Learning to extract information from large websites using sequential models

V.G.Vinod Vydiswaran
vgvinodv@iitb.ac.in

Sunita Sarawagi
sunita@iitb.ac.in

ABSTRACT

We propose a new method of information extraction from large websites by learning the sequence of links that lead to a specific goal page on the website. Sample applications include finding computer science publications starting from university root pages and fetching addresses of companies on a web database.

We model the website as a graph on a set of important states chosen via domain knowledge and train a Conditional Random Field (CRF) over it. The conditional exponential models of CRFs enable us to exploit a variety of features including keywords and patterns extracted from and around hyperlinks and HTML pages and any sequential orderings amongst states. Our technique provides two times better harvest rates than techniques used in generic focused crawlers.

1. INTRODUCTION

We propose a new method of information extraction from large websites by learning the sequence of links that lead to a specific goal page on the website. Often websites within a domain are structurally similar to each other. Humans are good at navigating these websites to reach specific information within large domain-specific websites. Our goal is to learn the navigation path by observing the user's clicks on as few example searches as possible and then use the learnt model to automatically find the desired pages using as few redundant page fetches as possible. Unlike in focused crawling[4], our goal is not to locate the websites to start with. These are collected either from web directories and similar resource websites, or obtained through focused crawling. We start from a listing of related websites and after watching the user find the specific information from a few websites in the list, we automate the search in the remaining.

1.1 Motivating Examples

We list a number of examples where such a functionality would be useful.

1.1.1 Company Contact Information

Suppose a user needs to collect the contact addresses of a number of companies. One motivating reason could be to apply for jobs. The user might only be interested in jobs that are available in a particular state or region. There are many websites[1] that give the URLs of the homepages of companies. Instead of visiting each of these URLs manually and checking if they meet his location constraints, the user would like to collect all contact addresses automatically and do a simple search to narrow down the list.

Is it possible to extract out this information and provide it directly to the user?

Usually, homepages of companies contain links with anchor text like "Contact Us" or "About Us". Following this link leads either to a page with the contact addresses, or to another page that lists links to various offices of the company. Contact information can be obtained by following these links.

1.1.2 Faculty publications

Citation portals like Citeseer need to gather publications on a particular discipline from homepages of faculty and students starting from lists of universities easily obtained from web directories like Dmoz.

Is it possible to help citation portals in finding all the faculty publications from an institute in a particular discipline, given the homepage of the institute?

This is a more difficult, yet compelling, example. Faculty members may have a list of their publications on their homepages; and the required result is a compilation of all such listings. Hence, the solution is to first visit the homepages of departments relevant to the discipline, from there visit the homepages of faculty members, and then search for links such as "Papers", "Publications", or "Research Interests" that lead to the publications page, if it exists.

1.1.3 Seminar Announcements

Suppose a researcher wants to keep track of technical talks from a list of favourite institutions. These talks are usually announced either at the departmental level or at the level of research groups within a department.

[1] E.g., `http://www.hoovers.com/`, `http://www.kdnuggets.com/`, etc.

ADVANCES IN DATA MANAGEMENT 2005
Jayant Haritsa, T.M. Vijayaraman (Editors)

Is it possible to provide that information, starting from homepages of the institutes in the user's favourite list?

This is similar to the faculty publications case above except that the path to talk announcements tend to be less templatized. Often talk announcements have a direct link from a department's main page and the link contains words like "events", "talks", "seminars", and so on. At other times, talk announcements can only be reached through research group or project pages within a department website.

1.1.4 Ph.D. requirements

A busy professor is asked to design the Ph.D. requirements for a new department. Naturally, she would first like to study the Ph.D. requirements of several existing departments. A list of such departments is readily accessed from a web directory. However, in our experience, searching for the requirements page starting from a department homepage is not obvious to start with but later a few patterns start to emerge. We would like to be able to automatically learn such patterns and use that to automatically find the requirements page from the remaining pages.

1.1.5 Electronic shopping

A similar scenario arises when a user is searching for a particular product on electronic stores and wishes to automatically find if the product is on sale on each of a given list of websites. Most of these follow similar product hierarchies which need to be navigated to reach the desired product.

In each of the above examples, it is not easy to formulate the user's needs using keywords alone. So keyword searches using a search engine with the domain restricted to each link in the list may not be sufficient. In these cases, the path leading to the goal page is part of the definition of the user's need. For example, a company website might list addresses of its partners on a page that might be hard to distinguish from a page listing its location. A stronger indicator of a page listing the company's contact address is that this page is within one or two links away from the root page and is associated with anchor words like "Contact us". This implies that a method that judges if a page is relevant or not just based on its content is not likely to be as accurate. The actual goal page may not have all the information to correctly identify the page as the goal page.

1.2 Problem Statement

There are two phases to this task: first is the training phase, where the user teaches the system by clicking through pages and labeling a subset with a dynamically defined set of classes, one of them being the Goal class. The classes assigned on intermittent pages along the path can be thought of as "milestones" that capture the structural similarity across websites. At the end of this process, we have a set of classes C and a set of training paths where a subset of the pages in the path are labeled with a class from C. All unlabeled pages before a labeled page are represented with a special prefix state for that label. The system trains a model using the example paths, modeling each class in C as a milestone state. The second phase is the foraging phase where the given list of websites are automatically navigated to find all goal pages.

In practice, we can also easily add an active learning phase where the system starts with a very small number of labeled paths and uses active learning[1] to seek labels of as few paths as possible from the user.

Formally, we are given a website as a graph $W(V, E)$ consisting of vertex set V and edge set E, where a vertex is a webpage and an edge $e = \langle u, v \rangle$ is a hyperlink pointing from a webpage u to a webpage v. The goal pages P_G constitute a subset of pages in W reachable from starting seed page P_S. We have to navigate to them starting from P_S visiting fewest possible additional pages. Let $\mathcal{P} : P_1, P_2, \ldots, P_n$ be one such path through W from the start page $P_1 = P_S$ to a goal page $P_n \in P_G$. The ratio of relevant pages visited to the total number of pages visited during the execution is called the **harvest rate**. The objective function is to maximize the harvest rate.

There are two parts to solving this problem.

Recognizing a page as the goal page. This is a classification problem where given a webpage we have to classify it as being a goal page or not. Often the page alone may not hold enough information to help identify it as the goal page. We will need to consider text around the entire path leading to the goal page in order to decide if it is relevant or not. For example, consider the publications scenario explained in section *1.1.2*. If Citeseer wants to get all computer science publications starting from a university root page, then it is necessary to follow a path through computer science and related departments' homepages. A publication page on its own might be hard to classify as holding "computer science publications".

Foraging for goal pages. This can be thought as a mini-crawling exercise where, starting from the entry point, we want to visit as few pages as possible in finding the goal pages. This problem is different from the previous work on focused crawling[4] where the goal is to find all webpages relevant to a particular broad topic from the entire web. In our case, we are interested in finding pages on a specific information need starting from given entry pages of several related websites. We exploit the regularity in the structures of websites in a given domain to build more powerful models than is possible in the case of general-purpose focused crawlers.

1.2.1 Possible Approaches

One possible method of solving the problem is to train a classifier that can discriminate the goal pages from the non-goal pages. Then, extract from the classifier the set of prominent features to serve as keywords to a search engine that indexes all the websites of interest. By restricting the domain to each given starting URL in turn, we issue a keyword search to get a set of candidate pages. We further classify these pages to identify if these are goal pages or not. We will show in the experimental section (section *4.3.2*) that this method cannot provide high accuracy for the simple reason that the goal page itself may not hold enough information to correctly identify it as the goal page. The path leading to the goal page is important too.

A second method is to cast this as a generic focused crawling problem. We will show in section *4.4.2* a comparison with this method.

A third approach and the one that we develop is to treat

this as a sequential labeling problem where we use graphical models like Hidden Markov Models (HMMs)[22] and their recent advanced conditional counterparts, the Conditional Random Fields[14], to learn to recognize paths that lead to goal states and then superimpose ideas from Reinforcement Learning[20] to prioritize the order in which pages should be fetched to reach the goal page. This provides an elegant and unified mechanism of modeling the path learning and foraging problem. Also, as we will see in the experimental section (section 4) that it provides very high accuracy.

2. GOAL PAGE RECOGNITION

In this section, we will address the problem of goal page recognition wherein given a path of webpages $P_1 \ldots P_n$, where $P_1 = P_S$ is the starting page, our aim is to recognize if the path ends in a goal page or not. During training, we are given examples of several paths of labeled pages where some of the paths end in goal pages and others end with a special "fail" label. We cast this as a sequential labeling problem: the set of pages is denoted by the vector $\boldsymbol{x}$ and their corresponding labels is denoted by $\boldsymbol{y}$. Each x_i is a webpage represented suitably in terms of features derived from the words in the page, its URL, and the anchor text in the link pointing to x_i. A number of methods have been proposed in the literature for solving this problem in the context of applications like speech recognition and Information Extraction[7].

One popular mechanism so far has been hidden Markov models that during training, learn a joint probability $\Pr(\boldsymbol{x}, \boldsymbol{y})$ of pairs of observation sequences $\boldsymbol{x}$ and label sequences $\boldsymbol{y}$. The parameters of the model are trained to maximize the joint likelihood of the training examples. A major shortcoming of generative models like HMMs is that they maximize the joint probability of sequence and labels. This does not necessarily maximize accuracy. Also, the conditional independence of features is a restrictive assumption. Conditional Random Fields (CRFs) are a recently introduced formalism that learn a single global conditional model for $\Pr(\mathbf{y}|\mathbf{x})$[14] and have been found to achieve high accuracy in a number of applications.

2.1 Background on CRFs

A CRF models $\Pr(\mathbf{y}|\mathbf{x})$ as a Markov random field, with nodes corresponding to elements of the structured object $\mathbf{y}$, and potential functions that are conditional on (features of) $\mathbf{x}$. One common use of CRFs is for sequential learning problems like NP chunking[25], POS tagging[14], and named-entity recognition (NER)[19]. For these problems, the Markov field is a chain and $\mathbf{y}$ is a linear sequence of labels from a fixed set $\mathcal{Y}$, and the label at position i depends only on its previous label. For instance, in the NER application, where the task is to identify entity types like people names and organization in plain text, $\mathbf{x}$ might be a sequence of words, and $\mathbf{y}$ might be a sequence in $\{I, O\}^{|\mathbf{x}|}$, where $y_i = I$ indicates "word x_i is inside a name" and $y_i = O$ indicates the opposite.

Notation: We will use bold-faced symbols to denote vectors and non-bold faced symbols to denote scalars.

Assume a vector $\mathbf{f}$ of *local feature functions* $\mathbf{f} = \langle f^1, \ldots, f^K \rangle$, each of which maps a pair $(\mathbf{x}, \mathbf{y})$ and a position i in the vector $\mathbf{x}$ to a measurement $f^k(i, \mathbf{x}, \mathbf{y}) \in \mathbb{R}$. Let $\mathbf{f}(i, \mathbf{x}, \mathbf{y})$ be the vector of these measurements and let $\mathbf{F}(\mathbf{x}, \mathbf{y}) = \sum_i^{|\mathbf{x}|} \mathbf{f}(i, \mathbf{x}, \mathbf{y})$. For the case of NER, the components of $\mathbf{f}$ might include the measurement $f^{13}(i, \mathbf{x}, \mathbf{y}) = [\![x_i \text{ is capitalized}]\!] \cdot [\![y_i = I]\!]$, where the indicator function $[\![c]\!] = 1$ if c if true and 0 otherwise; this implies that $F^{13}(\mathbf{x}, \mathbf{y})$ would be the number of capitalized words paired with the label I.

For the sake of efficiency, we restrict any feature $f^k(i, \mathbf{x}, \mathbf{y})$ to be **local** in the sense that the feature at a position i will depend only on the previous labels. With a slight abuse of notation, we claim that a local feature $f^k(i, \mathbf{x}, \mathbf{y})$ can be expressed as $f^k(y_i, y_{i-1}, \mathbf{x}, i)$. Some subset of these features can be simplified further to depend only on the current state and are independent of the previous state. We will refer to these as **state features** and denote these by $f^k(y_i, \mathbf{x}, i)$ when we want to make the distinction explicit. The term **transition features** refers to the remaining features that are not independent of the previous state.

A Conditional Random Field (CRF)[14, 25] is an estimator of the form

$$\Pr(\mathbf{y}|\mathbf{x}, \mathbf{W}) = \frac{1}{Z(\mathbf{x})} e^{\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y})} \tag{1}$$

where $\mathbf{W}$ is a weight vector over the components of $\mathbf{F}$ and the normalizing term $Z(\mathbf{x}) = \sum_{\mathbf{y}'} e^{\mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}')}$.

2.2 An efficient inference algorithm

The *inference problem* for a CRF is defined as follows: Given $\mathbf{W}$ and $\mathbf{x}$, find the best label sequence, $\arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}, \mathbf{W})$, where $\Pr(\mathbf{y}|\mathbf{x}, \mathbf{W})$ is defined by equation 1.

$$\begin{aligned} \arg\max_{\mathbf{y}} \Pr(\mathbf{y}|\mathbf{x}, \mathbf{W}) &= \arg\max_{\mathbf{y}} \mathbf{W} \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}) \\ &= \arg\max_{\mathbf{y}} \mathbf{W} \cdot \sum_j \mathbf{f}(y_j, y_{j-1}, \mathbf{x}, j) \end{aligned}$$

An efficient inference algorithm is possible because all features are assumed to be local. Let $\mathbf{y}_{i:y}$ denote the set of all partial labels starting from 1 (the first index of the sequence) to i, such that the i-th label is y. Let $\delta(i, y)$ denote the largest value of $W \cdot \mathbf{F}(\mathbf{x}, \mathbf{y}')$ for any $\mathbf{y}' \in \mathbf{y}_{i:y}$. The following recursive calculation implements the usual Viterbi algorithm[22]:

$$\delta(i, y) = \begin{cases} \max_{y'} \ \delta(i-1, y') + \mathbf{W} \cdot \mathbf{f}(y, y', \mathbf{x}, i) & \text{if } i > 0 \\ 0 & \text{if } i = 0 \end{cases} \tag{2}$$

The best label then corresponds to the path traced by $\max_y \delta(|\mathbf{x}|, y)$.

2.3 Training algorithm

Learning is performed by setting parameters to maximize the likelihood of a set of a training set $T = \{(\mathbf{x}_\ell, \mathbf{y}_\ell)\}_{\ell=1}^N$ expressed in logarithmic terms as

$$L(\mathbf{W}) = \sum_\ell \log \Pr(\mathbf{y}_\ell|\mathbf{x}_\ell, \mathbf{W}) = \sum_\ell (\mathbf{W} \cdot \mathbf{F}(\mathbf{x}_\ell, \mathbf{y}_\ell) - \log Z_{\mathbf{W}}(\mathbf{x}_\ell))$$

We wish to find a $\mathbf{W}$ that maximizes $L(\mathbf{W})$. The above equation is convex and can thus be maximized by gradient ascent or one of many related methods. (In our implementation, we use a limited-memory quasi-Newton method[15, 18].) The gradient of $L(\mathbf{W})$ is the following:

$$\nabla L(\mathbf{W}) = \sum_{\ell} \mathbf{F}(\mathbf{x}_\ell, \mathbf{y}_\ell) - \frac{\sum_{\mathbf{y}'} \mathbf{F}(\mathbf{x}_\ell, \mathbf{y}') e^{\mathbf{W}\cdot\mathbf{F}(\mathbf{x}_\ell, \mathbf{y}')}}{Z_{\mathbf{W}}(\mathbf{x}_\ell)}$$
$$= \sum_{\ell} \mathbf{F}(\mathbf{x}_\ell, \mathbf{y}_\ell) - E_{\Pr(\mathbf{y}'|\mathbf{W})}\mathbf{F}(\mathbf{x}_\ell, \mathbf{y}')$$

The first set of terms are easy to compute. However, we must use the Markov property of $\mathbf{F}$ and a dynamic programming step to compute the normalizer $Z_{\mathbf{W}}(\mathbf{x}_\ell)$, and the expected value of the features under the current weight vector, $E_{\Pr(\mathbf{y}'|\mathbf{W})}\mathbf{F}(\mathbf{x}_\ell, \mathbf{y}')$. Details of computing these can be found be in [25].

2.4 Using CRFs for path classification

A number of design decisions about the label space and the feature space need to be made in casting the path classification problem into a CRF.

2.4.1 Single state per label

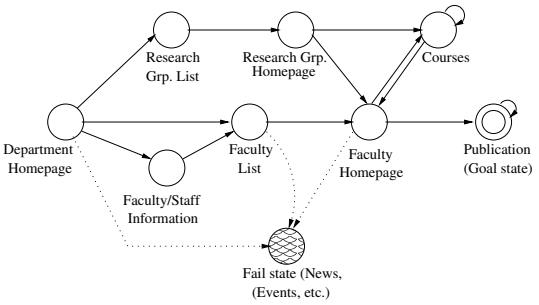


Figure 1: State transition diagram of the model for Publications domain with a single state per label. The `Fail` state has in-links from all other states (only a few shown above) and represents negative path end-state.

One option is to assign a state to each possible label in the set L where two labels are special "Goal" and "Fail". During classification, if the last state in the predicted label sequence $\mathbf{y}$ is labeled "Goal", we classify this as a positive path; otherwise it is considered a negative path.

An example of such a model for the Publications scenario (section *1.1.2*) is given in figure 1 where each circle represents a label. Any path classified as leading to the Goal state would be considered positive and all other paths would be considered negative.

State features are defined on the words or other properties comprising a page. For example, state features derived from words are of the form $f^k(i, \mathbf{x}, y_i) = [\![x_i \text{ is "computer" and } y_i = \text{"faculty"}]\!]$. The URL of a page also yields valuable features. For example, a tilde in the URL is strongly associated with a personal homepage and a link with text containing the word "contact" is strongly associated with an address page. We tokenize each URL on delimiters such as '/' and add a feature corresponding to each token.

Transition features capture the soft precedence order amongst labels. One set of transition features are of the form: $f^k(i, \mathbf{x}, y_i, y_{i-1}) = [\![y_i \text{ is "faculty" and } y_{i-1} \text{ is "department"}]\!]$.

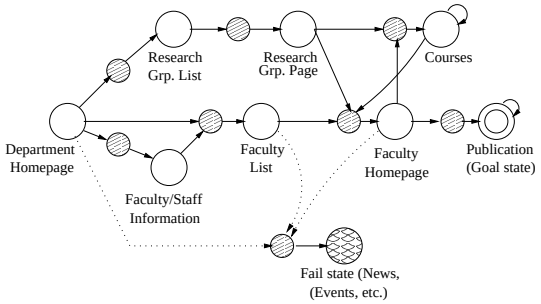


Figure 2: State transition diagram of the model for Publications domain with two states per label

They are independent of x_i and are called **edge features** since they capture dependency amongst adjacent labels. In this model, transition features are also derived from the text surrounding the link leading to the next state. Thus, a transition feature could be of the form $f^k(i, \mathbf{x}, y_i, y_{i-1}) = [\![x_i \text{ is "advisor"– an anchor word, } y_i \text{ is "faculty", and } y_{i-1} \text{ is "student"}]\!]$.

2.4.2 Two states per labels

A second option is to model each label as a dual-state — one for the characteristics of the page itself (**page-states**) and the other for the information around links that lead to such a page (**link-states**). Hence, every path alternates between a page-state and a link-state. In figure 2, we show the state space corresponding to this option for the Publications domain. There are two advantages of this labeling. First, it reduces the sparsity of parameters by making the anchor word features independent of the label of the source page. In practice, it is often found that the anchor text pointing to the same page are highly similar and this is captured by allowing multiple source labels to point to the same link-state of destination label. Second, for the foraging phase, it allows one to easily reason about intermediate probability of a path prefix where only the link is known and the page leading to it has not been fetched.

In this model, the state features of the page-states are the same as in the previous model; the state features of the link-states are derived from the anchor text. The anchor-text transition features of the previous model become state features of the link-state. Thus, the only transition features in this model are the edge features that capture the precedence order between labels.

3. PATH FORAGING

Given the trained sequential model M and a list of starting pages of websites, our goal is to find all paths from the list that lead to the Goal state in M while fetching as few unrelated pages.

The key technical issue in solving this is to be able to score from the prefix of a path already fetched, all the outgoing links with a value that is inversely proportional to the expected work involved in reaching the goal pages. Consider a path prefix of the form $P_1 L_2 P_3 \ldots L_i$ where L_{i-1} is a link to page P_i in the path. We need to find for link L_i a score value that would indicate the desirability of fetching

the page pointed to by L_i. This score is computed in two parts. First in section 3.1, we estimate for each state y, the proximity of the state to the Goal state. We call this the "reward" associated with the state. Then in section 3.2, we show for the link L_i, the probability of its being in state y.

3.1 Reward of a state

We apply techniques from Reinforcement Learning[20] to compute the reward score using the CRF model learnt during path classification phase. Reinforcement Learning is a machine learning paradigm that helps in choosing the optimal action at each state to reach the Goal states. The Goal states are associated with rewards that start to depreciate as the Goal states get farther from the current state. The actions are chosen so as to maximize the cumulative discounted reward.

We apply Reinforcement Learning to compute the probability of a partially-observed sequence to end-up in a Goal state. Since we cannot predict the state sequence that would be followed by the unseen observation subsequence, we cannot compute the actual probability of the sequence ending in a Goal state. Instead, we estimate this probability based on the training data by learning a reward function $\mathcal{R}$ for each state. For each position i of a given sequence $\mathbf{x}$ we estimate the expected proximity to the Goal state from a state y $\mathcal{R}_i^{\mathbf{x}}(y)$ recursively as follows:

$$\mathcal{R}_i^{\mathbf{x}}(y) = \begin{cases} \frac{\sum_{y'} e^{\mathbf{W}\cdot\mathbf{f}(y',y,\mathbf{x},i+1)}\mathcal{R}_{i+1}^{\mathbf{x}}(y')}{\sum_{y'} e^{\mathbf{W}\cdot\mathbf{f}(y',y,\mathbf{x},i+1)}} & 1 \le i < n \\ [\![y = Goal]\!] & i = n \end{cases} \quad (3)$$

When $i = n$, the reward is 1 for the Goal state and 0 for every other label. Otherwise, the values are computed recursively from the proximity of the next state and the probability of transition to the next state from the current state.

We then compute a weighted sum of these positioned reward values to get position independent reward values. The weight are controlled via γ, a discount factor that captures the desirability of preferring states that are closer to the Goal state, as follows:

$$\mathcal{R}^{\mathbf{x}}(y) = \frac{\sum_{k=0}^{n-1} \gamma^k \cdot \mathcal{R}_{n-k}^{\mathbf{x}}(y)}{\sum_{k=0}^{n-1} \gamma^k} \quad (4)$$

where n is the length of the sequence.

The final reward value of a state is computed by averaging over all training sequences $\mathbf{x}_1 \ldots \mathbf{x}_N$ as

$$\mathcal{R}(y) = \frac{\sum_{\ell=1}^{N} \mathcal{R}^{\mathbf{x}_\ell}(y)}{N} \quad (5)$$

3.2 Probability of being in a state

Consider a path prefix of the form $P_1 L_2 P_3 \ldots L_i$ where L_{i-1} is a link to page P_i in the path. We need to find for link L_i, the probability of its being in any one of the link-states. We provide a method for computing this. Let $\alpha_i(y)$ denote the total weight of ending in state y after i states. We thus define $\alpha_i(y)$ as the value of $\sum_{\mathbf{y}' \in \mathbf{y}_{i:y}} e^{\mathbf{W}\cdot\mathbf{F}(\mathbf{x},\mathbf{y}')}$ where $\mathbf{y}_{i:y}$ denotes all label sequences from 1 to i with i-th position labeled y. For $i > 0$, this can be expressed recursively as

$$\alpha_i(y) = \sum_{y' \in \mathcal{Y}} \alpha_{i-1}(y') e^{\mathbf{W}\cdot\mathbf{f}(y,y',\mathbf{x},i)} \quad (6)$$

with the base cases defined as $\alpha_0(y) = 1$.

The probability of L_i being in the link-state y is then $\frac{\alpha_i(y)}{\sum_{y' \in \mathcal{Y}_\mathcal{L}} \alpha_i(y')}$ where $\mathcal{Y}_\mathcal{L}$ denotes the set of link-states.

3.3 Score of a link

Finally, the score of a link L_i after i steps is calculated as the sum of the product of reaching a state y and the static reward at state y.

$$\text{Score}(L_i) = \sum_y \frac{\alpha_i(y)}{\sum_{y' \in \mathcal{Y}_\mathcal{L}} \alpha_i(y')} \mathcal{R}(y) \quad (7)$$

If a link appears in multiple paths, we sum over its score from each path.

3.4 Algorithm to prioritize links

So, to put it all together,

1. During training, for all training instances, compute $\mathcal{R}^{\mathbf{x}}(y)$ for all y (Eq. 4) during the backward pass.
2. Average the $\mathcal{R}(y)$-values computed in step 1 over all training instances (Eq. 5).
3. During testing,
   (a) Maintain a priority queue of links that lead from pages fetched. The links are scored using Eq. 7. Since computation of score requires the $\boldsymbol{\alpha}$-values (Eq. 6), those are also maintained along with the link information.
   (b) In addition to the score and the $\boldsymbol{\alpha}$-values, the $\boldsymbol{\delta}$-values (Eq. 2) used to compute the label are also maintained.
   (c) Initially, the queue contains the URL of seed page with score 0, and the $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$-values are set to 1 and 0 respectively.
4. For each seed URL in priority queue,
   (a) Crawl the highest priority link to fetch the target page P.
   (b) Compute $\boldsymbol{\alpha}$ and $\boldsymbol{\delta}$ for page P using Eqs. 6 and 2 respectively.
   (c) Label the page P with the label of the state that maximizes Eq. 2.
   (d) For every outlink from page P,
      i. calculate $\boldsymbol{\alpha}$-values to compute the score,
      ii. calculate $\boldsymbol{\delta}$-values to label the link,
      iii. enter the link in the priority queue.
   (e) If more URLs to be crawled, go back to step 4a.

4. EXPERIMENTAL RESULTS

Experiments were conducted over two application examples — the Publications example, as explained in section *1.1.2* and the Company Address example, as explained in section *1.1.1*. The experimentation was done in two phases. We first tested the accuracy of the CRF-based sequential classifier in distinguishing between positive and negative paths and segmenting a path. The results were compared with those using other known techniques, such as naïve Bayes and Maximum Entropy classifiers. In the second phase, the trained model was used in foraging mode to fetch relevant pages from the Web. The results were compared with generic focused crawlers.

4.1 Dataset description

Two datasets were generated with labeled sequences for the Publication and the Company Address extraction applications. The `Publications` dataset is used to train the CRF model to recognize paths that lead to the publications pages from department homepages (refer section *1.1.2*), while the `Address` dataset is used to recognize company address pages (refer section *1.1.1*). The datasets were built manually by crawling sample websites and enlisting the sequence of webpages from the entry page to a goal page. Sequences that led to irrelevant pages were identified as negative examples.

The statistics for the two datasets is given in Table 1. The CRF model for the `Publications` dataset was trained on 44 sequences from 7 university domains including domains from IIT Bombay[2], and computer science departments of US universities chosen randomly from an online list[3]. The training dataset included 28 positive and 16 negative sequences and the model was tested on 23 sequences. The test data included some sequences from domains that were not included in the training data. The state diagram with the page-states and the transitions between them is shown in Fig. 1.

Table 1: Description of the datasets

Parameters	Datasets	
	Publications	**Address**
#sites	7	15
#training examples	44	32
#training positives	28	17
#test examples	23	12
#test positives	14	7
#Labels	17	7
#Extracted words	3062	1035
#Features learnt	5834	1763

The key states in the CRF model for the Address domain correspond to company homepage, page on company details ("About Us"), and the address page. The state transition diagram is shown in Fig. 3. The `Address` dataset was trained on 32 sequences out of which 17 sequences were positive, and was tested on 12 sequences. All test sequences were from domains other than those in the training data.

[2]The two domains were `http://www.it.iitb.ac.in/` and `http://www.cse.iitb.ac.in/`

[3]`http://www.clas.ufl.edu/CLAS/american-universities.html`

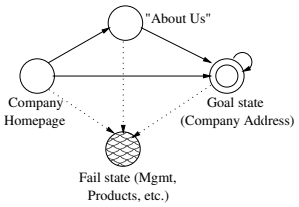


Figure 3: State transition diagram of the model for Company Address domain. Shows only page-states.

4.2 Feature Extraction

When an HTML page is fetched, the page is represented in DOM structure format (`http://www.w3.org/DOM/`) using the Hypertext Parsing suite[11]. The text content from the page is split into tokens on white-space delimiters (space, tab, etc.). The page-state tokens are extracted from the head and body fields of the HTML page, while the link-state tokens are collected from the URL anchor text and the neighbourhood text around the link. To capture the text around the anchor text of a hyperlink, we extracted tokens from a fixed-sized window before and after the link. In our experiments, we kept the window size constant at 10. In addition to these, the words from the relative part of the target URL and the target file extension are also included as tokens. As an illustration, for the URL "`http://www-2.cs.cmu.edu/~svc/papers/view-publications-ckl2004.html`", the tokens extracted from the URL were `~svc`, `papers`, `view`, `publications`, `ckl2004`, and `html`. Since '~' is a key feature that usually distinguishes homepages from departmental pages, we also add '~' as a separate token.

This way, for the publication dataset we had a total of 5834 features for which the CRF learnt corresponding weights. The corresponding number for the address dataset was 1763.

Experimental platform. Our experiments were performed on a two processor Pentium III server running Linux and with 1 GB of RAM. The CRF code was a JAVA implementation of the algorithm described in [25] and is available via free download at `http://crf.sourceforge.net/`.

4.3 Path segmentation and labeling

Our first set of experiments were to understand the per-state labeling accuracy of the CRF model. In this phase, we test whether the model correctly segments and labels path sequences.

4.3.1 Effect of number of page-state tokens

In this task, we vary the number of tokens extracted from the body of the HTML page. Table 2 shows the variation of the precision, recall, and F1 values as we decrease the number of tokens from 100 to 0. The F1-value is the harmonic mean of the precision and recall values.

The first row for each dataset (marked `(A)`) shows the average accuracy over all states, while the second row (marked `(G)`) shows the accuracy values for the Goal states alone. We observe that for the `Publications` dataset, the best performance is obtained on considering 100 tokens per page and the F1 accuracy mostly keeps decreasing as the number of

Table 2: Effect of number of page-state tokens on precision, recall, and F1 values. The rows marked (A) show average accuracy over All states and (G) over Goal states alone.

Dataset		Number of Tokens extracted from HTML page								
		#Page Tokens = 100			#Page Tokens = 10			#Page Tokens = 0		
		Precision	**Recall**	**F1**	**Precision**	**Recall**	**F1**	**Precision**	**Recall**	**F1**
Publication	(A)	88.3	81.4	84.7	87.0	87.7	87.3	72.4	75.4	73.9
Publication	(G)	93.4	96.3	94.8	83.3	92.6	87.7	75.2	82.8	78.8
Address	(A)	56.3	45.0	50.0	81.2	62.5	70.6	80.6	78.6	79.6
Address	(G)	54.6	85.7	66.7	100.0	42.9	60.0	71.4	83.3	76.9

tokens per page is decreased.

For the `Address` dataset, the accuracy increases as the number of page-tokens is reduced. This is because in the `Address` dataset, the tokens extracted from the link and link neighbourhood is sufficient to correctly classify the path sequence. Addition of too many page-tokens reduces the effect of the link-tokens. So, the accuracy drops by about 30%.

4.3.2 Comparison with naïve Bayes and Maximum Entropy classifiers

To emphasize the power of Conditional Random Fields, we compare the results with

- A set of multinomial naïve Bayes classifiers with Laplacian smoothing.
- A Maximum Entropy model, which learns a per-state exponential distribution of conditional probability of labels given the input sequence.

Both the naïve Bayes and the Maximum Entropy models were learnt on the same set of tokens extracted from the current page or link. The naïve Bayes model was learnt with the set of tokens treated as a "bag-of-words" associated with the current state. The Maximum Entropy model, however, was learnt over an exponential feature space of the type $e^{\mathbf{W}\cdot\mathbf{F}(\mathbf{x},\mathbf{y})}$. Since the transition features were deactivated, the Maximum Entropy model was trained, in effect, as a multi-state exponential-model classifier instead of a sequence classifier.

The experimental results are shown in Table 3. The first row for each dataset (marked `(A)`) shows the average accuracy over all states, while the second row (marked `(G)`) shows the accuracy values for the Goal state alone. The third row in both datasets (marked `(B)`) shows the precision, recall, and F1 values for the goal state when trained using a binary classification scenario where all goal pages are labeled *relevant* and all other pages are labeled *irrelevant*. The number of page-tokens extracted was kept constant at 10 for both datasets.

We observe that CRFs perform better than naïve Bayes for both datasets and for both binary and multi-class classification experiments. The naïve Bayes technique gives very high precision value for Goal state, but the recall values are very poor. This is because the naïve Bayes technique labels very few examples as the Goal state, thereby getting away with a large precision value.

For the `Publications` dataset, CRFs also perform about 20% better than the Maximum Entropy models in the multi-class classification. For the binary classification, we found that both models performed at par.

On comparing the CRFs with the Maximum Entropy models for the `Address` dataset, we see that the CRFs label pages with higher precision and recall values; thereby confirming that the path information captured by the transition features helps building a more accurate classification model.

For the CRF method, we find the binary model to be worse than the multi-class model in recognizing the Goal state (comparing methods (G) in the second row and (B) in the third row) in the case of the `Publications` dataset. The reason is that this dataset has lot more useful milestone states than the `Address` dataset which has just one additional state.

4.4 Performance of our system in foraging mode

4.4.1 Publications dataset

The CRF model for the `Publications` dataset was learnt on domains that included websites from Indian and US universities as described in section 4.1. The model was then run in foraging mode, in which the model is left to crawl links based on the scoring measure as described by Eq. 7. The γ parameter in equation 4, required for computation of the per-state reward, was set to 0.9. The experiments were run on the following sites:

- `http://www.it.iitb.ac.in/`, henceforth referred to as the **IT** domain.
- `http://www.cse.iitb.ac.in/`, henceforth referred to as the **CS** domain.
- `http://www.cs.cmu.edu/`, henceforth referred to as the **CMU** domain.
- `http://www.cs.utexas.edu/`, henceforth referred to as the **UTX** domain.

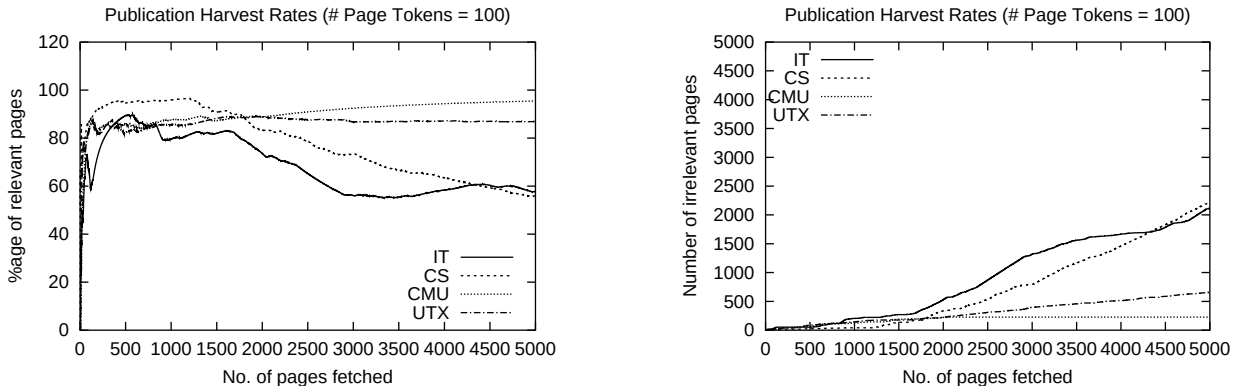
Performance was measured in terms of "harvest rates". **Harvest Rate** is defined as the ratio of relevant pages (goal pages, in our case) found to the total number of pages visited.

Fig. 4(a) shows the performance of our model in the four domains. Our model is able to achieve harvest rates of over 80% after the transients. In the **CMU** and **UTX** domains, where the number of publication pages is higher, the model achieved higher harvest rates. In the **IT** and **CS** domains, the harvest rates reduced after the initial surge, probably because most of the publication pages were fetched in the initial 1500 pages crawled.

The graphs in Fig. 4(b) show the number of irrelevant pages fetched in the total pages crawled. The slope of the

Table 3: Comparison of CRF with naïve Bayes and Maximum Entropy models. The precision, recall, and F1 values shown for (A) average of All states, (G) only Goal state, and (B) goal state when trained using a Binary (relevant/irrelevant) classifier.

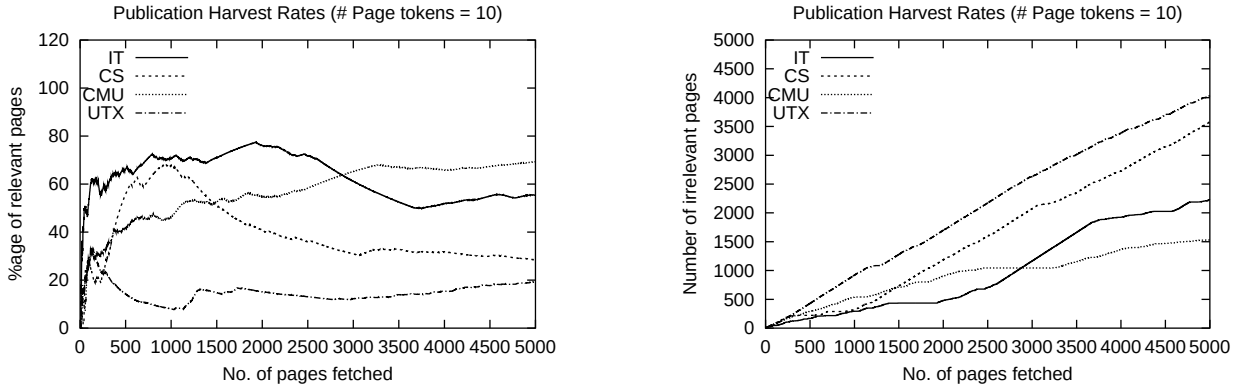
Dataset		naïve Bayes			Maximum Entropy			CRFs		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Publications	(A)	49.6	46.4	48.0	67.2	61.6	64.3	87.0	87.7	87.3
Publications	(G)	100.0	11.1	20.0	81.8	66.7	73.5	83.3	92.6	87.7
Publications	(B)	100.0	18.5	31.3	91.3	77.8	84.0	91.3	77.8	84.0
Address	(A)	29.5	27.5	28.5	51.4	25.0	33.7	81.2	62.5	70.6
Address	(G)	36.4	57.1	44.4	66.7	28.6	40.0	100.0	42.9	60.0
Address	(B)	100.0	20.0	33.3	64.3	60.0	62.1	89.8	48.3	62.8



(a) Hit Rates(%age) on different domains

(b) No. of irrelevant pages fetched

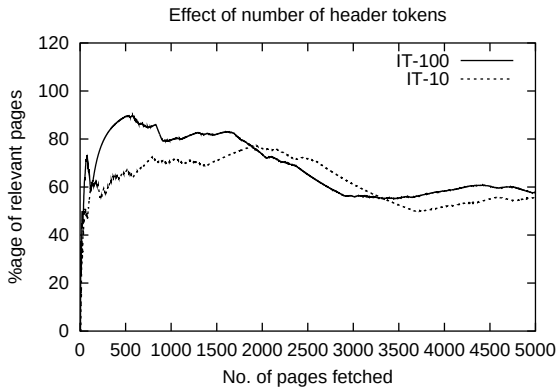
Figure 4: Results for all domains showing relevant page fetch values, with number of page-tokens = 100



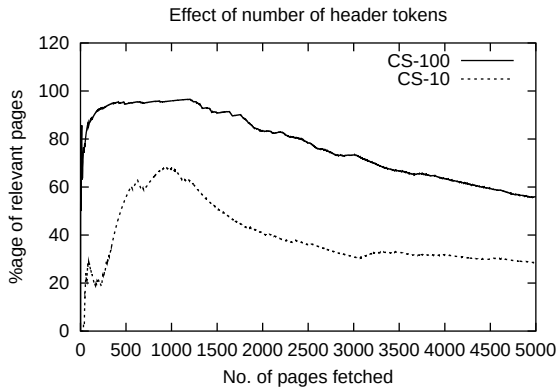
(a) Hit Rates(%age) on different domains

(b) No. of irrelevant pages fetched

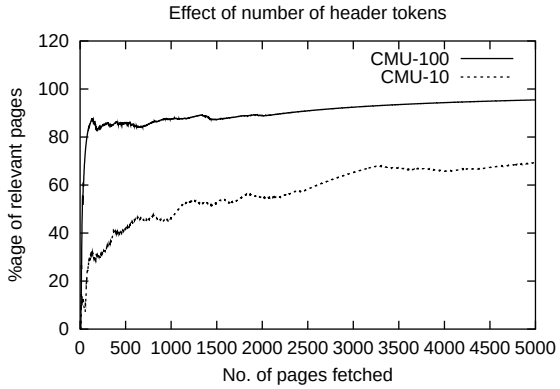
Figure 5: Results for all domains showing relevant page fetch values, with number of page-tokens = 10



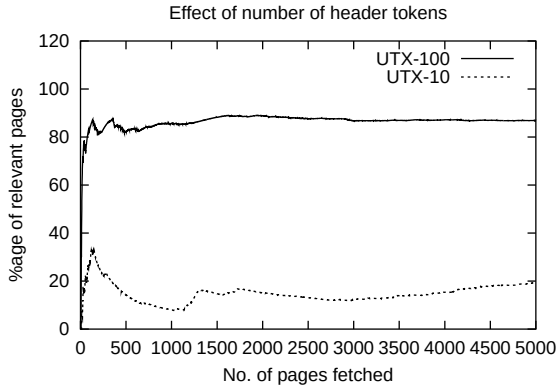
(a) For **IT** domain



(b) For **CS** domain



(c) For **CMU** domain



(d) For **UTX** domain

Figure 6: Effect of the number of tokens on harvest rates. The graph labels show the domain and the number of tokens.

line must be as small as possible (with the worst case slope = 1, i.e. a 45° line). The slope of graphs in our experiment is between 0.12 and 0.44 (6.5° to 24°).

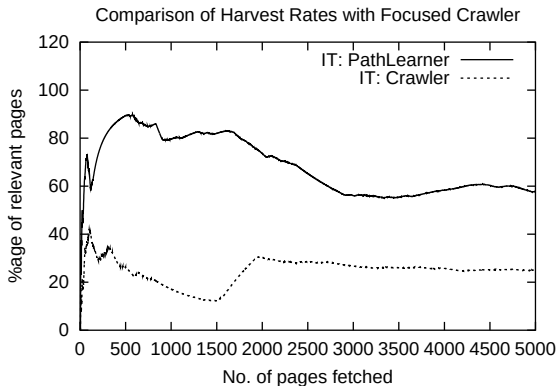
For the sake of comparison, we ran the CRF in foraging mode using a model that was trained on 10 page-tokens. Our experiments on path classification (section *4.3.1*) had shown comparable results between models trained with 10 and 100 page-tokens, though the model with 100 page-tokens had a better precision value. On running the model trained on 10 page-tokens in foraging mode, the harvest rates reduce to 20–70% range. As seen in Fig. 5(b), the slope of graph for the **UTX** is as high as 0.8 (i.e. 38.7°).

The comparison showed that with very few page-tokens, the information on the page, and hence the page type, is not captured sufficiently. A head-to-head comparison to show the effect of number of page-tokens is shown in Fig. 6.

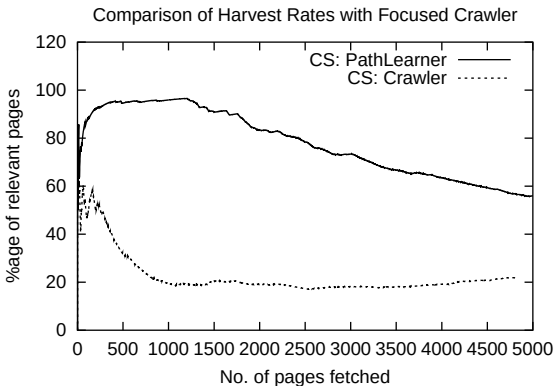
4.4.2 Comparison with Accelerated Focused Crawler

Focused crawlers[4] are designed to crawl all webpages on a topic specified through examples of pages related to the topic. The basic property that such crawlers exploit is that pages on a topic are often linked to each other. Accelerated Focused Crawlers[4] use a naïve Bayes model to classify a crawled page. The crawler consists of two classifiers — a Baseline classifier that trains on the page-tokens, and an apprentice that trains on link-tokens to choose the best hyperlink out of the crawl frontier. The focused crawler decides the most potential outlink by first choosing pages on the crawl frontier that are close enough to the goal page, and then selecting the most probable link from the hyperlinks out of those pages.

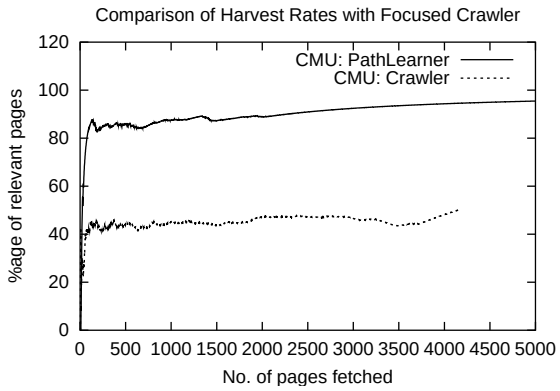
We approximated the Focused Crawler by building a C-way Maximum Entropy classifier on the C labels defined during training. The experimental results in Table 3 already clearly show that the Maximum Entropy classifier is significantly better than the naïve Bayes classifier. Interestingly,



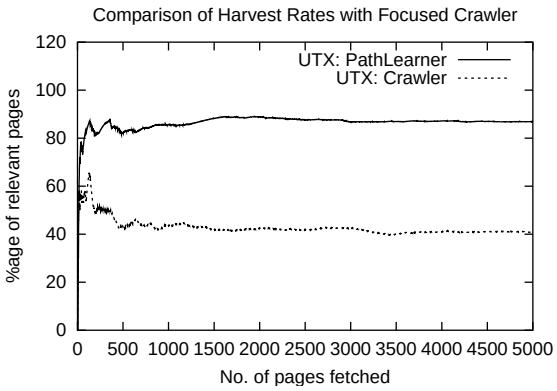
(a) For **IT** domain



(b) For **CS** domain



(c) For **CMU** domain



(d) For **UTX** domain

Figure 7: Comparison with simplified Accelerated Focused Crawler. The graphs labeled `PathLearner` show the performance of our model.

this makes the classifier similar to our CRF model but without the transition features. The set of features used in the focused crawler were identical to the state features of our CRF model to standardize comparison. In this mode, we pick that link for expanding which has the highest probability of being the Goal state.

Fig. 7 shows a comparison on how our model performs against the simplified model of the Accelerated Focused Crawler (AFC). We observe that the performance of our model is significantly better than the AFC model. In all four domains, the relevant pages fetched by the CRF model increases rapidly at the beginning before stabilizing at over 60%, when the AFC model barely reaches 40%.

This shows that using path information helps in fetching relevant pages faster. The scoring measure, which employs discounted rewards using Reinforcement Learning, is able to weigh the links leading to goal page higher than other links.

4.4.3 Address dataset

The foraging experimentation on `Address` dataset differs slightly from the one on the `Publications` dataset.

In the `Publications` dataset, we have multiple goal pages within a website. During the foraging experiment, the model aims at reaching as many goal pages as possible quickly. In effect, the model tries to reach a hub — i.e. a page that links many desired pages directly such that the outlink probability from the page to Goal state is maximum. As the goal pages get crawled and the unfetched goal pages become scanty, the harvest rates tend to decline as the number of crawled pages increase. After an initial burst of fetching a lot of goal pages, the number of goal pages fetched become scarce. This effect was seen prominently in the **IT** and **CS** domains in the `Publications` dataset when the model was run for extended crawls (refer Fig. 4(a)).

In the `Address` dataset, there is only one (or a countable few) goal pages. Hence, following the approach similar to that of the `Publications` dataset would lead to declining harvest rates once the address page is fetched. Hence, we modify the foraging run to stop when a goal page is reached. We proceed with the crawling only when we have a link with a higher score of reaching the Goal state than the current page score.

The experiment was run on 108 domains of company addresses taken randomly from the list of companies available at `http://www.hoovers.com/`. We calculate the average number of pages required to reach the goal page from the homepage.

The average length of path from homepage to goal page was observed to be 3.426, with the median and mode value being 2. This agrees with the usual practice of having a "Contact Us" link on the company homepage that leads in one link access to the contact address.

5. RELATED WORK

The work presented here is related to work in web wrapper induction, focused crawling, information extraction, and sequential learning.

A popular subproblem in the domain of web wrapper induction is extracting structured fields from HTML documents. These do shallow information extraction based on syntactic cues present as HTML tags. Except for a few initial systems based on manual approaches, most of the recent ones follow the same learn-from-example approach. Example systems of these kind are WEIN[13], SoftMealy[10], Stalker[21, 3], W4F[2] and XWrap[16].

One of the earliest projects that perform extraction of structured information from multi-page sources like a website is WebKB[6]. They describe similar learning tasks of recognizing relations by traversing paths through hyperlinks. However, their approach is based on generative classifiers (like naïve Bayes) for recognizing correct hits coupled with first order rules (like FOIL[12]) for finding the right page. We have already shown the inferiority of the naïve Bayes classifier compared to conditional classifiers even for the underlying path classification problem.

Rennie and McCallum[23] study a similar path learning problem by using Reinforcement Learning. They learn a mapping function from the text in the neighbourhood of the link to a Q-value determined using Reinforcement Learning. They then bin the Q-values based on the minimum number of links required to reach the current page from the seed page. However, they make an assumption that states are independent of which on-topic documents have been visited. In other words, they collapse all states into one. By doing this, an occurrence of a word is considered independent of the context of its usage. The Q function simply becomes a mapping from a "bag-of-words" to a scalar value. In our model, we try to preserve the context of usage of keywords, by having different states to represent different page-types. This facilitates in having different mappings for a similar set of keywords depending on the state. The availability of multiple states also makes the model more robust for unseen data. Also as we have shown in Table 3, CRF outperforms the naïve Bayes model that follows the "bag-of-words" technique.

We have already compared our work with focused crawlers in section *4.4.2*. Another related paper on focused crawling is by Diligenti, et al. [8]. This paper suggests creation of a graph in which the topic-relevant documents are at level 0, all pages linking to these are at level 1, and so on. All documents marked by the same level are clubbed together as being of the same class. The distance from the goal page is used as a direct measure in determining the class. In this respect, the idea is similar to Q-value binning in [23]. However, this does not take into account the case in which multiple pages of similar kind may have to be passed before transiting to a page of another kind. Graphical models such as HMMs and CRFs are able to include such cases as well, thereby making the model stronger.

6. CONCLUSION

We show that Conditional Random Fields provide an elegant, unified, and high-performance method of solving the information foraging task from large domain-specific websites. The proposed model performs significantly better than a generic focused crawler and is easy to train and deploy.

We are continuing further work on making the system work in a semi-supervised setting where the user does not need to specify milestone states. Future work includes integrating it with a finer grained information extraction engine that can find more specific information like the exact address from the goal page.

7. REFERENCES

[1] Shlomo Argamon-Engelson and Ido Dagan. Committee-based sample selection for probabilistic classifiers. *Journal of Artificial Intelligence Research*, 11:335–360, 1999.

[2] Arnaud Sahuguet and Fabien Azavant. Building light-weight wrappers for legacy Web data-sources using W4F. In *International Conference on Very Large Databases (VLDB)*, 1999.

[3] Greg Barish, Yi-Shin Chen, Dan DiPasquo, Craig A. Knoblock, Steve Minton, Ion Muslea, and Cyrus Shahabi. Theaterloc: Using information integration technology to rapidly build virtual applications. In *Intl. Conf. on Data Engineering ICDE*, pages 681–682, 2000.

[4] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *WWW, Hawaii*. ACM, May 2002.

[5] Ed H. Chi, Peter Pirolli, Kim Chen, and James Pitkow. Using information scent to model user information needs and actions and the Web. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 490–497. ACM Press, 2001.

[6] Mark Craven, Dan DiPasquo, Dayne Freitag, Andrew K. McCallum, Tom M. Mitchell, Kamal Nigam, and Seán Slattery. Learning to construct Knowledge Bases from the World Wide Web. *Artificial Intelligence*, 118(1/2):69–113, 2000.

[7] T. G. Dietterich. Machine learning for sequential data: A review. In T. Caelli, editor, *Structural, Syntactic, and Statistical Pattern Recognition; Lecture Notes in Computer Science*, volume 2396, pages 15–30. Springer-Verlag, 2002.

[8] Michelangelo Diligenti, Frans Coetzee, Steve Lawrence, C. Lee Giles, and Marco Gori. Focused crawling using Context graphs. In *26th International Conference on Very Large Databases, VLDB 2000*, pages 527–534, Cairo, Egypt, 10–14 September 2000.

[9] Dayne Freitag and Andrew Kachites McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of the AAAI-99 Workshop on Machine Learning for Information Extraction*, pages 31–36, 1999.

[10] C.-N. Hsu and M.-T. Dung. Generating finite-state transducers for semistructured data extraction from the web. *Information Systems Special Issue on Semistructured Data*, 23(8), 1998.

[11] Ravindra R. Jaju. Robust html to dom conversion and applications. Master's thesis, Kanwal Rekhi School of Information Technology, Indian Institute of Technology Bombay, 2003.

[12] J.R.Quinlan. Learning logical definitions from relations. *Machine Learning*, 5:239–266, 1990.

[13] N. Kushmerick, D.S. Weld, and R. Doorenbos. Wrapper induction for information extraction. In *Proceedings of IJCAI*, 1997.

[14] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data. In *Proceedings of the 18th International Conference on Machine Learning (ICML-2001)*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[15] D. C. Liu and J. Nocedal. On the limited memory bfgs method for large-scale optimization. *Mathematic Programming*, 45:503–528, 1989.

[16] L. Liu, C. Pu, and W. Han. Xwrap: An xml-enabled wrapper construction system for web information sources. In *International Conference on Data Engineering (ICDE)*, pages 611–621, 2000.

[17] William H. Majoros. Unveil: An HMM-based Genefinder for Eukaryotic DNA, 2003. White Paper.

[18] Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *Proceedings of The Sixth Conference on Natural Language Learning (CoNLL-2002)*, pages 49–55, 2002.

[19] Andrew McCallum and Wei Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In *Proceedings of The Seventh Conference on Natural Language Learning (CoNLL-2003)*, Edmonton, Canada, 2003.

[20] Tom Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[21] Ion Muslea, Steve Minton, and Craig A. Knoblock. A hierarchical approach to wrapper induction. In *Proceedings of the Third International Conference on Autonomous Agents*, Seattle, WA, 1999.

[22] Lawrence R. Rabiner. A tutorial on Hidden Markov Models and selected applications in speech recognition. In *Proceedings of the IEEE*, volume 77(2), pages 257–286, February 1989.

[23] Jason Rennie and Andrew Kachites McCallum. Using reinforcement learning to spider the Web efficiently. In Ivan Bratko and Saso Dzeroski, editors, *Proceedings of ICML-99, 16th International Conference on Machine Learning*, pages 335–343, Bled, SL, 1999. Morgan Kaufmann Publishers, San Francisco, US.

[24] Sunita Sarawagi. JAVA implementation of CRF available for research purposes at http://crf.sourceforge.net/.

[25] Fei Sha and Fernando Pereira. Shallow parsing with conditional random fields. In *Proceedings of HLT-NAACL 2003*, pages 213–220. Association for Computational Linguistics, 2003.

[26] Hanna M. Wallach. Efficient training of conditional random fields. Master's thesis, School of Cognitive Science, Division of Informatics, University of Edinburgh, 2002.

[27] Online resources of lists:

1. Listing of company names: http://www.hoovers.com
2. Compilation of KD product companies: http://www.kdnuggets.com/companies/products.html
3. Compilation of links to american universities: http://www.clas.ufl.edu/CLAS/american-universities.html

.