined by a given distance from a relevant page. A Naïve Bayes classifier is trained for each layer based on the text of the pages in it. As a new document u is found, it is classified into the layer corresponding to the estimated link distance from u to the target page. Documents classified into layers

[1] `http://www.dmoz.org/`

closer to the target are crawled first.

Two issues remain unaddressed by the focused crawling literature. One is that the assumption that all pages in a certain layer defined by a target document belong to the same topic described by a set of terms does not always hold. As pointed out in [7], the number of distracting outlinks emerging from even fairly relevant pages has grown substantially since the early days of Web authoring. Second, there is no discrimination among different links in a page. Since only a fraction of outlinks from a page are worth following, offering additional guidance to the crawler based on local features in the page to rule out some unimportant links can be helpful. One exception is a simple focused crawler, which is built on the assumption that a relevant page can lead to other relevant pages downstream even though immediate descendants are not very relevant [50]. A degree of relatedness is defined, which determines how far downstream to expand a given page: the higher the degree of relatedness, the farther a page is expanded.

The work of [14] inspired our idea to capture *longer* path information leading to targets. We believe that, in practice, there is wide variation in the number of links (hops) leading to related topics. However, the sequence information along the path leading to the targets can be further exploited to capture both content and linkage relations. This provides the motivation to model focused crawling as a sequential task. We model focused crawler as a surfer or agent, moving from one state to another state based on the actual pages as observations. The state of a web page is defined as the number of hops from it to a target page, and the web page is the observation. The use of Hidden Markov Models (HMM) for capturing path information leading to targets from the user's browsing behaviour on specific topics was explored in our previous work [22], followed up by [46]. Documents were grouped semantically by a clustering algorithm to build a concept graph forming the basis of recognition of sequences of topics leading to targets. Observations are the cluster identifiers to which the observed pages belong, and the corresponding hidden states are based on the hop distance from the target. HMMs make a number of assumptions: the next state depends only on the current state (Markov assumption); the state transition and emission matrices do not change with time (stationarity assumption); the current observation is independent of the previous observations given the current state. The work presented here represents efforts to relax these strict HMM assumptions.

Determination of the relevance of a visited page to a topic of focus has been addressed as a classification problem in [14, 4, 5, 11]. In our work we chose the simple approach of a test on the cosine distance between a visited page and the set of initial target pages. The reason is that it is difficult to build a robust classifier for a highly unbalanced classification problem, with a very small training set of relevant pages. Introducing a relevant page classifier to our approach where sufficient training data is available is a future extension. Ontology-based relevance is used in [16].

2 Overview of the proposed approach

We model focused crawling as a sequential task and learn the linkage patterns by using a combination of content analysis and link structure of paths leading to targets. Examples of sequential patterns can be found in human-authored topical hierarchies such as the ODP or Yahoo directory. In the real Web, we conjecture that such sequential patterns exist, but in a more implicit way. Due to the small world nature of the Web [2], 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. This is often the result of web masters following some general design rules to organize the pages of a Web site semantically or hierarchically. For example, university pages point to department pages, then to pages of faculty members; they rarely point to home gardening pages. Although different university Web sites may have different style and content presentations on the surface, the $Homepage \rightarrow Department \rightarrow People \rightarrow Faculty \rightarrow Research$ is a very common underlying sequential pattern. In other words, off-topic, but semantically related, pages may often lead reliably to relevant pages.

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. That is, there is significant amount of information in the sequential patterns of links in addition to content of individual pages. Therefore, our hypothesis is that by learning such sequential linkage patterns hidden in the Web graph during crawling, a focused crawler may be able to follow links leading to targets more effectively.

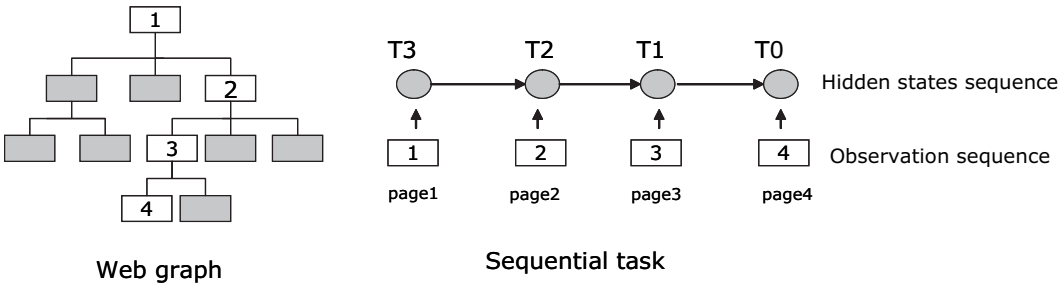


Figure 1: Our Approach: Model Focused Crawling as a Sequential Task, over an underlying chain of hidden states defined by hop distance from targets.

Our approach is unique in the following important ways. We model focused crawling as a sequential task, over an underlying chain of hidden states, defined by hop distance from targets, from which the actual documents are observed. As shown in Fig. 1, suppose page 4 is a target page, and the sequence $o = page1 \rightarrow page2 \rightarrow page3 \rightarrow page4$ is an observed page sequence leading to the target in the graph. Then $s = T_3, T_2, T_1, T_0$ is the corresponding sequence of hidden states, indicating the number of hops each page in the sequence is away from the target. After training, our system may have learned patterns like "University pages are more likely to lead to

Research papers than Sports pages". When a new document is seen, the prediction task is to estimate how many hops this document is away from a target based on the observations so far, and the crawler always follows the most promising link. The use of probabilistic finite-state 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 $p3$. Note that this is different from other systems in that we capture content and linkage structure relations along paths as sequential patterns to make predictions.

2.1 Applying Probabilistic Graphical Models

To capture such sequential patterns, we propose to apply probabilistic models for focused crawling. Probabilistic Graphical Models are a natural tool for reasoning with conditional probability distributions. The nodes in the graph represent random variables (either observed or hidden), and the links indicate conditional dependencies between random variables. The graphical model not only gives a graphical representation of the joint probability distributions, but also provides inference methods for estimating the probability distribution based on observing a subset of the random variables. The graph can either be directed, as in Bayesian Networks or Belief Networks (BNs), or undirected, as in Markov Random Fields (MRFs) or Markov networks.

2.1.1 Directed Graphical Models

A directed graphical model is defined on an acyclic graph $G = \{V, E\}$ where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes and E is a set of links between nodes with directions. Each node v_i in the graph represents a random variable and has a set of parent nodes $parent(v_i)$. The structure of the directed graph represents the conditional dependence between random variables. Namely, the joint probability over variables $V = \{v_1, v_2, ..., v_n\}$ can be calculated as the product of the conditional probability of each variable conditioned on its parents, that is,

$$p(v_1, v_2, ..., v_n) = \prod_{v_i \in V} p(v_i | parent(v_i)) \tag{1}$$

MEMMs [39, 27] are a variation on the traditional Hidden Markov Models (HMMs). They are discriminative models that attempt to estimate the probability of the hidden states directly given the data, as shown in Fig. 2(a). Unlike generative HMMs, MEMMs maximize the conditional probability of the state sequence given the observation sequence, by expressing the conditional probability as a weighted sum of a set of features, and training a model to estimate the weights of these features that are consistent with the training data. These features can interact. Examples of such features for Web pages would be word identity, keywords in anchor surrounding text, keywords contained in the title/meta data of current page.

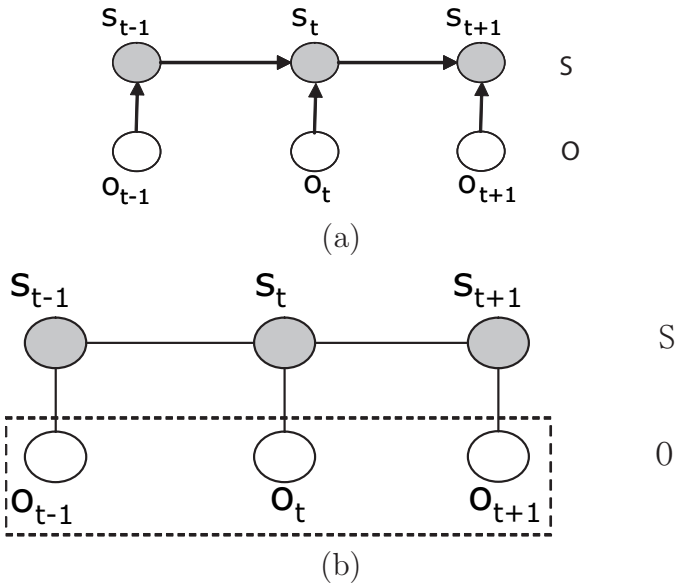


Figure 2: Dependency Graphical structures for sequences, with state sequence $s = \{s_1, s_2, ..., s_n\}$ and input sequence $o = \{o_1, o_2, ..., o_n\}$, and t ranging over input positions. (a) MEMM, arrow shows dependency (cause) (b) The dashed box over o's denotes the sets of observation sequence variables. Although we have shown links only to observations at the same step, the state nodes can depend on observations at any time step.

Intuitively, the principle of maximum entropy is simple: model all that is known and assume nothing about that which is unknown. In other words, given a collection of facts, choose a model consistent with all the facts, but otherwise as uniform as possible. Accordingly, the model seeks to maximize the entropy of the posterior conditional distribution subject to the constraint that the expected values of certain feature functions as predicted by the model should comply with their corresponding empirical frequencies observed in a training set. Namely, we would like to choose an optimal model so as to minimize the difference between the predicted values and the observed values.

Training an MEMM involves maximizing the conditional probability, $p(s|o)$, of state sequences $s = s_1, s_2, ..., s_n$ given observation sequences $o = o_1, o_2, ..., o_n$, rather than the joint probability $p(s, o)$. MEMMs consider the observation to be conditioned upon the state rather than generated by it, which allows us to use many, arbitrary, overlapping features of the observation. At every position of the observation sequence, MEMMs specify a set of distributions of possible states based on the observed feature values in the form of an exponential model. Each distribution function uses per-state normalization to define the conditional probability of possible next states given the current state and the next observation element. The conditional probability in

MEMMs is defined as

$$p(s|o) = \prod_{j=1}^{n} \frac{1}{z_j} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{j-1}, s_j, o_j)) \quad (2)$$

where z_j is a normalizing factor over the j^{th} position in the sequence, the f_i are arbitrary features and λ_i is the weight of the feature f_i. To apply this approach to the focused crawling problem, the hidden states are based on hop distance from the target and observations are a set of pre-defined feature values of observed pages, such as anchor text, URLs, and keywords extracted from the pages. Detailed descriptions of the features used in focused crawling are given in Section 4.

2.1.2 Undirected Graphical Models

An undirected graphical model is defined on a graph $G = \{V, E\}$ where $V = \{v_1, v_2, ..., v_n\}$ is a set of nodes and E is a set of undirected edges between nodes. A single node v_i is independent of all the other nodes in the graph, given its neighbors.

Linear-chain Conditional Random Fields (CRFs) are one type of widely used undirected graphical models for sequential data. They have been proven very effective in many applications including POS tagging, information extraction, document summarization and shallow parsing [21, 38, 37, 44, 43]. Given the graphical representation in Fig. 2(b), the conditional distribution $p(s|o)$ in CRFs is defined as a product of exponential functions of the features:

$$p(s|o) = \frac{1}{Z(o)} \exp\left(\sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_j f_i(s_{j-1}, s_j, o)\right) \quad (3)$$

where $Z(o)$ is a normalization factor based on the observation sequence $o = o_1, o_2, ..., o_n$:

$$Z(o) = \sum_{s' \in |S|^n} \exp\left(\sum_{j=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s_{j-1}, s', o)\right) \quad (4)$$

Therefore, CRFs define a single distribution $p(s|o)$ over the entire state sequence $s = s_1, s_2, ..., s_n$ given the observation sequence $o = o_1, o_2, ..., o_n$, rather than defining per-state distributions over the next states, given the current state at each position, as in MEMMs. In Section 5 we describe the application of CRFs to focused crawling by defining hidden states as $\{T_3, T_2, T_1, T_0\}$ and observations as a set of pre-defined feature values of observed pages, such as anchor text, URLs, and keywords extracted from the pages.

3 System Architecture Overview

The proposed focused crawling system is based on a 3-tier architecture: Data Collection, Pattern Learning and Focused Crawling, as shown in Fig. 3. In our formulation

the target pages are Web pages the user specified or extracted from DMOZ[2]. Given the target pages, *Data Collection* involves the collection of Web pages to build a Web graph and generation of page sequences for training. *Pattern Learning* involves the extraction of features from these sequences and learning of model parameters. *Focused Crawling* has the pattern parameters as input, and crawls the Web looking for relevant pages. When a new page is seen, the model predicts the distance leading to the targets, which determines a visit priority for crawling.

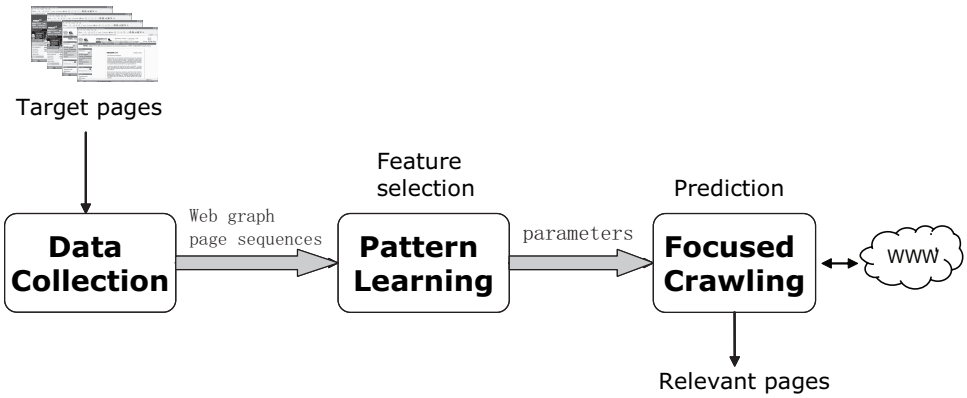


Figure 3: System Architecture: Data Collection, Pattern Learning, and Focused Crawling.

3.1 Data Collection

As in any supervised learning problem, we need to collect labeled page sequences to be used as training data. For each page sequence, the label or state sequence has to be recorded as well. In our system, the hidden state of each page is defined as T_i, where i represents the smallest number of hops required to reach a target from this page. Target nodes are always in state T_0. For instance, a page sequence is $p_1 \rightarrow p_2 \rightarrow p_3$ and its corresponding state sequence is $T_3 \rightarrow T_2 \rightarrow T_1$. All these labeled page sequences form the training data.

We collect page sequences from the local Web graph constructed using the Yahoo API in-link service[3]. The target pages are selected from ODP and the graph is created backwards by finding backlinks pointing to them from the Web, up to 4 layers.

Web data collection consists of two components: Local Web graph creation and Page sequence extraction. It takes specified target pages as input, and builds the local Web graph from bottom to top using the Yahoo in-link service. Page sequences are sampled randomly directly from the constructed local Web graph. The range

[2] `http://www.dmoz.org/`

[3] `http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html`

of sequence length values is based on the depth of the Yahoo! directory, which is approximately 6. We allow for additional hops to accommodate potential side and backward steps, so the sampled sequences have length between 2 to 10. As an example, on the right side in Fig. 4, the input is node 0 representing the pre-specified target Web page on a particular topic, and the outputs are all page sequences extracted from collected local Web graph, such as page sequence $7 \rightarrow 3 \rightarrow 1 \rightarrow 0$. Node 0 is the initial pre-specified target page, and node 5 and 7 are target pages that are marked after the whole graph is created.

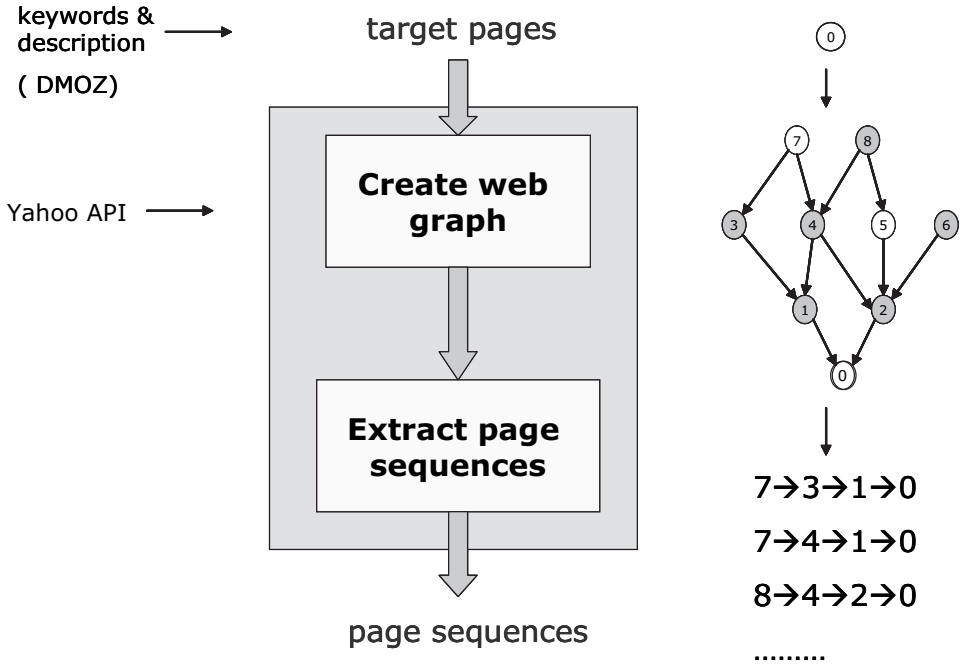


Figure 4: Web Data Collection: it takes specified target pages as input, creates Web graph using Yahoo API, outputs extracted sequences of pages as training data. It consists of two components: Create Web graph and Extract page sequences.

Selection of Target Pages on Topics. The initial target pages are selected pages on a specified topic used in both the training and crawling procedures. They can be either manually created by the users for their own target pages or user specified from existing pages. In the system we propose, for each topic we chose target pages from the ODP. All the initial target pages we used in our experiments can be found in [23].

Topic specification plays a very important role in evaluating focused crawlers. Ideally, pages pages should be judged as relevant by real users, which is not practical for thousands of pages visited in focused crawling. Furthermore, topics can be obtained from different sources and can be defined at different levels of specificity: for example, the general topic "sports" or the more specific topic "Ice Hockey World Championships 2005". Therefore, as in [47], we select topics from an existing hierar-

chical concept index such as the ODP, and pick topics which are neither too general nor too specific. An example of such a topic is *House Plants*, to be found under the ODP topic hierarchy *Home - Gardening - Plants - House Plants*.

During the process of creating the local web graph based on the initial target pages, all the Web pages collected are tested on whether they are sufficiently similar to the initial target pages to be considered as additional target pages. Cosine similarity is used for this purpose with the *training data collection threshold* of 0.8. Therefore, the initial target pages and the target pages that are added later form the final target page set for training.

Creation of Web Graph. In order to capture the page sequences leading to targets for training, first we construct a local Web graph to represent the content and linkage structure associated with the targets (Fig. 5).

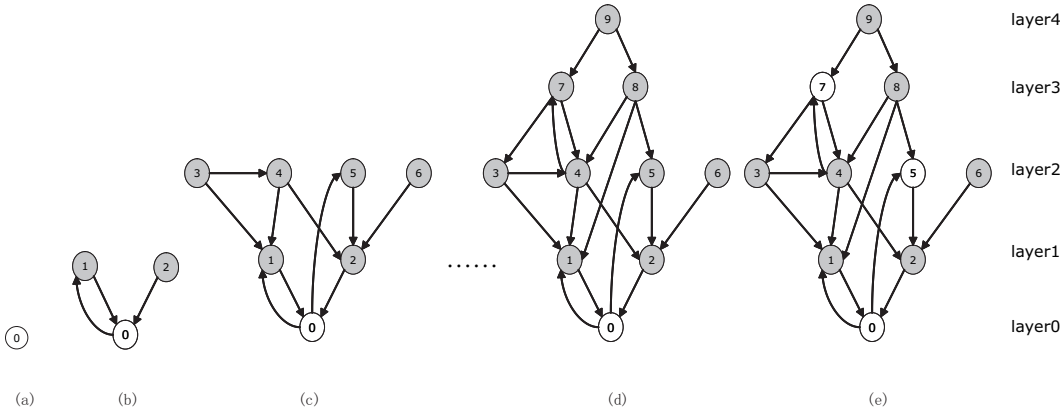


Figure 5: Creation of Web graph layer by layer using Yahoo API inlink service. White nodes represent target pages, and grey nodes represent pages obtained through backlink service tracing with Yahoo API. Node 0 is the initial target page, and nodes 5 and 7 are target pages that are marked as such by virtue of their similarity to the initial target pages.

To construct a local Web graph, each Web page is represented by a node, and all hyperlinks between pages are added as edges between the nodes. When a new Web page is found and added to the existing graph, 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, when page 5 in Fig. 5(c) is added to the graph with existing nodes 0, 1, 2, 3, and 4, then node 5 is created and edges from $5 \rightarrow 2$ and $0 \rightarrow 5$ will be added.

The local Web graph is created layer-by-layer starting from one or more user-specified target pages, say node 0, which is obtained from ODP described in the section above (Fig. 5). Note that we only use ODP to select initial target pages on the topic, rather than use the ODP hierarchy for training. ODP is a manually created hierarchical concept directory, which does not reflect the Web linkage structure in the real world. On the other hand, focused crawling is designed to retrieve Web pages

from the actual Web, therefore we have to collect training data from the real Web to reflect the actual Web linkage structure and its dynamic nature. To do so, we make use of the inlink service from Yahoo Web APIs service[4], which lists Web pages that have links to the specified Web page. Starting from the layer 0 with user-specified target page(s), the graph is created from bottom to top, up to layer 4.

There are no repeated nodes in the graph. When a new URL is seen, if it is not in the graph, both a new node and all corresponding edges (hyperlinks) are added into the graph; if it is already included in the graph, only the edges between it and existing nodes are added. For instance, when node 5 is added to the graph as shown in Fig. 5(c), and the URL of node 0 is found as the inlink of node 5 by Yahoo inlink service, then only the edge from $0 \rightarrow 5$ will be added to the graph, due to the fact that node 0 is already in the graph. Since some Web pages may have hundreds of backlinks pointing to it, while others may only have a few, we limit the total number of nodes in each layer to a maximum of 600 and average the number of backlinks for each node. For example, if the current layer i contains 200 nodes, then the number of the backlinks for each node is limited to $600/200 = 3$, although some nodes may have hundreds of inlinks found by Yahoo API. As a result, the graph is created in a more balanced manner in terms of the number of nodes in each layer and the number of the backlinks for each node in order to extract page sequences effectively for training. We follow [14] in selecting the number of layers.

After the graph is created, all nodes are also classified to mark new targets based on the cosine similarity between them and the initial target pages, such as nodes 5 and 7 in Fig. 5(e). If the similarity between a node and one of the initial target pages is equal to or greater than 0.8, it is marked as the target as well. In Fig. 5, nodes corresponding to targets are marked as white nodes in the local Web graph, for instance nodes 0, 5 and 7, and grey nodes represent the others. The local Web graph thus captures sufficient content and link structure of the pages leading to targets, which is the information we try to capture. The motivation is that the focused crawler will eventually crawl the real Web, so the local Web graph should reflect the actual Web linkage structure. The number of nodes in each layer chosen is twice the number used in [14].

Extraction of Page Sequences and State Sequences. Given the created local Web graph, the next step is to extract page sequences from the graph.

The nature of the graph shows that one node may have multiple child URLs. For example, node 8 has nodes 4, 1, 5 as its children in the right side of Fig. 5. If we extract sequences following all the arcs in the graph, we may get many duplicated paths in the sequences, such as $9 \rightarrow 8 \rightarrow 4$ in the sequences $9 \rightarrow 8 \rightarrow 4 \rightarrow 1$ and $9 \rightarrow 8 \rightarrow 4 \rightarrow 2$. Therefore, to avoid this problem, we extract sequences in a totally random manner: every sequence to be extracted starts with a randomly-picked node, randomly selects one of its children for the next node, and repeats until a randomly-generated sequence length between 2 and 10 is reached.

The following rules are considered for the page sequence extraction:

[4]http://developer.yahoo.com/search/siteexplorer/V1/inlinkData.html

- Only extract pages from higher layers to lower layers or from the same layer. For instance, in Fig. 5(e), $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is valid, while $9 \rightarrow 8 \rightarrow 4 \rightarrow 7 \rightarrow 3 \rightarrow 1 \rightarrow 0$ is not valid in this case, because $4 \rightarrow 7$ is the reverse order in the sequence. The advantage of this option is to try to collect all positive sequences as training data; in other words, these sequences are good examples for learning.

- Avoid loops in the sequence. For example, $9 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is valid, and $9 \rightarrow 7 \rightarrow 4 \rightarrow 7 \rightarrow 4 \rightarrow 1$ is not valid.

The length of the sequences is randomly generated between 2 and 10. As a result, page sequences with length 2 are also included. For instance, $9 \rightarrow 8$, $4 \rightarrow 7$, $1 \rightarrow 0$. The training with short parent-child pairs aims to focus on local features.

The hidden state for a page in our system is its lowest layer number. For example, given graph Fig. 5(e), sequence $9 \rightarrow 7 \rightarrow 4 \rightarrow 1 \rightarrow 0$ is a valid sequence and its corresponding state sequence is $T_1 \rightarrow T_0 \rightarrow T_1 \rightarrow T_1 \rightarrow T_0$. The rule is that a target node is always set to layer number 0, no matter which physical layer it is in the graph, and other nodes will be re-assigned a new layer number if the layer numbers of its adjacent nodes are changed. For example, the original state sequence is $T_4 \rightarrow T_3 \rightarrow T_2 \rightarrow T_2 \rightarrow T_1 \rightarrow T_0$ for the page sequence $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$, according to the graph of Fig. 5(d), since the original physical layer number of node 7 is layer 3. However, when nodes 7 and 5 are marked as target pages in Fig. 5(e), their layer number will be changed to 0; accordingly, the layer number of other nodes in the graph will also be changed. For example, the new layer number of node 8 will be changed from 3 to 1, since the new layer number of its direct child node 5 is now 0. That is, according to our definition of hidden state T_i of a page, which represents the smallest number of hops to reach a target from this page, the hidden states of all nodes from node 1 to node 9 are $\{T_0, T_1, T_1, T_2, T_1, T_0, T_2, T_0, T_1, T_1\}$ in Fig. 5(e). Therefore, the new state sequence for the page sequence $9 \rightarrow 7 \rightarrow 3 \rightarrow 4 \rightarrow 2 \rightarrow 0$ is $T_1 \rightarrow T_0 \rightarrow T_2 \rightarrow T_1 \rightarrow T_1 \rightarrow T_0$.

The complete training data includes page sequences and their corresponding layer sequences. For MEMM and CRF crawls, page sequences are referred to as observation sequences or observable input sequences, and layer sequences are referred to as hidden state sequences or state (label) sequences.

3.2 Pattern Learning

The second stage of the system architecture is *Pattern Learning*. As shown in Fig. 3, the objective of this component is to take the training data (page sequences) as input, extract features and estimate the parameters for different underlying learning models (MEMMs, CRFs).

MEMMs and CRFs allow the independence assumptions on the observations inherent in generative models to be relaxed. As introduced in Section 2.1, both MEMMs and CRFs are conditional probabilistic sequence models used to estimate conditional probability, $p(s|o)$, of state sequences $s = s_1, s_2, ..., s_n$ given observation sequences

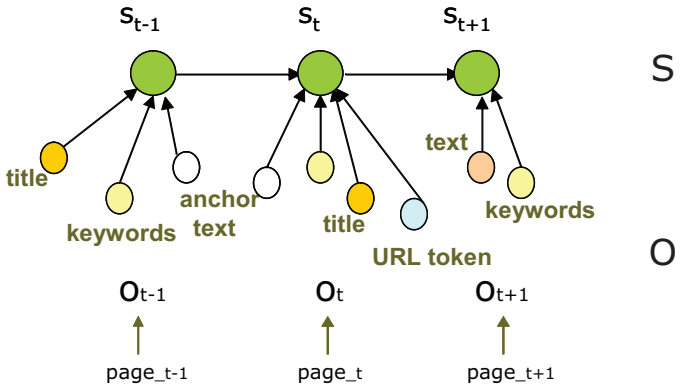


Figure 6: Dependency structure of MEMMs on modeling a sequence of Web pages. Directed graphical model, arrow shows dependency (cause). It defines *separate* conditional probabilities $p(s_t|s_{t-1}, o_t)$ at each position t based on features such as title, keywords, and URL token. t ranges over input positions.

$o = o_1, o_2, ..., o_n$. Both of them use a linear combination of weighted feature functions $\sum_i \lambda_i f_i(s, o)$ to encode the information encapsulated in the training data, which allows dependent, interacting, and arbitrary features of the observation sequence. Web pages are richly represented using extracted multiple features such as title, keywords, and URL token. This flexibility provides us the power to better model many real-world problems and gives better performance on a number of real-world sequence tasks. The parameters of MEMMs and CRFs are the feature weights $\lambda = \{\lambda_i\}$. However, they are different models: MEMMs are directed graphical models, while CRFs are undirected graphical models. The graphical dependency structures of MEMMs and CRFs for modeling the focused crawling problem in Fig. 6 and Fig. 7 show us the differences. MEMMs calculate the conditional probability $p(s|o)$ by multiplying a chain of *local* conditional probabilities $p(s_t|s_{t-1}, o_t)$ based on the multiple features, such as anchor text, at each position t, whereas, CRFs define a *single* conditional probability $p(s|o)$ over the entire state sequence s, given the observation sequence o.

Although the underlying models are different, the elements of the dependency structure and the features used in MEMMs and CRFs are the same, as illustrated in Fig. 6 and Fig. 7.

therefore, we describe them together in the following sections.

Structure of MEMM/CRF for Focused Crawling. Let k be the number of hidden states. The key quantities associated with MEMM/CRF models are the hidden states, observations (features), and the parameters (feature weights vector λ).

- Hidden states: $S = \{T_{k-1}, T_{k-2}, ..., T_1, T_0\}$
  - The focused crawler is assigned to be in state T_i if the current page is i

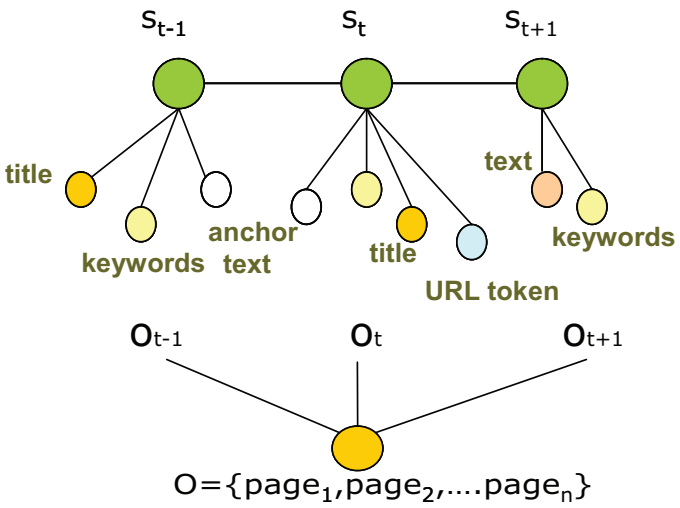


Figure 7: Graphical structure of CRFs on modeling a sequence of Web pages. This is an undirected graphical model. It defines a *single* conditional probability $p(s|o)$ over the entire state sequence s given the observation sequence o based on features such as title, keywords, and URL token. t ranges over input positions.

hops away from a target. The state T_{k-1} represent "$k-1$" or more hops to a target page.

- Observations: Collections of feature values of page sequences $O = \{page_1, page_2, page_3, ...\}$
  - Observable page sequences represented by a sequence of values for a set of predefined feature functions $f = \{f_1, f_2, ..., f_m\}$. m is the number of feature functions.
- Set of parameters $\lambda = \{\lambda_1, \lambda_2, ... , \lambda_m\}$
  - The parameter is a weight vector associated with each feature function $f = \{f_1, f_2, ..., f_m\}$.

Parameter estimation in MEMMs/CRFs uses Limited Memory Quasi-Newton Method (L-BFGS) [33, 43] to iteratively estimate the model parameters λ.

Features and Feature Functions. MEMMs and linear-chain CRFs make a first-order Markov independence assumption among states, that is, the current state depends only on the previous state and not on any earlier states. We use s to represent the entire state sequence, o for the entire observation sequence, and s_t and o_t to indicate the state and observation at the position t, respectively. Since MEMMs define separate conditional probabilities $p(s_t|s_{t-1}, o_t)$ based on features at each position

Table 1: Summary of Features

Feature Name	**Description**
Edge	states transition features
Text	cosine similarity between text of current page and target pages
Description	cosine similarity between the description of current page and target pages)
Words	Words themselves (meta data, title, alt, head, important words)
URL Token	Contain at least one of target keywords
Pointing Anchor	Anchor surrounding text in parent page which points to the page
Child Anchor	Anchor surrounding text in the page

t, whereas CRFs define a single conditional probability $p(s|o)$ over the entire state sequence s given the observation sequence o. We can rewrite $f(s, o)$ at position t specifically into $f(s_{t-1}, s_t, o_t)$ in MEMMs, and $f(s_{t-1}, s_t, o, t)$ in CRFs. For simplicity, we use $f(s, o, t)$ in this section. Note that $f(s, o, t)$ in this work only depends on s_{t-1} and s_t and the content of pages observed at time t.

Each feature function $f(s, o)$ is defined as a factored representation. Formally, we assume the feature functions to be factorized as:

$$f(s, o, t) = L(s_{t-1}, s_t, t) * O(o, t) \tag{5}$$

where $L(s_{t-1}, s_t, t)$ are transition feature functions, and $O(o, t)$ are observation feature functions. This allows us to combine information around the current position t. With defined feature functions, we construct a set of real-valued features to capture whether the observed Web pages have specific characteristics. In MEMMs and CRFs, each feature can be represented as either a binary or a real value. For binary values, a 1 indicates the presence of the feature, whereas a 0 means the absence of the feature. Real-valued features often represent cosine similarity values between text segments.

Example. Suppose current page is the page at time t, and parent page is the page at time $t-1$. Let $o = page1, page2, page3$ be a sequence of Web pages, and $s = T_2, T_1, T_0$ be the corresponding state sequence. To represent the information of $page3$ being in state T_0 at current time $t = 3$ if the anchor text of the URL of $page3$ in its parent page $page2$ contains specified keywords "linux", the feature functions $f(s, o, t)$ can be written as:

$$f(s, o, t = 3) = \begin{cases} 1 & \text{if } L(s_{t-1}, s_t = T_0, t = 3) = 1 \text{ and } O(o, t = 3) = 1 \ ; \\ 0 & \text{otherwise.} \end{cases}$$

For each i for which a transition $T_i \rightarrow T_0$ is possible,

$$L(s_{t-1}, s_t = T_0, t = 3) = \begin{cases} 1 & \text{if } t = 3 \text{ and } s_{t-1} = T_i \text{ and } s_t = T_0 \text{ exist;} \\ 0 & \text{otherwise.} \end{cases}$$

$$O(o, t = 3) = \begin{cases} 1 & \text{if anchor text of } o_t = page3 \text{ contains "linux"} \\ 0 & \text{otherwise.} \end{cases}$$

In other words, $L(s_{t-1}, s_t, t)$ captures the possible transition features of the states at time $t-1$ and current states (in this case, $T_1 \rightarrow T_0$), and $O(o, t)$ expresses the some observation features at current time t. □

We collect three kinds of data for generating the features: keywords, descriptions and target pages. Keywords are formed by concatenating the words appearing in the different levels along the topical hierarchy directory from the top. Descriptions are generated using the descriptive text and the anchor text in the page of the topic in the ODP. These hyperlinks and their descriptions are created by expert editors, which summarize the contents of the pages linked and are independent of the page contents. Target pages are the external Web pages the hyperlinks point to. We also further extract important words embedded in the target pages themselves, including words from title and headers (<title>...</title>, <h1>...</h1> etc.) and keywords and descriptions from meta data (<meta>...</meta>).

Table 1 summarizes all features we defined. We now describe each of them in detail.

1. **Edge Features:** Edge feature functions $L(s_{t-1}, s_t, t)$ can have two forms: L_1 and L_2. We use Edge feature $L_1(s_{t-1}, s_t, t)$ to capture the possible transitions from states s_{t-1} to s_t, and $L_2(s_{t-1}, s_t, t)$ to capture the possible states at time t.

   Formally, for all $i, j = 0, 1, ..., k-1$ so that specified $T_i, T_j \in S = \{T_{k-1}, T_{k-2}, ..., T_1, T_0\}$, we can have feature functions of the following form:

$$L_1^{(i,j)}(s_{t-1}, s_t, t) = \begin{cases} 1 & \text{if } s_{t-1} = T_i \text{ and } s_t = T_j \text{ is an allowed transition;} \\ 0 & \text{otherwise.} \end{cases}$$

$$L_2^{(i)}(s_{t-1}, s_t, t) = \begin{cases} 1 & \text{if } s_t = T_i \text{ exists;} \\ 0 & \text{otherwise.} \end{cases}$$

   Some states and state transitions may not happen at some time positions, for example, it is impossible to have state transition $T_2 \rightarrow T_0$. If state transition $T_3 \rightarrow T_2$ is possible at time t, then the feature value of $L_1(T_3, T_2, t)$ is 1.

2. **Text Feature:** It captures the maximal cosine similarity value between the content of a given candidate page and the set of targets. We define it as $O_1(o, t)$:

   $O_1(o, t) = \max_{d \in T} \cos(\text{text of current page } p \text{ at time } t, \text{ text of target page } d)$;

where T is the set of target pages, and cos(.,.) is the standard cosine similarity function between two term vectors, i.e.,

$$\cos(p, d) = \frac{\sum_{k \in p \cap d} w_{pk} \cdot w_{dk}}{\sqrt{\sum_{k \in p} w_{pk}^2 \sum_{k \in d} w_{dk}^2}}$$

where w_{dk} is the term frequency of term k in document d.

3. **Description Feature:** This is the cosine similarity value between the page description of a given candidate page and the target description. Normally every page has meta data in addition to the text content of the page, including description and keywords, which summarizes the whole page. We define feature function $O_2(o, t)$ to capture the description feature:

$$O_2(o, t) = \text{ cos(description of current page at time } t\text{, the target description);}$$

4. **Word Feature:** Word feature $O_w(o, t)$ captures the important words (meta data, title, head) appearing in the page text.

$$O_w(o, t) = \begin{cases} 1 & \text{if word } w \text{ appears in the current page at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

We may also use the counts of word w as the value of this feature, instead of the binary value.

5. **URL Token Feature:** The tokens in the URL of an observed page may contain valuable information about predicting whether a page is a target page or potentially leads to a target. For example, a URL containing "linux" is more likely to be a Web page about linux-related information, and a URL which contains the word "operating system" or "OS" indicates that with high probability, it may lead to a Linux page. There are two possible kinds of URLs related to the current observed page: one is the URL of the current page itself, and another one is the URL the current page is pointing to. We define two token feature functions $O_3(o, t)$ and $O_4(o, t)$ to identify if the keywords appear in the URLs.

$$O_3(o, t) = \begin{cases} 1 & \text{if any of URLs in the current page at time } t \\ & \text{contains at least one target keyword;} \\ 0 & \text{otherwise.} \end{cases}$$

$$O_4(o, t) = \begin{cases} 1 & \text{if the URL of the current page at time } t \\ & \text{(contained in the parent page) contains at least} \\ & \text{one target keyword;} \\ 0 & \text{otherwise.} \end{cases}$$

6. **Anchor Text Feature:** The anchor text around a link pointing to an observed page o is often closely related to the topic of the page. A human's ability to discriminate between links mostly relies on the anchor text. We capture the

important word w in the anchor text by defining two anchor features: $O_{5,w}(o,t)$ is used to identify the anchor text in the parent page surrounding the link which points to the given page, and $O_{6,w}(o,t)$ is to identify anchor text in the given page.

$$O_{5,w}(o,t) = \begin{cases} 1 & \text{if word } w \text{ appears in the anchor text of the link in the} \\ & \text{the parent page linking to current page at time } t; \\ 0 & \text{otherwise.} \end{cases}$$

$$O_{6,w}(o,t) = \begin{cases} 1 & \text{if word } w \text{ appears in the anchor text in the current} \\ & \text{page at time } t \text{ pointing to the page at time } t+1; \\ 0 & \text{otherwise.} \end{cases}$$

To represent each feature at each time step, we use a feature vector F_t to include all the arguments for each feature function, which are needed for computing feature values at each time t. For example, as shown in $O_{5,w}(o,t)$, if word w appearing in the anchor text in the previous page is used as a feature, then the feature vector F_t is assumed to include the identity of word w.

Suppose we have m feature functions (Eq. 5). The parameter estimation procedure for MEMMs and CRFs involves learning the parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$. Each λ_i is the weight associated with feature f_i, indicating the informativeness of feature f_i. For each feature f_i, it is an optimization problem to find the best λ_i so that the expected "count" of feature f_i using current value equals to the empirical "count" of feature f_i in the training data. Details are described in Sections 4 and 5.

3.3 Focused Crawling

After the learning phase, the system is ready to start focused crawling. An overview of the crawling algorithm is shown in Fig. 8. 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 seconds for Web downloads and filter out all pages except those with text/html content. The crawling respects the Robot Exclusion Protocol and distributes the load over remote Web servers. The crawler downloads the page pointed to by the URL at the head of the queue and extracts all the outlinks and performs feature selections. The predicted state for each child URL is calculated based on the current observable features and corresponding weight parameters, and the visit priority values are assigned accordingly. In the flow chart, the boxes *Analyze Page* and *Prediction* are implemented differently for different underlying models (MEMMs, CRFs).

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 whether a Web page is relevant to the user's interests or target topics during crawling. If its maximal cosine similarity to the target set is greater than γ, the URL is considered relevant.

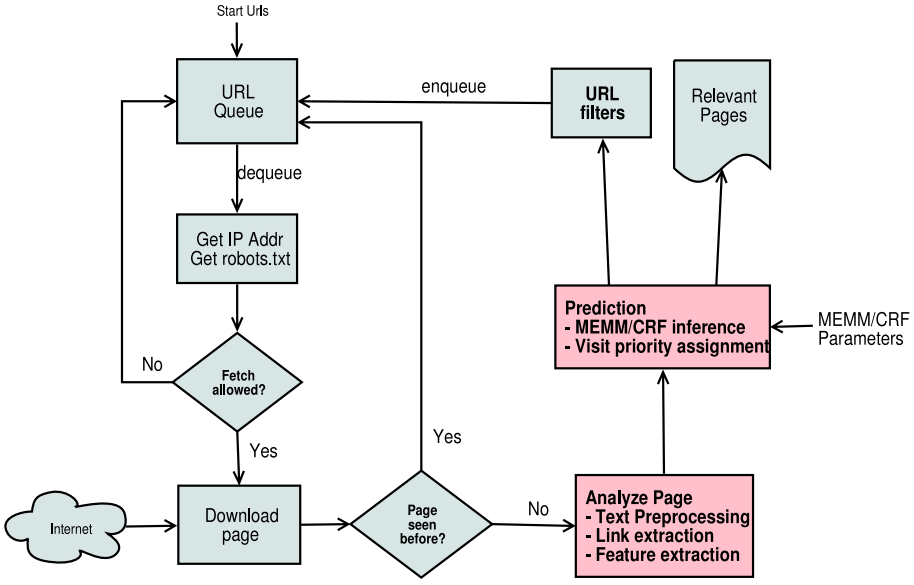


Figure 8: Flow Chart of Focused Crawling with MEMM/CRF Models

3.3.1 Efficient Inference

The task during the crawling phase is to associate a priority value with each URL that is being added to the queue. The goal of inference is to infer the probability of s_t given $o_{1..t}$ based on the parameters of the underlying models. Given a downloaded Web page w_t at current time t, the URLs associated with its outgoing links w_{t+1} are inserted into the visit queue, sorted by the assigned priority value based on the probabilities of the predicted state of the children of w_t, $p(s_t|o_{1..t})$.

The estimated distribution of the state of w_t, $p(s_t|o_{1..t})$, also called belief state, is a probability distribution over all possible state values $s_t = T_0, T_1, T_2, .., T_{k-1}$, given observations $o_{1..t}$. The distribution at time t is computed recursively based on the distribution at time $t-1$, corresponding to the parent page w_{t-1} of w_t. Parent page w_{t-1} of w_t is the Web page containing the URL of w_t, which led to the insertion of w_t in the queue. The sequence $1..t$ 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.

The probability distribution $p(s_t|o_{1..t})$ of the state at the next step t, given observations $o_{1..t}$, can be calculated recursively from the result up to time $t-1$ by conditioning on the previous state s_{t-1}:

$$p(s_t|o_{1..t}) = \sum_{s_{t-1}} p(s_t|s_{t-1}, o_t)\ p(s_{t-1}|o_{1..t-1}) \tag{6}$$

where $p(s_t|s_{t-1}, o_t)$ is the conditional probability of the transition from state s_{t-1} to

state s_t on observation o_t. The probability can be calculated efficiently by dynamic programming. We define forward vector α_t with each visited page. Each forward value $\alpha(s_t, t)$, is the probability that the system is in state s_t at time t given all observations made up to time t. Hence the values $\alpha(s_t, t)$ are the calculated values of $p(s_t|o_{1..t})$. MEMMs and CRFs define forward value $\alpha(s_t, t)$ differently. The details are explained in Sections 4 and 5.

In MEMMs, $p(s_t|s_{t-1}, o_t)$ can be directly obtained by the definition

$$\begin{aligned} p(s_t|s_{t-1}, o_t) &= \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)) \\ z(o_t) &= \sum_{s' \in S} \exp \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o_t) \end{aligned}$$

Therefore, Eq. 6 is calculated in MEMMs by

$$p(s_t|o_{1..t}) = \sum_{s_{t-1}} \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t))\ p(s_{t-1}|o_{1..t-1}) \tag{7}$$

In CRFs, $p(s_t|s_{t-1}, o_t)$ is calculated through the forward value $\alpha(s_t, t)$,

$$\alpha(s_t, t) = \sum_{s_{t-1}} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t))\ \alpha(s_{t-1}, t-1) \tag{8}$$

then $p(s_t|o_{1..t})$ is obtained by

$$p(s_t|o_{1..t}) = \frac{\alpha(s_t, t)}{\sum_{s_i} \alpha(s_i, t)} \tag{9}$$

where, s_i ranges all possible states $s_i = T_0, T_1, T_2, .., T_{k-1}$.

Children of page w_t will have different visit priorities because additional features such as Anchor Text Feature and URL Token Feature of w_{t+1} extracted from page w_t are included in the feature vector F_t at time t, described in Section 3.2.

3.3.2 Calculation of the Priority

The visit priority is determined based on the estimated state distribution $p(s_t|o_{1..t})$ for each URL, defining a vector key ($p(T_0)$, $p(T_1)$, $p(T_2)$, ..., $p(T_{k-1})$), where k is the number of hidden states. The sorting keys are arranged according to the estimated probabilities of the k possible states in the order of (T_0, T_1, T_2,..., T_{k-1}). URLs are sorted in a lexicographic manner based on the sequence of probabilities, beginning with $p(T_0)$.

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. More clearly, given two state distributions $X[p(T_0), p(T_1), p(T_2),$

..., $p(T_k)$] and $Y[p(T_0), p(T_1), p(T_2), ..., p(T_k)]$, 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 $Y[p(T_1)]$. A non-zero threshold is needed, because the probabilities are real numbers, and therefore lexicographic ordering of the vectors would not be possible.

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 consists of the following three basic elements: the URL of page w_t; the visit *priority* of w_t; probabilities $\alpha(T_j, t-1)$ that page w_{t-1} is in hidden state T_j, for all j, capturing $p(s_{t-1}|o_{1..t-1})$. Other elements needed for inference may vary depending on different underlying models.

With MEMMs/CRFs, we keep the anchor text feature and URL Token feature values of the parent page w_{t-1} of page w_t. Detailed crawling algorithms with each model are described in the following sections.

3.4 Evaluation Methods

It is important that the focused crawler returns as many relevant pages as possible while minimizing the irrelevant ones. The standard information retrieval (IR) measures used to evaluate the performance of a crawler are *Precision* and *Recall*.

The *precision* is the percentage of the Web pages crawled that are relevant to the topic. It is ideal to have real users identify the relevant pages. However, involving real users for judging thousands of pages is not practical. The use of cosine similarity to judge relevance of pages has been adopted in several topical crawling studies [19, 32, 48]. Since in general there are multiple target pages in our system either marked by the user or extracted from DMOZ, the relevance assessment of a page p is based on maximal cosine similarity to the set of target pages T with a confidence threshold γ. That is, if $\max_{d \in T} \cos(p, d) \geq \gamma$ 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. Instead, we use the *Maximum Average Similarity* as the second evaluation metric.

The ability of the crawler to remain focused on the topical Web pages during crawling can be measured by the average relevance of the downloaded documents [32, 31]. Average Similarity is the accumulated similarity over the number of the crawled pages. This relevance measure also has been used in [32, 31, 47, 14]. In our system, since there are multiple user-marked or pre-specified target pages, if the query document is close to any one of the targets, it is considered as matching the user's interests or the topic. 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.

$$\sigma = \max_{d \in T} \frac{\sum_{p \in S} \cos(p, d)}{|S|} \tag{10}$$

where T is the set of target pages, S is the set of pages crawled, $|S|$ is the number of targets, and

$$\cos(p, d) = \frac{\sum_{k \in p \cap d} w_{pk} \cdot w_{dk}}{\sqrt{\sum_{k \in p} w_{pk}^2 \sum_{k \in d} w_{dk}^2}}$$

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

4 Focused Crawling with Maximum Entropy Markov Models

In this section, we elaborate on the details of prediction of paths to relevant web pages by using Maximum Entropy Markov Models (MEMMs). This will allow us to explore multiple overlapping features for training and crawling. All features and feature functions have been described in Section 3.2.

4.1 Maximum Entropy Markov Model

Maximum Entropy Markov Model or MEMM is an augmentation of the basic Maximum Entropy Model so that it can be applied to calculate the conditional probability for each element in a sequence [39, 27]. It is a probabilistic sequence model that defines conditional probabilities of state sequences given observation sequences.

Formally, let s and o be random variables ranging over observation sequences and their corresponding state (label) sequences respectively. We use $s = s_1, s_2, ..., s_n$ and $o = o_1, o_2, ..., o_n$ for the generic state sequence and observation sequence respectively, where s and o has the same length n. As shown in Fig. 2(a), state s_t depends on observations o_t and previous state s_{t-1}.

Therefore, MEMMs are discriminative models that define the conditional probability of a state sequence s given an observation sequence o, $p(s|o)$.

Let n be the length of the input sequence, m be the number of features. Then $p(s|o)$ is written as

$$p(s|o) = p(s_1, s_2, ..., s_n|o_1, o_2, ..., o_n) \tag{11}$$

MEMMs make a first-order Markov independence assumption among states, that is, the current state depends only on the previous state and not on any earlier states, so $p(s|o)$ can be rewritten as:

$$p(s|o) = \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t) \tag{12}$$

where t ranges over input positions $1..n$. Applying the maximum entropy principle, $p(s_t|s_{t-1}, o_t)$ can be rewritten as follows:

$$p(s_t|s_{t-1}, o_t) = \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o_t)) \tag{13}$$

$$z(o_t) = \sum_{s' \in S} \exp \sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o_t) \tag{14}$$

where, s_t is the state at time t, o_t is the observation at time t, S represents a set of possible states corresponding to finite state machines(FSMs). Combining Eq. 12

with Eq. 13, we get

$$\begin{aligned} p(s|o) &= \prod_{t=1}^{n} p(s_t|s_{t-1}, o_t) \\ &= \frac{\exp(\sum_{t=1}^{n}\sum_{i=1}^{m}\lambda_i f_i(s_{t-1}, s_t, o_t))}{\prod_{t=1}^{n} z(o_t)} \end{aligned} \tag{15}$$

$z(o_t)$ is calculated using Eq. 14. As seen in Eq. 15, MEMM defines the conditional probability of state sequence s given observation sequence o as the product of separate conditional probabilities $p(s_t|s_{t-1}, o_t)$ at each position t, ranging over all positions. $z(o_t)$ is called the per-state normalizing factor. In other words, MEMM uses local normalization factor $z(o_t)$ to form a chain of local models, which implies each position t contains a "next-state classifier" that makes the distribution sum to 1 across all next state s_t at current position t.

4.2 Parameter Estimation

The task is to estimate the parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$, which are the weights for each feature function $f_1, f_2, ..., f_m$, based on the training data. We start with the formulation of the objective function, its smoothing via a penalty term, and the training algorithm based on the Limited-Memory Quasi-Newton method L-BFGS [33]. We adopted L-BFGS as it has been shown to perform better than Iterative Scaling and Conjugate Gradient methods in sequential tasks of a similar nature as ours [25, 43].

The training data D consists of N state-observation sequences, $D = \{\mathbf{S}, \mathbf{O}\} = \{(s^j, o^j)\}_{j=1}^{N}$ (we use superscripts j to represent training instances), where each $o^j = \{o_1^j, o_1^j, ..., o_n^j\}$ is a sequence of observations of length n, and each $s^j = \{s_1^j, s_1^j, ..., s_n^j\}$ is the corresponding sequence of states. The task of training is to choose values of parameters $\{\lambda_i\}$ which maximize the log-likelihood, $L_\lambda = \log p(\mathbf{S}|\mathbf{O})$, of the training data. We use (s^j, o^j) to represent the j^{th} state-observation sequence from the training data set, s_t^j, o_t^j to indicate the state and observation at position t of the j^{th} state-observation sequence respectively.

The goal of smoothing is to penalize large weights, since Maximum Entropy may converge to very large weights which overfit the training data. There are a number of smoothing methods for maximum entropy models [17, 15, 8, 9]. To avoid overfitting and sparsity, we use smoothing to penalize the likelihood with a Gaussian weight prior [8, 9] assuming that weights are distributed according to a Gaussian distribution with mean μ and variance σ^2. We can prevent overfitting by modifying the optimization objective function $\log p(\mathbf{S}|\mathbf{O})$ to maximum posterior likelihood $\log p(\mathbf{S}, \lambda|\mathbf{O})$:

$$\begin{aligned} L_\lambda = \log p(\mathbf{S}, \lambda|\mathbf{O}) &= \log p(\mathbf{S}|\mathbf{O}) + \log p(\lambda) \\ \textit{Posterior} \qquad & \textit{Evidence} \qquad \textit{Prior} \end{aligned} \tag{16}$$

Its derivative with respect to λ_i is:

$$\begin{aligned}\frac{\partial L_\lambda}{\partial \lambda_i} &= \sum_{j=1}^{N}\sum_{t=1}^{n} f_i(s_{t-1}^j, s_t^j, o_t^j) - \sum_{j=1}^{N}\sum_{t=1}^{n}\sum_{s'\in S} p(s'|s_{t-1}^j, o_t^j) f_i(s_{t-1}^j, s', o_t^j) - \sum_{i=1}^{m}\frac{\lambda_i}{\sigma^2} \\ &= \tilde{E}_i - E_i - \sum_{i=1}^{m}\frac{\lambda_i}{\sigma^2} \qquad (17)\end{aligned}$$

$\tilde{E}_i$ is the first (double) sum and gives the empirical value for feature f_i of the training data, equal to the sum of feature f_i values for position t in all N sequences. E_i is the second (triple) sum and gives the expected value for feature f_i using current parameters with respect to the model distribution p. Eq. 17 is the first-derivative with respect to λ_i of the objective function L_λ. Therefore, the optimum parameter λ_i is obtained by setting these derivatives to zero. We observe that, without the smoothing term (the last sum), the derivative of L_λ is $\tilde{E}_i - E_i$, and the optimum parameters λ are the ones for which the expected value of each feature equals its empirical value. In practice, the L-BFGS optimization method is applied, which requires the first derivative of the objective function, given by Eq. 17.

4.3 Focused Crawling

We now discuss two kinds of inference we use in the Focused Crawling stage. When the crawler sees a new page, the task of the inference is to estimate the probability that the page is in a given state s based on the values of all observed pages already visited before. We are using two major approaches to compute the probabilities in our experiments: marginal probability and the Viterbi algorithm, which can be performed efficiently using dynamic programming.

Marginal Mode. The marginal probability of states at each position t in the sequence is defined as the probability of states given the observation sequence up to position t. Specifically, the forward probability, $\alpha(s,t)$ is defined as the probability of being in state s at position t given the observation sequence up to position t. We set $\alpha(s,1)$ equal to the probability of starting with state s and then use the following recursion:

$$\alpha(s,t) = \sum_{s'} \alpha(s', t-1)\ p(s|s', o_t) \qquad (18)$$

The calculation of $p(s|s', o_t)$ is straightforward in MEMMs. As we discussed above, MEMM directly defines separate conditional probabilities $p(s|s', o_t)$ at each position t, that is,

$$\begin{aligned}p(s|s', o_t) &= \frac{1}{z(o_t)}\exp(\sum_{i=1}^{m}\lambda_i f_i(s', s, o_t)) \\ z(o_t) &= \sum_{s"\in S}\exp\sum_{i=1}^{m}\lambda_i f_i(s', s", o_t)\end{aligned}$$

Substituting into Eq. 18, the forward probability values $\alpha(s,t)$ are calculated.

In our notation, hidden states are defined as T_j, $j = 0..k-1$, therefore we associate values $\alpha(T_j,t)$ with each visited page. Forward value $\alpha(T_j,t)$ is the probability that the system is in state T_j at time t, based on all observations made thus far. Given the values $\alpha(T_j,t-1)$, $j = 0..k-1$, of the parent page, we can calculate the values $\alpha(T_j,t)$ using the following recursion, derived from Eq. 18:

$$\alpha(T_j,t) = \sum_{j'=0}^{k-1} p(T_j|T_{j'},o_t)\alpha(T_{j'},t-1) \tag{19}$$

Hence the values $\alpha(T_j,t)$ in our focused crawling system are calculated as:

$$\alpha(T_j,t) = \sum_{j'=0}^{k-1} \alpha(T_{j'},t-1)\ \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j,T_{j'},o_t)) \tag{20}$$

$$z(o_t) = \sum_{j"=0}^{k-1} \exp \sum_{i=1}^{m} \lambda_i f_i(T_{j'},T_{j"},o_t) \tag{21}$$

All the $\alpha(T_j,t)$ values for all $j = 0..k-1$ are stored in the priority queue, and used to determine the visit priority as described in Section 3.3.2, and to perform inference for the next time step.

The Viterbi Algorithm. In Viterbi decoding, the goal is to compute the most likely hidden state sequence given the data:

$$s^* = \arg\max_s p(s|o) \tag{22}$$

By Bellman's principle of optimality, the most likely path to reach state s_t consists of the most likely path to some state at time $t-1$ followed by a transition to s_t. Hence the Viterbi algorithm can be derived accordingly. $\delta(s,t)$ is defined as the best score (the highest probability) over all possible configurations of the state sequence ending at the position t in state s given the observation up to position t. That is

$$\delta(s,t) = \max_{s'} \delta(s',t-1)\ p(s|s',o_t) \tag{23}$$

This is the same as the forward values (Eq. 18), except we replace sum with max. Applying Eq. 23 to our problem, with hidden states defined as T_j, $j = 0..k-1$, we have:

$$\delta(T_j,t) = \max_{j'=0}^{k-1} \delta(T_{j'},t-1)\ \frac{1}{z(o_t)} \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j,T_{j'},o_t)) \tag{24}$$

$$z(o_t) = \sum_{j"=0}^{k-1} \exp \sum_{i=1}^{m} \lambda_i f_i(T_{j'},T_{j"},o_t) \tag{25}$$

Priority Queue Data Structure. To perform the dynamic programming for efficient inference during crawling, since MEMM crawling uses some features from parent page, we need to keep them in the queue. When a new page is seen, all features including features extracted from current page and those from parent page are collected together for prediction. Therefore, anchor text feature and URL token feature in the previous page should be kept in addition to the three basic elements. In summary, the following are four elements of the priority queue in MEMMs crawling:

- the URL of page w_t
- anchor text feature and URL Token feature vector P_t extracted from the parent page w_{t-1} of page w_t
- the visit *priority* of w_t
- probabilities $\alpha(T_j, t-1)$ that page w_{t-1} is in hidden state T_j, for all j, capturing $p(s_{t-1}|o_{1..t-1})$.

The flow chart of focused crawling with MEMMs is shown in Fig. 8 in Section 3. The pseudocode of crawling algorithm with MEMMs is shown in Fig. 9. Each feature vector F_t is different for each URL, therefore, the children of page w_t have different visit priorities.

We now analyze the computational complexity of the crawling algorithm with MEMM. For training, MEMM training requires $O(I*N*S^2*F)$ time, where I is the number of iterations, N is the size of training set, S is the total possible states, and F is the number of observation features that capture the relationship among state and observation sequences (see the outline of algorithm in [27]). The complexity of MEMM crawling algorithm is $O(L+Q\log Q+S^2*F)$, where L is the average number of outlinks per page, Q is the size of URL queue, S is the total possible states, and F is the number of observation features. For each Web page, all outlinks are extracted and saved in the URL queue. We use a hash table to implement the operations of inserting and removing URLs from the queue and checking if a newly extracted URL is already in the queue. The queue is sorted according to the visit priority value associated with each URL with merge sort, yielding $O(L+Q\log Q)$ complexity. The MEMM inference with the Viterbi algorithm uses dynamic programming for a running time complexity of $O(S^2*F)$. Therefore, MEMM crawling takes $O(L+Q\log Q+S^2*F)$ in the general case.

Algorithm Focused_Crawler(MEMM, classifier(d), k)
    $urlQueue$:= {Seed URLs};
    WHILE(not(termination)) DO
        w_t := dequeue head of $urlQueue$;
        extract from w_t its URL, possible anchor text
            and URL Token feature vector of parent page P_{t-1},
            and $\alpha(T_j, t-1)$ for all possible states j;
        Download contents of w_t;
        Parse and preprocess content, feature extraction vector F_t;
        IF classifier(w_t)=1, THEN store w_t as relevant;
        FOR EACH outlink w_{t+1} of w_t with url
            Calculate $\alpha(T_j, t)$ for all j and $priority$ for w_t's children:
                $\alpha(T_j, t) = \sum_i \alpha(T_i, t-1) p(T_j|T_i, o_t)$
                $\alpha(T_j, t) = \max_i \alpha(T_i, t-1) p(T_j|T_i, o_t)$
                Assign the visit $priority$ based on α;
            Update P_t;
            $urlQueueEntry$:= ($priority$, url, P_t, α);
        Enqueue ($urlQueueEntry$); // entries sorted by $priority$

Figure 9: Pseudocode of Crawling Algorithm with MEMMs

5 Focused Crawling with Conditional Random Fields

A conditional random field (CRF) [21, 51] is a form of undirected graphical model or Markov Random Field. It assumes a single log-linear distribution $p(s|o)$ over the entire state sequence $s = s_1, s_2, ..., s_n$ given the observation sequence $o = o_1, o_2, ..., o_n$, rather than defining per-state distributions over the next states given the current state at each position, as in MEMMs. Our treatment is adapted from the formulation and algorithms of Probabilistic Models for labeling Sequence Data [21], which are special cases of Dynamic Conditional Random Fields (DCRF) as developed in [49].

5.1 Conditional Random Fields (CRFs)

When modeling sequences, the simplest and most common linear-chain structure (Fig. 2(b)) is used, one where all the nodes in the graph form a linear chain. Such models have been used extensively in POS tagging, information extraction, document summarization and shallow parsing [21, 38, 37, 44, 43]. In these models, the set of cliques C is just the set of all cliques of size 1 (the nodes) and the set of all cliques of size 2 (the edges). When we model the focused crawling problem with hyperlinked Web pages, the state (label) of a page is assumed to be dependent upon the page itself and the states of the pages that link into it or out of it, which corresponds to an undirected graphical model in the shape of a linear chain.

$$p(s|o) = \frac{1}{Z(o)} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s_t, o)\right) \tag{26}$$

where $Z(o)$ is a normalization on o:

$$Z(o) = \sum_{s' \in |S|^n} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(s_{t-1}, s', o)\right) \tag{27}$$

The space of s' is now the space of state sequences, thus the number of all possible state sequences is exponential in the length of the input sequence n, $|S|^n$.

We observe a difference compared with MEMMs: the capitalized $Z(o)$ represents a global normalization factor on the whole sequence o, while $z(o_t)$ in MEMMs represents a local normalization factor at position t of the sequence o.

As mentioned above, in the case of linear-chain CRFs, each feature function will operate only on pairs of state variables s_{t-1} and s_t for each state transition, therefore, we can simply define $f_i(s_{t-1}, s_t, o)$ as $f_i(s, o, t)$, and rewrite the equation above as:

$$p(s|o) = \frac{1}{Z(o)} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(s, o, t)\right) \tag{28}$$

$$Z(o) = \sum_{s' \in |S|^n} \exp\left(\sum_{t=1}^{n}\sum_{i=1}^{m} \lambda_i f_i(s', o, t)\right) \tag{29}$$

5.2 Parameter Estimation in CRFs

Assuming we have training data D with N sequence pairs $\{(s^j, o^j)\}_{j=1}^N$, training consists of estimating the λ_i to maximize the conditional log-likelihood, L_λ, of all sequences in the training data set. We use (s^j, o^j) to represent the j^{th} page sequence pair from the training data set, then

$$L_\lambda = \sum_{j=1}^{N} \log p(s^j|o^j) = \sum_{j=1}^{N} \left[\sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s^j, o^j, t) - \log Z(o^j) \right] \tag{30}$$

where

$$Z(o^j) = \sum_{s' \in |S|^n} \exp\left(\sum_{t=1}^{n} \sum_{i=1}^{m} \lambda_i f_i(s', o^j, t) \right) \tag{31}$$

To perform this optimization as required by the L-BFGS method, we differentiate the log-likelihood with respect to parameters λ:

$$\frac{\partial L_\lambda}{\partial \lambda_i} = \sum_{j=1}^{N} \left[\sum_{t=1}^{n} f_i(s^j, o^j, t) - \sum_{s' \in |S|^n} p(s'|o^j) \sum_{t=1}^{n} f_i(s', o^j, t) \right] \tag{32}$$

The differences with a similar expression for MEMMs are the following. The first part inside the parenthesis of the right-hand side of Eq. 32 is the total sum of the values of feature function i for all positions given sequences o^j and s^j, and it can be calculated easily from the training data set. The second part is to compute the sum of the expectation value of each feature function with respect to the current CRF parameter for every observation sequence o^j in the training data. There are $|S|^n$ possible state sequences, so the computation is prohibitively expensive and must be summed over using dynamic programming.

Since the conditional distribution obeys the Markov property, the second part can be written as:

$$\sum_{s' \in |S|^n} p(s'|o^j) \sum_{t=1}^{n} f_i(s', o^j, t) = \sum_{t=1}^{n} \sum_{v,w} p(s_{t-1} = v, s_t = w|o^j) f_i(v, w, o^j, t) \tag{33}$$

This eliminates the need to sum over $|S|^n$ possible state sequences, and breaks the calculation into pairs.

The second part of Eq. 32 can be computed efficiently using a dynamic programming method. To do so, first we augment the graph with two additional state nodes, s_0 and s_{n+1}, with corresponding observations *start* and *stop* respectively, to the top (left) and bottom (right) level of the Web graph. That is, $s = s_0, s_1, s_2, ..., s_n, s_{n+1}$ is the state sequence, $o = start, o_1, o_2, ..., o_n, stop$ is the corresponding observation sequence. For a given o, we define a set of $n+1$ transition matrices for position $t, (t = 1..n+1)$, and each matrix M_t is a $n+1$ by $n+1$ matrix with elements of the form

$$M_t[s', s] = \exp(\sum_{i=1}^{m} \lambda_i f_i(s', s, o, t))$$

Thus, the probability of s given observation sequence o can be written as the product of the appropriate elements of the $n+1$ transition matrices as following:

$$p(s|o) = \frac{1}{Z(o)} \prod_{t=1}^{n+1} M_t[s_{t-1}, s_t] \tag{34}$$

We define forward vector $\alpha_t = \alpha(s,t)$ as the unnormalized probability of being in state s ending at position t and backward vector $\beta_t = \beta(s,t)$ as the unnormalized probability of being in state s at position t given the observation sequence after position t respectively for each node in the graph as

$$\alpha_t = \begin{cases} \alpha_{t-1} M_t & 0 < t <= n+1 \\ 1 & t = 0. \end{cases}$$

$$\beta_t = \begin{cases} M_{t+1} \beta_{t+1} & 0 <= t <= n \\ 1 & t = n+1. \end{cases}$$

Thus, we get:

$$Z(o) = \ 1 \cdot \alpha_{n+1} = \textstyle\sum_s \alpha(s, n+1)$$

We also define a matrix for features as $f_t[s', s] = f_i(s', s, o, t)$ and the second part of Eq. 32 is calculated as

$$\sum_{t=1}^{n} \sum_{v,w} p(s_{t-1} = v, s_t = w|o) f_i(v, w, o, t) = \sum_{t=1}^{n} \frac{\alpha_t (f_t * M_t) \beta_t}{Z(o)} \tag{35}$$

We use smoothing to penalize the likelihood over parameters, thus the derivative in Eq. 32 is modified into:

$$\frac{\partial L_\lambda}{\partial \lambda_j} \ = \ \sum_{k=1}^{N} \left[\sum_{t=1}^{n} f_j(s^k, o^k, t) - \sum_{s' \in |S|^n} p(s'|o^k) \sum_{t=1}^{n} f_j(s', o^k, t) \right] - \sum_{i=1}^{m} \frac{\lambda_i}{\sigma^2} \tag{36}$$

Therefore the parameter estimation problem consists of finding the best model parameters $\lambda = \{\lambda_1, \lambda_2, ..., \lambda_m\}$ with the first-derivative of the objective function, Eq. 36. The L-BFGS method can be used for this problem [33, 43].

5.3 Focused Crawling

Following the discussion in the previous Section about efficient inference in MEMMs, the marginal mode and the Viterbi algorithm for CRFs can be derived similarly.

Marginal Mode Back to our system notations, the hidden states s are defined as $T_j \in \{T_0, T_1, ..., T_{k-1}\}$. The forward probability values $\alpha(T_j, t)$ denotes the probability of being in state T_j at time t given the observed page sequence up to time t. Thus, $\alpha(T_j, t)$ in CRFs is calculated as:

$$\alpha(T_j, t) = \sum_{j'=0}^{k-1} \alpha(T_{j'}, t-1) \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o)) \tag{37}$$

Unlike in MEMMs, forward values are not calculated as $p(s_t|o_{1..t})$, since CRFs use global normalization $Z(o)$ to determine the conditional probability. $\alpha(T_j, t)$ is unnormalized probability. Therefore, the probability of state $s_t = T_j$ given the observation sequence o so far, $p(T_j|o_{1..t})$ is

$$p(T_j|o_{1..t}) = \frac{\alpha(T_j, t)}{Z(o)} = \frac{\alpha(T_j, t)}{\sum_{j=0}^{k-1} \alpha(T_j, t)} \tag{38}$$

The global normalization factor $Z(o)$ is defined when the recursion terminates at time step t, that is, $Z(o) = \sum_{j=0}^{k-1} \alpha(T_j, t)$.

The Viterbi Algorithm Recall that the Viterbi algorithm is to find the best choice of hidden state assignments for a sequence given the model parameters. In CRFs, $\delta(T_j, t)$ maintains the unnormalized probability of the best labeling ending at time t with the state T_j.

$$\delta(T_j, t) = \max_{j'=0}^{k-1} \delta(T_{j'}, t-1) \exp(\sum_{i=1}^{m} \lambda_i f_i(T_j, T_{j'}, o)) \tag{39}$$

Similarly, the normalized probability of the best labeling is given by $\frac{\delta(T_j,t)}{Z(o)}$.

Algorithm and System Structure The system structure and algorithm for focused crawling with CRFs are the same as that in MEMMs, only with CRFs parameters and inference forward values. The flow chart of focused crawling with CRFs is in Fig. 8. The computational complexity of the crawling algorithm with CRF mirrors that of crawling with MEMM, as described in Section 4.3.

6 Evaluation of feature selection and inference methods

In this section, we discuss the experimental evaluation of the MEMM- and CRF-based focused crawlers. The crawl topics are chosen from the ODP categories, and the target pages are chosen based on the listed URLs under each category. The 10 topics, followed by a quantitative description of the data, are shown in Table 2.

We use two measures in the evaluation of the results. The first one is the number of pages crawled that are relevant. The second one is *Maximum Average Similarity* σ as discussed in 3.4.

/Computers/Software/Operating_Systems/Linux/
/Home/Gardening/Gardens/Wildlife/Butterfly/
/Arts/Performing_Arts/Dance/Ballet/
/Sports/Cycling/Mountain_Biking/Downhill/
/Business/Management/Management_Information_Systems/Call_For_Papers/
/Society/Religion_and_Spirituality/Yoga/
/Health/Nutrition/Disease_Prevention/Heart_Disease/
/Sports/Hockey/Ice_Hockey/Training/
/Science/Astronomy/Amateur/Sky_Maps_and_Atlases/
/Society/Law/Legal_Information/Computer_and_Technology_Law/Internet/

Topic	**# of Target Pages**	**# of training sequences**	**Start Urls**
Linux	19	11394	6
Biking	17	9790	2
Butterfly	17	12172	2
Hearthealthy	8	12928	2
Hockey	19	4368	2
Fitnessyoga	7	11680	3
Balletdance	7	12317	3
Skymaps	16	12073	3
Callforpapers	11	9632	3
Internetlaw	18	12210	4

Table 2: The 10 topics from the ODP used for training crawlers, and quantities associated with the training data.

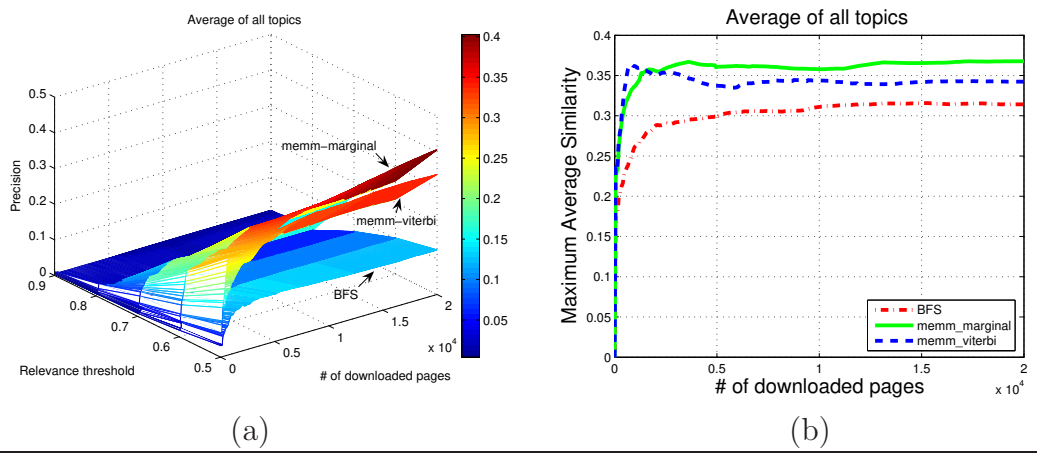


Figure 10: Performance of the MEMM-based crawler using marginal mode inference and Viterbi inference. (a) Precision as a function of number of downloaded pages and relevance threshold. The mesh has 400 points along the axis of number of downloaded pages and 5 points along the axis of relevance threshold; (b) Maximum Average Similarity as a function of the number of downloaded pages.

A baseline method Best-First Search (BFS) crawler is used for comparison. BFS 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.

We conducted two sets of experiments. The first set of experiments is to compare MEMM- and CRF-based methods with the two inference algorithms, Viterbi and Marginal Mode, against BFS crawl. The second set of experiments is to test the impact of multiple features on performance.

6.1 Inference in MEMM-based crawling

We now compare the results of MEMM-based crawler using marginal mode and the Viterbi algorithm for the inference. We use *BFS, memm-marginal, memm-viterbi* in the figures to denote Best-First Search crawler (BFS crawl), MEMM crawler with marginal mode inference (MEMM-marginal crawler), and MEMM crawler with Viterbi algorithm (MEMM-Viterbi crawler), respectively. We choose the cosine similarity threshold between 0.5-0.8 for general comparisons. Too high or too low a threshold may result in too few or too many relevant pages based on the target pages and the start URLs, which does not provide sufficient information for comparison.

First we compare our MEMM-based methods with all the features against BFS crawl. The results are shown in Fig. 10.

We observe that our MEMM-based crawl outperforms BFS crawl. We furthermore note that the marginal inference mode outperforms the Viterbi algorithm.

MEMM is a chain of a "next-state classifier" with the property that all the probability mass that arrives at a state must be distributed among the possible successor

states. At each position t, the marginal probability method involves finding the most likely state at current position given the observation sequence up to t, while the Viterbi algorithm involves finding the most likely path to reach state s_t based on the principle that the most likely path to reach state s_t consists of the most likely path to *some* state at time $t-1$ followed by a transition to s_t. In the focused crawling problem, finding the distribution for each individual hidden state at a particular instant is more important than finding the best "sequence" of hidden states of each Web page along the path, since there may be many very unlikely paths that led to large marginal probability. To be more specific, let us see an example. If we have

"aaa" 30% probability
"abb" 20% probability
"bab" 25% probability
"bbb" 25% probability

In this example, "aaa" is the most likely sequence, 'a' is the most likely first character, 'a' is the most likely second character, 'b' is the most likely third character, but the string "aab" has 0 probability.

This is a potential explanation why MEMM-marginal crawl demonstrates better performance than MEMM-Viterbi crawl on the average.

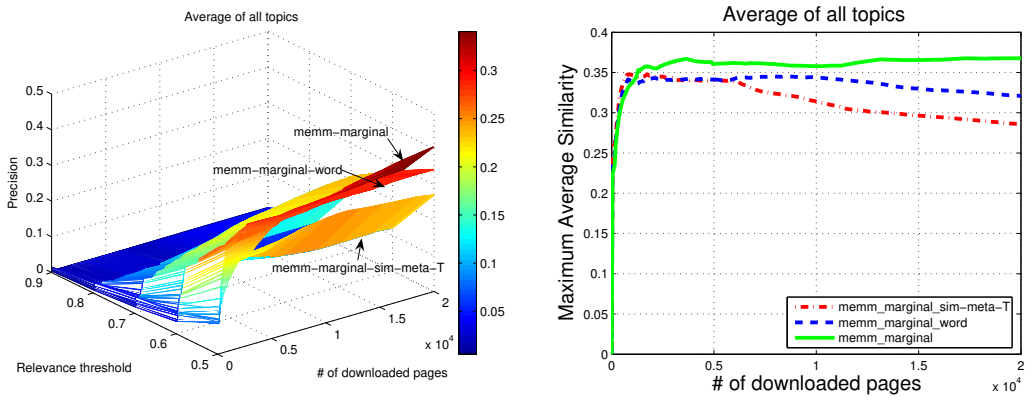
6.2 Comparison of MEMM-based crawlers with different features.

We next investigate the impact of feature selection on performance. In this experiment, we choose to compare MEMM-marginal crawl (which performs better than MEMM-viterbi) with different features: with all features, with Word feature only, and with the remaining features: Text feature, Description feature, URL token feature and Anchor text feature (features are described in Section 3.2). We denote them as *MEMM-marginal, MEMM-marginal-word, and MEMM-marginal-sim-meta-T* in the figures, respectively.

We observe in Fig. 11 that MEMM-marginal crawl with the combination of all features performs better on average than with Word feature only and with sim-meta-T features only. This confirms that, even if only some of the features are used, relevant paths can be effectively identified. Furthermore, we observe that the contribution of the non-word features is minimal, since the performance using all the features and the word features only is very similar.

6.3 Inference in CRF-based crawling

We next present the results of CRF-based crawler using marginal mode and the Viterbi algorithm for the inference. We use *BFS, crf-marginal, crf-viterbi* in the figures to denote Best-First Search crawler (BFS crawl), CRF crawl with marginal mode inference (CRF-marginal crawl), and CRF crawl with Viterbi algorithm (CRF-Viterbi crawl), respectively.



(a) (b)

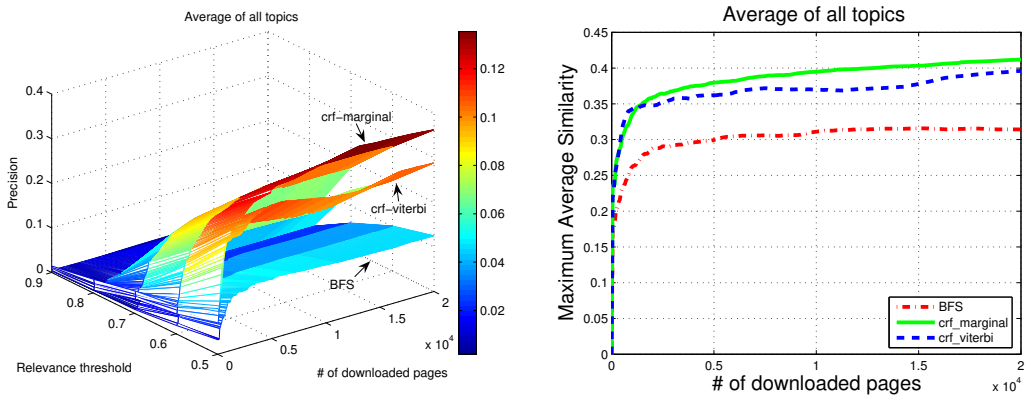
Figure 11: Performance of the MEMM-based crawler using marginal mode inference with different features. (a) Precision as a function of number of downloaded pages and relevance threshold. The mesh has 400 points along the axis of number of downloaded pages and 5 points along the axis of relevance threshold; (b) Maximum Average Similarity as a function of the number of downloaded pages.

First we compare our CRF-based methods with all the features against BFS crawl. The results are shown in Fig. 12. We observe that our CRF-based crawl outperforms BFS crawl. We furthermore note that the marginal inference mode outperforms the Viterbi algorithm.

One of the possible reasons for the better performance of CRF-marginal crawl and CRF-viterbi crawl over BFS crawl is that CRF-marginal and CRF-viterbi use more features for prediction. By incorporating more features, CRF-based methods can fully employ the rich content and context of the observed pages and produce better prediction in many situations, instead of depending only upon the relevance of the page as BFS crawl does. Furthermore, CRF-based methods are able to handle the inter-page relationship along the page sequence in a better way than BFS method which solves a simple classification problem on an individual page. Taking the interactions along the path into consideration is another potential reason why CRF-based methods outperform BFS crawl in most of these cases.

We compare CRF-marginal and CRF-viterbi inference to investigate whether finding the most likely hidden state sequence is preferred over finding the distribution for each individual hidden state. Several applications of CRFs have shown good performance when applying the Viterbi algorithm to problems in natural language processing [28, 41], feature induction for NER [26], and bioinformatics [42, 24]. These applications are trying to make predictions based on the most likely sequence labeling. Thus the Viterbi algorithm makes more sense because the single most likely sequence of hidden states could differ greatly from the sum of a number of possible sequences with a different hidden state, as calculated in marginal probability.

For our focused crawling problem, knowing what the probability of the hidden state a page is in at a particular time makes more sense, therefore finding marginal probability gives better prediction than using the Viterbi algorithm. The experimen-



(a) (b)

Figure 12: Performance of the CRF-based crawler using marginal mode inference and Viterbi inference. (a) Precision as a function of number of downloaded pages and relevance threshold. The mesh has 400 points along the axis of number of downloaded pages and 5 points along the axis of relevance threshold; (b) Maximum Average Similarity as a function of the number of downloaded pages.

tal results for CRF are consistent with the results of MEMM using marginal mode and Viterbi inference discussed previously.

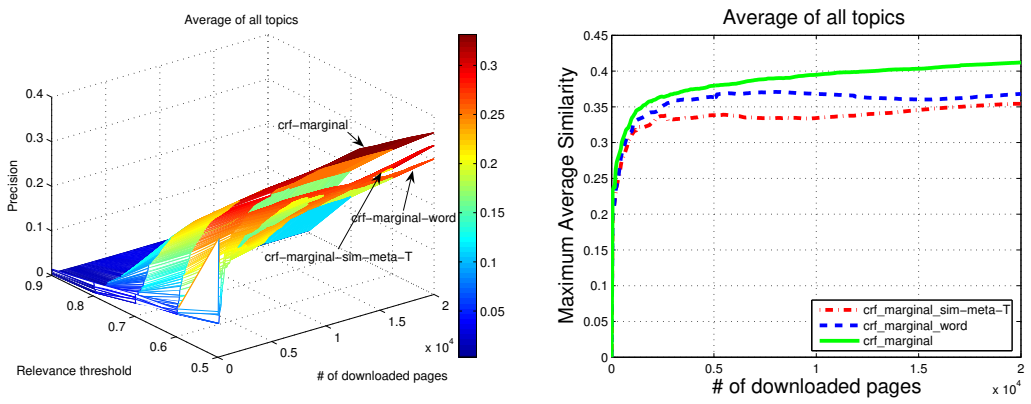
6.4 Comparison of CRF crawlers with Different Features.

Next we test the impact of the features on the performance. In this experiment, we compare CRF-marginal crawl (which performs better than CRF-viterbi) with different features: with all features, with Word feature only, and with the remaining features: Text feature, Description feature, URL token feature and Anchor text feature (described in Section 3.2). We denote them as *CRF-marginal, CRF-marginal-word, and CRF-marginal-sim-meta-T* in the figures, respectively.

We observe in Fig. 13 that CRF-marginal crawl with all features combination performs better on average than with Word feature only and with non-Word features only. This confirms that, even if only some of the features are used, relevant paths can be effectively identified. Furthermore, we observe, like in the MEMM crawler, that the contribution of the non-Word features is minimal, since the performance using all the features and the word features only is very similar.

7 Comparison of MEMM- and CRF- Based crawlers

Although MEMM and CRF are different models - MEMM is a directed discriminative model, whereas CRF is an undirected graphical model - they have many properties in common [21]. Both of them assume the first-order Markov assumption, i.e. $p(s_{t+1}|s_t) = p(s_{t+1}|s_t, s_{t-1})$, and calculate the conditional probability $p(s|o)$ directly. Both of them use the same training data and incorporate the same multiple features.



(a) (b)

Figure 13: Performance of the CRF-based crawler using marginal mode inference with different features. (a) Precision as a function of number of downloaded pages and relevance threshold. The mesh has 400 points along the axis of number of downloaded pages and 5 points along the axis of relevance threshold; (b) Maximum Average Similarity as a function of the number of downloaded pages.

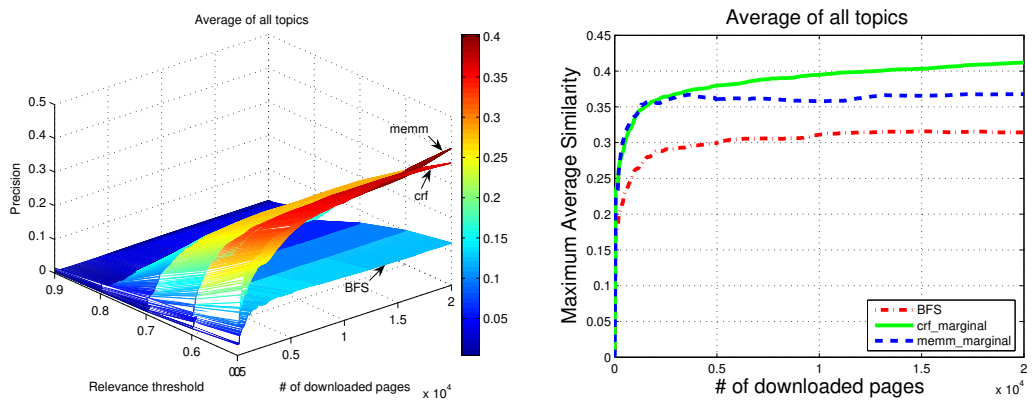
MEMM can be seen as a localized version of CRF since the important difference between the two is that MEMM uses a per-state local normalizer while CRF assumes a global normalizer over the whole sequence. Therefore, it is appropriate to compare these two models on the focused crawling problem.

Since both CRF and MEMM showed better performance with the marginal mode in previous sections, we compare CRF-marginal and MEMM-marginal against BFS crawler. As observed in Fig. 14, CRF-marginal crawler slightly outperforms MEMM-marginal crawler on average.

Generally speaking, the performance with CRF is better than with MEMM on these topics. CRF exhibits slight improvement in predicting the next important links to follow. Global normalization in CRF seems to help since this is the only difference between MEMM and linear-chain CRF.

Comparison of Computational Costs with BFS, MEMM, and CRF It is interesting to compare the computational cost of graphical models for focused crawling. For a supervised learning problem, there are two main steps: learning/training and prediction/inference. Compared to the BFS crawl, which has no extra cost in training and low computational cost on prediction during crawling, graphical model-based methods incur higher computational cost on both training and crawling.

During the training, since MEMM calculates the normalization factor over each position, in theory it has significantly lower computational cost compared to CRF, which needs to compute a global normalizer. However, CRF training can be implemented efficiently using dynamic programming, therefore the computational cost differences between MEMM and CRF are minor. The left side of Table 3 shows the average computational cost over all topics for MEMM and CRF crawlers. In our experiments, which use L-BFGS for training, the training stops either when the num-



(a) (b)

Figure 14: Comparing the Performance of MEMM- and CRF-based crawler using marginal mode inference. (a) Precision as a function of number of downloaded pages and relevance threshold. The mesh has 400 points along the axis of number of downloaded pages and 5 points along the axis of relevance threshold; (b) Maximum Average Similarity as a function of the number of downloaded pages.

ber of iterations reaches 100, which is the maximum number of iterations over the training data, or until convergence.

Table 3: Computational Costs

Topics	Training Time (s)			Crawling Time (s)		
	BFS	MEMMs	CRFs	BFS	MEMMs	CRFs
Mean	-	836	846	22,799	27,423	30,631
Standard Deviation	-	245	243	9,398	7,327	13,407

During crawling, all methods follow the process shown in Fig. 8. The two boxes marked *Prediction* and *Analyze Page* are the only differences among the different methods. BFS crawler has lower computational cost since it only involves text preprocessing, extracting links, calculating similarity value between the targets and the newly-seen page, and assigning visit priority based on the sorted similarity values of URLs in the queue. MEMM and CRFs crawlers, on the other hand, require additional feature extraction (Section 3.2) and MEMM/CRF inference based on parameters learned from training data.

In our implementation, we use queue size equal to 500 and there are parts of the code that intentionally slow down the crawlers, as they respect the Robot Exclusion Protocol and spread the crawling load over many remote Web servers. For example, the focused crawler follows the top URLs in the queue; if two consecutive URLs in the queue have the same server domain, the crawler will wait ten seconds to avoid overloading the same server.

The right side of Table 3 gives the average of computational costs for different crawl

strategies on all topics. In general, the BFS crawler requires less time for crawling than the MEMM and CRF crawlers. However, the relatively small differences among methods imply that crawling time is dominated by network delays, not computation time.

The times shown in Table 3 may appear very slow compared with state-of-the-art crawlers. It should be pointed out that focused crawling is more difficult to optimize through techniques like multithreading or cluster computing because of the need for queue maintenance and prioritization. Speeding up focused crawling is a topic of future research.

7.1 Comparison of MEMM, CRF, Hidden Markov Model, and Context Graph crawling

In our previous research, we introduced a focused crawler using a Hidden Markov Model (HMM) to learn from user's browsing patterns, and compared its performance with BFS and CGS crawling [22]. In that study, training data was created from user-visited pages, and target pages were marked by the user during browsing. The crawler based on HMM showed better results than those based on CGS and BFS, which motivated us to move towards exploiting multiple features to find relevant pages in a broader extent with MEMM and CRF. To have sufficient training data as required for learning from multiple features with MEMM and CRF, we chose a different way to obtain training data for this work, relying on a Web Data Collection. Target pages on the topics were chosen from the ODP, and training data constructed as described in Section 3.1.

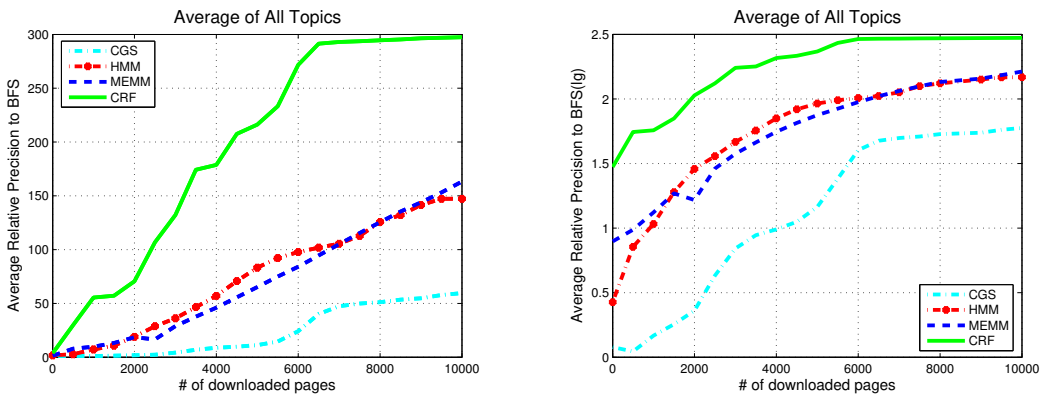
HMM-based crawling and Context Graph Search (CGS), in our previous study [22], were performed under different assumptions from the current study. In the previous study training data was collected through user browsing, whereas in the current study it was obtained by crawling ODP. As a result training data and target pages were different.

To perform a meaningful comparison, we use *Average Precision Relative to BFS* (APR) measure, which *indirectly* compares the relative performance of CGS, HMM, MEMM, and CRF against the baseline method, BFS. *Average Precision Relative to BFS*(APR) is defined as follows:

$$APR = \frac{\texttt{Average precision using the method to be evaluated}}{\texttt{Average precision using the baseline method, BFS}} \tag{40}$$

The denominator in Eq. 40 depends on the target pages, hence it is different between the previous and current study.

Essentially, we compare the relative performance of different methods over BFS crawl: the larger the APR, the better performance the method has. We first calculate the precision relative to BFS for each similarity threshold 0.5-0.8, then average over all thresholds. The average APR over the four common topics between the two studies (Biking, Call for Papers, Hockey, Linux) is shown in Fig. 15. We observe that, on average, CRF outperforms the other three methods (CGS, HMM and MEMM), while



(a) (b)

Figure 15: Average Precision Relative to BFS (APR) of the CRF, MEMM, HMM, and CGS crawling methods over four topics, Biking, Call for Papers, Hockey and Linux. (a) APR averaged over the four topics. (b) log_{10} of (a), to show details near the origin.

HMM and MEMM have similar performance. All three sequential pattern modelling methods outperform CGS and BFS, since the ARP is greater than one.

7.2 Discussion

We modeled the focused crawling problem as a sequential task and have applied HMM-, MEMM- and CRF-based models for capturing sequential patterns of pages leading to target pages. The following table summarizes the advantages and disadvantages of three graphical models.

Table 4: Pros (+) and Cons (−) of Three Graphical Models

Models	First-order Markov	Label Bias	Flexibility of Features	Global Solutions	Computational Cost
HMM	+	+	-	-	+
MEMM	+	-	+	-	-
CRF	+	+	+	+	-

Based on the assumptions and definition of HMM discussed in [22], this model lacks the flexibility to use arbitrary features and the ability to obtain global solutions. We apply HMM to learn from the user's browsing data, since the user-visited data contains strong sequence structures about the topic when the user seeks to fulfill his/her topic-specific information need. MEMM is a more expressive model, it makes fewer assumptions about the data than HMM, and allows arbitrary and overlapping features. However, MEMM uses a local normalization factor at each position, so that it actually forms a chain of local models, that is, each position contains a "local classifier". As a result, MEMM may suffer the label bias problem [21], i.e. the total

probability received by s_{t-1} must be passed on to the state labels at next position t even if o_t is completely incompatible with s_{t-1}, since the distribution sums to 1 across next state s_t at position t. Therefore, MEMM also gives locally optimal solutions rather than global solutions. Linear-chain CRF, on the other hand, has all the advantages of MEMM without the label bias problem. CRF uses a global normalizer which results in a global solution and avoids the label bias problem. Like MEMM, CRF uses multiple features from the data to solve the problem directly, rather than using an intermediate step, as in HMM. Experimental results show that CRF is able to outperform MEMM and HMM, and the properties of CRF make it possible to capture the topical relations better along the hyperlinks for the focused Web crawling.

8 Conclusion

The goal of this research has been a novel approach for focused crawling that captures sequential patterns along paths leading to targets based on probabilistic models. Our approach involves the use of a combination of content analysis and link structure of paths leading to targets to learn focused crawling strategies from training data. The system is unique in that it models the process of crawling by a walk along an underlying chain of hidden states, defined by hop distance from target pages, from which the actual topics of the documents are observed. When a new document is seen, prediction amounts to estimating the distance of this document from a target. In this way, good performance depends on powerful modeling of context as well as current observations.

Two probabilistic models have been explored in this work, based on MEMM and linear-chain CRF, following our previous study of HMM-based focused crawlers [22], which demonstrated that an HMM crawler trained on user browsing data performs better than a Context-Graph crawler and a Best-First crawler. For each of the models within this framework, we developed an inference and a learning algorithm.

We further extended our work to take advantage of richer representations of multiple features extracted from Web pages. The advantages and flexibility of MEMM- and CRF-based crawlers fit our approach well and are able to represent useful context. With the MEMM-based model, we exploit multiple overlapping features, such as anchor text, to represent useful context and form a chain of local classifier models. With Linear-chain Conditional Random Field Model (CRF), a form of undirected graphical model, we further focus on obtaining global optimal solutions along the sequences. Experimental results demonstrate that focused Web crawling using MEMM/CRF is a promising approach over Context Graph or Best-First search, and offers advantages over our previous HMM-based model.

We studied the impact of different formulations of the MEMM- and CRF-based crawling, and demonstrated that using marginal mode for crawling performs better than using the Viterbi algorithm, and the crawler using the combination of all feature types performs consistently better than the crawler that depends on just one or some of them. We also introduced the *Average Precision Relative to BFS* (APR) measure

to compare models under different assumptions about input data.

Topics for further research include: (i) the sensitivity of the focused crawling algorithms to the presence of noise in the training data; (ii) more advanced representations of web pages that capture their semantics better than the vector space model used in this study; (iii) the degree of specificity of the relevant topic, and how this may affect the design and performance of focused crawlers.

Acknowledgements. The research presented here was funded by the Natural Sciences and Engineering Research Council of Canada (NSERC), the MITACS NCE, and IT Interactive Services Ltd. The contribution to this research by Prof. Jeannette Janssen is gratefully acknowledged.

References

[1] C. Aggarwal, F. Al-Garawi, and P. Yu. Intelligent Crawling on the World Wide Web with Arbitrary Predicates. In *Proceedings of the 10th International WWW Conference*, Hong Kong, 2001.

[2] R. Albert, H. Jeong, and A. Barabasi. Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.

[3] G. Almpanidis, C. Kotropoulos, and I. Pitas. Combining text and link analysis for focused crawling - an application for vertical search engines. *Information Systems*, 32(6):886908, 2007.

[4] L. Barbosa and J. Freire. Searching for hidden web databases. In *Eighth International Workshop on the Web and Databases (WebDB)*, Baltimore, Maryland, June 16-17 2005.

[5] L. Barbosa and J. Freire. An adaptive crawler for locating hidden web entry points. In *World Wide Web Conference (WWW)*, pages 441–450, Banff, Alberta, Canada, May 812 2007.

[6] D. Bergmark, C. Lagoze, and A. Sbityakov. Focused Crawls, Tunneling, and Digital Libraries. In *Proceedings of the 6th European Conference on Digital Libraries*, Rome, Italy, 2002.

[7] S. Chakrabarti. Integrating the document object model with hyperlinks for enhanced topic distillation and information extraction. In *Proceedings of the 10th International WWW Conference*, Hong Kong, May 2001. Accessed on July 10, 2010 at `http://www10.org/cdrom/papers/489`.

[8] S. F. Chen and R. Rosenfeld. A Gaussian Prior for Smoothing Maximum Entropy Models. Technical Report CMU-CS-99-108, CMU, 1999.

[9] S. F. Chen and R. Rosenfeld. Inducing features of random fields. *A Survey of Smoothing Techniques for ME Models*, 8(1):37–50, 2000.

[10] Z. Chen, J. Ma, J. Lei, B. Yuan, L. Lian, and L. Song. A cross-language focused crawling algorithm based on multiple relevance prediction strategies. *Computers and Mathematics with Applications*, doi:10.1016/j.camwa.2008.09.021, 2008.

[11] Y. Choi, K. Kim, and M. Kang. A focused crawling for the web resource discovery using a modified proximal support vector machines. In O. Gervasi et al., editor, *ICCSA 2005*, volume LNCS 3480, pages 186–194, Berlin Heidelberg, 2005. Springer-Verlag.

[12] B. D. Davison. Topical locality in the Web. In *Proceedings of the 23rd Annual International Conference on Research and Development in Information Retrieval (SIGIR 2000)*, pages 272–279, Athens, Greece, July 2000.

[13] G. de Assis, A. Laender, A. da Silva, and M. André Gonçalves. The impact of term selection in genre-aware focused crawling. In *SAC '08: Proceedings of the 2008 ACM symposium on Applied computing*, pages 1158–1163, New York, NY, USA, 2008. ACM.

[14] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, and M. Gori. Focused Crawling Using Context Graphs. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB 2000)*, Cairo, Egypt, 2000.

[15] M. Dudik and R. E. Schapire. Maximum Entropy Distribution Estimation with Generalized Regularization. In *Proceedings of the 19th Annual Conference on Learning Theory*, pages 123–138, 2006.

[16] M. Ehrig and A. Maedche. Ontology-focused crawling of web documents. In *SAC '03: Proceedings of the 2003 ACM symposium on Applied computing*, pages 1174–1178, New York, NY, USA, 2003. ACM.

[17] J. Goodman. Exponential Priors for Maximum Entropy Models. In *Proceedings of the ACL-04*, 2004.

[18] T.H. Haveliwala. Topic-sensitive pagerank: a context-sensitive ranking algorithm for web search. *Knowledge and Data Engineering, IEEE Transactions on*, 15(4):784–796, July-Aug. 2003.

[19] M. Hersovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The Shark-Search Algorithm - An Application: Tailored Web Site Mapping. In *Proceedings of the Seventh International WWW Conference*, Brisbane, Australia, 1998.

[20] J. Johnson, K. Tsioutsiouliklis, and C. Lee Giles. Evolving Strategies for Focused Web Crawling. In *Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003)*, Washington D.C., USA, 2003.

[21] J. Lafferty, A. McCallum, and 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.

[22] H. Liu, J. Janssen, and E. Milios. Using HMM to learn user browsing patterns for focused web crawling. *Data & Knowledge Engineering*, 59(2):270–291, November 2006.

[23] H. Liu and E. Milios. Probabilistic models for focused web crawling. Technical Report -1, Faculty of Computer Science, Dalhousie University, Halifax, Canada, 2009.

[24] Y. Liu, J. Carbonell, P. Weigele, and V. Gopalakrishnan. Segmentation Conditional Random Fields (SCRFs): A New Approach for Protein Fold Recognition. In *Proceedings of the ACM International Conference on Research in Computational Molecular Biology (RECOMB05)*, 2005.

[25] R. Malouf. A Comparison of Algorithms for Maximum Entropy Parameter Estimation. In *Sixth Workshop on Computational Language Learning (CoNLL)*, 2002.

[26] A. McCallum. Efficiently Inducing Features of Conditional Random Fields. In *Proceedings of the Conference on Uncertainty in AI (UAI)*, 2003.

[27] A. Mccallum, D. Freitag, and D. Freitag. Maximum entropy markov models for information extraction and segmentation. In *Proceedings of the Seventeenth International Conference on Machine Learning*, pages 591–598. Morgan Kaufmann, 2000.

[28] A. McCallum and W. Li. Early results for Named Entity Recognition with Conditional Random Fields, Feature Induciton and Web-enhanced Lexicons. In *Proceedings of the Seventh Conference on Natural Language Learning(CoNLL)*, 2003.

[29] F. Menczer. Links tell us about lexical and semantic Web content. In *Technical Report Computer Science Abstract CS.IR/0108004*, arXiv.org, Aug. 2001. Accessed on July 10, 2010 at `http://arxiv.org/abs/cs.IR/0108004.`

[30] F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. In *Machine Learning*, pages 39(2/3):203–242, 2000.

[31] F. Menczer, G. Pant, and P. Srinivasan. Topical Web Crawlers: Evaluating Adaptive Algorithms. *ACM TOIT*, 4(4):378–419, 2004.

[32] F. Menczer, G. Pant, P. Srinivasan, and M. Ruiz. Evaluating Topic-Driven Web Crawlers. In *Proceedings of the 24th Annual International ACM/SIGIR Conference. Research and Development in Information Retrieval*, New Orleans, USA, 2001.

[33] J. Nocedal and S.J.Wright. *Numerical Optimization*. Springer, 1999.

[34] G. Pant and P. Srinivasan. Learning to Crawl: Comparing Classification Schemes. In *ACM Trans. Information Systems*, pages vol. 23, no. 4, 2005.

[35] G. Pant and P. Srinivasan. Link Contexts in Classifier-Guided Topical Crawlers. *IEEE Transactions on Knowledge and Data Engineering*, 18(1):107–122, 2006.

[36] G. Pant, K. Tsioutsiouliklis, J. Johnson, and C.L. Giles. Panorama: Extending Digital Libraries with Topical Crawlers. In *Proceedings of ACM/IEEE Joint Conference on Digital Libraries (JCDL 2004)*, pages 142–150, Tucson, Arizona, June 2004.

[37] F. Peng and A. McCallum. Accurate information extraction from research papers using conditional random fields. In *Human Language Technology Conference and North American Chapter of the Association for Computational Linguistics (HLT/NAACL-04)*, 2004.

[38] D. Pinto, A. McCallum, X. Wei, and W. Bruce Croft. Table Extraction Using Conditional Random Fields. In *Proceedings of the 26th Annual International ACM SIGIR conference*, Toronto, Canada, 2003.

[39] A. Ratnaparkhi. A Maximum Entropy Part-Of-Speech Tagger. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 133–142, 1996.

[40] J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. In *Proceedings of the Sixteenth International Conference on Machine Learning(ICML-99)*, Bled, Slovenia, 1999.

[41] S. Sarawagi and W. Cohen. Semi-Markov Conditional Random Fields for Information Extraction. In *Advances in Neural Information Processing System*, pages Volume 17, Pages 1185–1192, 2005.

[42] K. Sato and Y. Sakakibara. RNA Secondary Structural Alignment with Conditional Random Fields. In *Bioinformatics*, pages Volume 21, Pages 237–242, 2005.

[43] F. Sha and F. Pereira. Shallow Parsing with Conditional Random Fields. In *Proceedings of the 2003 Conference of the North American Chapter of the Association for Computational Linguistics on Human Language Technology (NAACL'03)*, pages 134–141, Morristown, NJ, USA, 2003.

[44] D. Shen, J-T. Sun, H. Li, Q. Yang, and Z. Chen. Document summarization using conditional random fields. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence (IJCAI 07)*, Hyderabad, India. January 6-12, 2007.

[45] C. Sherman and G. Price. *The Invisible Web: Uncovering Information Sources Search Engines Can't See*. Cyberage Books, 2003.

[46] T-K. Shih. Focused crawling for information gathering using hidden markov model. Master's thesis, Computer Science and Information Engineering, National Central University, Taiwan, October 2007.

[47] P. Srinivasan, F. Menczer, and G. Pant. A General Evaluation Framework for Topical Crawlers. *Information Retrieval*, 8(3):417–447, 2005.

[48] M. Subramanyam, G.V.R. Phanindra, M. Tiwari, and M. Jain. Focused Crawling Using TFIDF Centroid. In *Hypertext Retrieval and Mining*, Apr. 2001.

[49] C. Sutton, A. McCallum, and K. Rohanimanesh. Dynamic conditional random fields: Factorized probabilistic models for labeling and segmenting sequence data. *J. Mach. Learn. Res.*, 8:693–723, 2007.

[50] A. Tsoi, D. Forsali, M. Gori, M. Hagenbuchner, and F. Scarselli. A simple focused crawler. In *World Wide Web Conference*, pages 181–184, Budapest, Hungary, May 20-24 2003.

[51] H. Wallach. Conditional Random Fields: An Introduction. Technical Report MS-CIS-04-21, Department of Computer and Information Science, University of Pennsylvania, 2004.

[52] Search Engine Watch, August 2010. Accessed on July 10, 2010 at `http://searchenginewatch.com`.