Searching for Hidden-Web Databases

Luciano Barbosa
University of Utah
lab@sci.utah.edu

Juliana Freire
University of Utah
juliana@cs.utah.edu

ABSTRACT

Recently, there has been increased interest in the retrieval and integration of *hidden-Web data* with a view to leverage high-quality information available in online databases. Although previous works have addressed many aspects of the actual integration, including matching form schemata and automatically filling out forms, the problem of *locating relevant data sources* has been largely overlooked. Given the dynamic nature of the Web, where data sources are constantly changing, it is crucial to automatically discover these resources. However, considering the number of documents on the Web (Google already indexes over 8 billion documents), automatically finding tens, hundreds or even thousands of forms that are relevant to the integration task is really like looking for a few needles in a haystack. Besides, since the vocabulary and structure of forms for a given domain are unknown until the forms are actually found, it is hard to define exactly what to look for.

We propose a new crawling strategy to automatically locate hidden-Web databases which aims to achieve a balance between the two conflicting requirements of this problem: the need to perform a broad search while at the same time avoiding the need to crawl a large number of irrelevant pages. The proposed strategy does that by focusing the crawl on a given topic; by judiciously choosing links to follow within a topic that are more likely to lead to pages that contain forms; and by employing appropriate stopping criteria. We describe the algorithms underlying this strategy and an experimental evaluation which shows that our approach is both effective and efficient, leading to larger numbers of forms retrieved as a function of the number of pages visited than other crawlers.

Keywords

hidden Web, large scale information integration, focused crawler

1. INTRODUCTION

Recent studies estimate the hidden Web contains anywhere between 7,500 and 91,850 terabytes of information [2, 14]. As the volume of information in the hidden Web grows, there is increased interest in techniques and tools that allow users and applications to leverage this information. In this paper, we address a crucial problem that has been largely overlooked in the literature: how to efficiently locate the searchable forms that serve as the entry points for the hidden Web. Having these entry points is a necessary condition to perform several of the hidden-Web data retrieval and integration tasks. The searchable forms can be used as the starting point for deep crawls [18, 1] and for techniques that probe these databases to derive source descriptions [11]; and in form matching they can serve as inputs to algorithms that find correspondences among attributes of different forms [12, 24, 13].

Several factors contribute to making this problem particularly challenging. The Web is constantly changing – new sources are added, and old sources are removed and modified. A scalable solution, suitable for a large-scale integration or deep-crawling task, must *automatically find the hidden-Web sources*. In addition, even for a well-defined domain (*e.g.,* books for sale), it is hard to specify a schema (or schemata) that accurately describes the relevant forms. Since there is a wide variation both in the structure and vocabulary of forms, if the definition is too strict, we risk missing relevant forms that use a slightly different schema vocabulary or structure. And in order to obtain a general definition that covers the domain well, it is necessary to have the forms to discover the correspondences among attributes [12, 24, 13]. Thus, we *need to perform a broad search*. But forms are very sparsely distributed. A recent study estimates that there are 307,000 deep Web sites, and an average of 4.2 query interfaces per deep Web site [7]. Thus, searching for tens, hundreds or even thousands of forms, that are relevant to the integration task among billions of Web pages is really like looking for a few needles in a haystack. In order to be practical, the search process must be *efficient* and avoid visiting large unproductive portions of the Web.

A possible approach to address this problem would be to perform a full crawl of the Web, but this would be highly inefficient. An exhaustive crawl can take weeks; and as the ratio of forms to Web pages is small, this would lead to unnecessarily crawling too many pages. Another alternative would be to use a focused crawler. Focused crawlers try to retrieve only a subset of the pages on the Web that are relevant to a particular topic. They have been shown to lead to better quality indexes and to substantially improved crawling efficiency than exhaustive crawlers [6, 19, 5, 10, 21]. However, existing strategies fail to meet the requirements of our problem.

Crawlers that focus the search based solely on the contents of the retrieved pages, such as the best-first crawler of [6], are not effective. Since forms are sparsely distributed even within a restricted domain, the number of forms retrieved per total of visited pages can be very low (see Section 4). Renee and McCallum [19] used reinforcement learning to build a focused crawler that is effective for sparse concepts. Instead of just considering the content of individual pages and crawling through pages that give immediate benefit, they train a learner with *features collected from paths leading to a page*. They do this by repeatedly crawling sample sites to build the connectivity graphs with the optimized paths to the target documents. However, this approach was designed for tasks which,

Eighth International Workshop on the Web and Databases (WebDB 2005), June 16-17, 2005, Baltimore, Maryland.

unlike searching for hidden-Web databases, consist of well-defined search problems within a well-defined set of Web sites. For example, locating the name of the CEO in a given company site; and locating research papers available in the sites of computer science departments [19].

We propose a new crawling strategy that combines ideas from these two approaches. Similar to [6], we use a page classifier to guide the crawler and focus the search on pages that belong to a specific topic. But in order to further focus the search, like in [19], our crawler learns to identify promising links, including links whose benefit may not be immediate – in our case, a link classifier selects links that are likely to reach pages that contain forms (in one or more steps). However, instead of explicitly building the Web graph through repeated crawls of selected sites (which can be prohibitively expensive for a broad search), we rely on the backward crawling facilities provided by search engines in order to approximate this graph [3, 10]. In addition, based on form-specific characteristics, we introduce new stopping criteria that are very effective in guiding the crawler to avoid excessive speculative work in a single site.

Our experimental results over three distinct domains show that, even using an approximated connectivity graph, our crawler is more efficient (up to an order of magnitude) than a set of representative crawlers. Not only it is able to perform a broad search and retrieve a large number of searchable forms, but, for a fixed number of visited pages, it also retrieves a significantly larger number of forms than other crawlers. The experiments also show an added benefit of combining a focused crawl with the identification of links that lead to pages that contain forms: focusing the crawl on a topic helps improve effectiveness of the link classifier, since the features that are learned for links are often specific to a topic/domain.

The outline of the paper is as follows. We review related work in Section 2. In Section 3, we describe the architecture of our crawler, its implementation and underlying algorithms. An experimental evaluation and comparison against other approaches is given in Section 4. We conclude in Section 5, where we discuss directions for future work.

2. RELATED WORK

In what follows, we give an overview of previous works on focused crawling. We also briefly review approaches that address different aspects of retrieval and integration of hidden-Web data.

Focused Crawling. The goal of a focused crawler is to select links that lead to documents of interest, while avoiding links that lead to off-topic regions. Several techniques have been proposed to focus web crawls (see *e.g.,* [5, 6, 10, 19, 21]). In [6], Chakrabarti et al describe a best-first focused crawler (called *baseline* in the remainder of this paper) which uses a page classifier to guide the search. The classifier learns to classify pages as belonging to topics in a taxonomy (*e.g.,* dmoz.org). Unlike an exhaustive crawler which follows each link in a page in a breadth first manner, this focused crawler gives priority to links that belong to pages classified as relevant. Although this best-first strategy is effective, it can lead to suboptimal harvest rates, since even in domains that are not very narrow, the number of links that are irrelevant to the topic can be very high. An improvement to the baseline strategy was proposed in [5], where instead of following all links in relevant pages, the crawler used an additional classifier, the apprentice, to select the most promising links in a relevant page. The baseline classifier captures the user's specification of the topic and functions as a *critic* of the apprentice, by giving feedback about its choices. The apprentice, using this feedback, learns the features of good links and is responsible for prioritizing the links in the crawling frontier. Although our approach also attempts to estimate the benefit of following a particular link, there are two key differences. Whereas the apprentice only considers links that give immediate benefit, our link classifier learns to predict the distance between a link and a target page, and thus, our crawler considers links that may be multiple steps away from a target page. Besides, the goal of our link classifier is complementary to that of [5] – we want to learn which links lead to pages that contain searchable forms, whereas the goal of [5] to avoid off-topic pages. In fact, our approach would also benefit from such an apprentice, since it would reduce the number of off-topic pages retrieved and improve the overall crawling efficiency. Integrating the apprentice in our framework is a direction we plan to pursue in future work.

One issue with focused crawlers is that they may miss relevant pages by only crawling pages that are expected to give immediate benefit. In order to address this limitation, strategies have been proposed that train a learner with features collected from *paths leading to a page*, as opposed to just considering a page's contents [10, 19]. Rennie and McCallum [19] use reinforcement to train a classifier to evaluate the benefit of following a particular link. Their classifier learns features of links which include words in the title and body of the document where the link is located, and words in the URL, anchor and text in the neighborhood of the link. Given a link *(u,v)*, the classifier returns an estimate of the number of relevant pages that can be reached by following *(u,v)*. Diligenti et al [10] also collect paths to relevant pages. But their classifier estimates the distance from a page u to some relevant page w; it does not distinguish among the links in u – if the classifier estimates that a page u has high benefit, all pages v directly reachable from u are retrieved. Similar to [19], we estimate the benefit of individual links and select the ones that are more likely to reach pages that contain forms. However, in order to train our classifier, instead of explicitly building the Web graph through an exhaustive crawl of selected sites, we use the same optimization applied in [10] to build context graphs, *i.e.,* we rely on the backward crawling facilities provided by search engines in order to approximate the Web connectivity graph.

Retrieving and Integrating Hidden-Web Data. MetaQuerier [8] is a system that enables large-scale integration of hidden-Web data. It consists of several components that address different aspects of the integration. One of these components is a crawler for locating online databases, the *Database Crawler*. Unlike our approach, the Database Crawler neither focuses the search on a topic, nor does it attempt to select the most promising links to follow. Instead, it uses as seeds for the crawl the IP addresses of valid Web servers; then, from the root pages of these servers, it crawls up to a fixed depth using a breadth-first search. Their design choice is based on the observation that searchable forms are often close to the root page of the site [7]. As we discuss in Section 3, our crawler prioritizes links that belong to pages close to the root of a site. However, we also show that just limiting the depth of a breadth-first search leads to low crawling efficiency (see Section 4).

Raghavan and Garcia-Molina [18] proposed HiWe, a task-specific hidden-Web crawler. The key problem they addressed was how to automatically fill out structured Web forms. Although this pioneering work automates deep crawling to a great extent, it still requires substantial human input to construct the label value set table. In [1], we proposed a completely automated strategy to crawl through (simpler) unstructured forms (*i.e.,* keyword-based interfaces). Both crawlers can benefit from our system and use the returned form pages as the starting point for deep crawls.

To further automate the process crawling through structured forms,

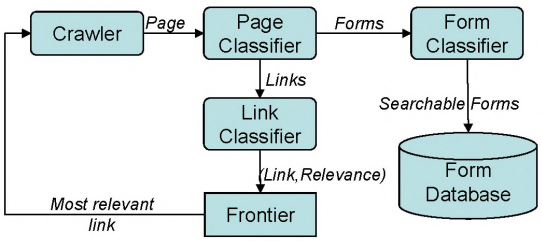

Figure 1: Form Crawler Architecture.

a hidden-Web crawler must *understand* the form interfaces so that it can generate meaningful submissions. Several techniques have been proposed that improve form understanding by finding matches among attributes of distinct forms (see *e.g.,* [12, 13, 24]). The forms we find can serve as inputs to these techniques.

Finally, it is worth pointing out that there are directories specialized on hidden-Web sources, *e.g.,* [4, 17, 20]. Hidden-Web directories organize pointers to online databases in a searchable topic hierarchy. Chang et al [7] note that these directories cover a small percentage of the hidden-Web databases; and they posit this low coverage is due to their "apparent manual classification". Being focused on a topic makes our crawler naturally suitable for automatically building a hidden-Web directory.

3. FORM-FOCUSED CRAWLER

To deal with the sparse distribution of forms on the Web, our Form Crawler avoids crawling through unproductive paths by: limiting the search to a particular topic; learning features of links and paths that lead to pages that contain searchable forms; and employing appropriate stopping criteria. The architecture of the Form Crawler is depicted in Figure 1.

The crawler uses two classifiers to guide its search: the page and the link classifiers. A third classifier, the form classifier, is used to filter out useless forms. The *page classifier* is trained to classify pages as belonging to topics in a taxonomy (*e.g.,* arts, movies, jobs in Dmoz). It uses the same strategy as the best-first crawler of [6]. Once the crawler retrieves a page P, if P is classified as being on-topic, forms and links are extracted from it. A form is added to the *Form Database* if the *form classifier* decides it is a searchable form, and if it is not already present in the Form Database.[1] The *link classifier* is trained to identify links that are likely to lead to pages that contain searchable form interfaces in one or more steps. It examines links extracted from on-topic pages and adds to the crawling frontier in the order of their importance. In the remainder of this section we describe the core elements of the system in detail.

3.1 Link Classifier

Since forms are sparsely distributed, by selecting only links that bring immediate return (*i.e.,* links that directly point to pages containing searchable forms), the crawler may miss "good" target pages that can only be reached with additional steps. Thus, the link classifier aims to identify links that may bring *delayed benefit*, *i.e.,* links that *eventually* lead to pages that contain forms. It learns the following features of links: anchor, URL, and text in the proximity of the URL; and assigns a score to a link which corresponds to the distance between the link and a relevant page that is reachable from that link.

Learning Distance through Backward Crawling. In order to learn the features of "good" paths, the link classifier needs examples of paths that lead to pages that contain searchable forms. These examples can be obtained from the connectivity graphs for a set of representative sites. Note that to build this graph, it may be necessary to perform exhaustive crawls over the sites. While this is possible for a small set of sites [19], the task would be extraordinarily expensive and time-consuming to apply in a large-scale crawling task that may involve thousands of sites.

Instead of building the exact connectivity graph, we get an approximation of this graph by performing a backward crawl using Google's "link:" facility, which returns pages that point to a given document [3, 10]. The backward crawl consists of a breadth-first search starting from pages that contain searchable forms. Each level $l+1$ is constructed by finding all documents that point to the documents in level l. The resulting graph is an approximation because: Google does not provide complete connectivity information; and since the number of backlinks can be large, we select only a subset of these backlinks. Nonetheless, this approximation is sufficient to train the link classifier. As we show in Section 4, using multiple backward crawl levels leads to substantial gains in the form harvest rates.

For each level of the backward crawl, we extract the features of the links in pages that belong to that level. The classifier then learns the distance between a given link (from its associated features) and the target page which contains a searchable form. Intuitively, a link that matches the features of level 1 is likely to point to a page that contains a form; and a link that matches the features of level l is likely l steps away from a page that contains a form.

Feature Space Construction and Focused Crawl. The effectiveness of the link classifier is highly dependent on the features it considers. We experimented with different sets of features, as well as with different ways of extracting them. Due to space limitations, we discuss only the best of the strategies we examined.

For each level l in the backward crawl, we extract the words in the neighborhood of the links in l. We consider three *contexts*: URL, anchor, and text around the link. Since the number of extracted features tends to be large (and most of them have very low frequency), we remove stop-words and stem the remaining words. Then, for each context, we select only words with frequency larger than a fixed threshold. Note that features are associated with a context. For example, if the word "search" appears in both in the URL and in the anchor text of a link, it is added as a feature in both contexts.

Table 1 shows an excerpt of the feature space we constructed for the *jobs* domain; for each context, it shows the most common words and their frequencies. For illustrative purposes, we also show the common words in page title and text of the page where the link is located. Note that:

- Link features contain words that are clearly associated with the domain as well as with searchable forms: common words include "search", "career" and "job". We have observed similar behavior in other domains we explored, for example, for *cars*, common words in the anchor included "search", "used" and "car";
- The document text is a very good indicator of the relevance of a page. For example, words such as "job", "search" and "career" have very high frequencies in the document text in all levels;
- As a link gets farther from the target page, the frequency of clearly related words decreases. For example, whereas the anchor frequency of the word "job" in level 0 is 39, it goes down to 11 in level 2. And although the number of words that are apparently related to topic decreases with the distance, many of the words in the higher levels are still related to the topic.

[1] We check for duplicates because many Web sites have the same form interface in multiple pages.

level/field	URL	Anchor	Around the link	Title of page	Text of page	Number of pages
1	**job 111** **search 38** **career 30** opm 10 htdocs 10 roberthalf 10 accountemps 10	**job 39** **search 22** ent 13 **advanced 12** **career 7** width 6 popup 6	**job 66** **search 49** **career 38** **work 25** home 16 keyword 16 help 15	job 77 career 39 work 25 search 23 staffing 15 results 14 accounting 13	job 186 search 71 service 42 new 40 career 35 work 34 site 27	187
2	**job 40** classified 29 news 18 annual 16 links 13 topics 12 default 12 ivillage 12	**job 30** **career 14** today 10 ticket 10 corporate 10 big 8 list 8 find 6	**job 33** home 20 ticket 20 **career 18** program 16 sales 11 sports 11 search 11	job 46 career 28 employment 16 find 13 work 13 search 13 merchandise 13 los 10	job 103 search 57 new 36 career 35 home 32 site 32 resume 26 service 22	212
3	ivillage 18 cosmopolitan 17 ctnow 14 state 10 archive 10 hc-advertise 10 **job 9** poac 9	**job 11** advertise 8 web 5 oak 5 fight 5 **career 5** against 5 military 5	**job 21** new 17 online 11 **career 11** contact 10 web 9 real 9 home 9	job 17 ctnow 8 service 8 links 7 county 7 career 7 employment 7 work 6	font 37 job 33 service 24 cosmo 20 new 19 career 19 color 16 search 16	137

Table 1: Excerpt of feature space for *jobs* domain. This table shows the most frequent words in each context for 3 levels of the backward crawl, as well as the total number of pages in examined in each level. The selected features used for the different contexts in the link classifier are shown in bold.

These observations reinforce our decision to combine a focused crawler, that takes the page contents into account, with a mechanism that allows the crawler to select links that have delayed benefit. The alternative of using a traditional breadth-first (non-focused) crawler in conjunction with our link classifier would not be effective. Although such a crawler might succeed in retrieving forms reachable from links with generic features (*e.g.,* "search"), it is likely to miss links whose features are domain-dependent (*e.g.,* "used", "car") if the frequencies of these features in the feature table are below the fixed threshold.

In addition to determining the distance of a particular link in relation to the target, *i.e.,* its category, it is also interesting to obtain the probabilistic class membership of this link in the category. This enables the crawler to prioritize links with higher probability of belonging to a given class. For this reason, we chose a naïve Bayes classifier [16] to classify the links. It is worthy of note that other crawlers have used this type of classifier to estimate link relevance [5, 19].

3.2 Page Classifier

We used Rainbow [15], a freely-available naïve Bayes classifier, to build our page classifier. In the Form Crawler, Rainbow is trained with samples obtained in the topic taxonomy of the Dmoz Directory (dmoz.org) – similar to what is done in other focused crawlers [5, 6]. When the crawler retrieves a page P, the page classifier analyzes the page and assigns to it score which reflects the probability that P belongs to the focus topic. If this probability is greater than a certain threshold (0.5 in our case), the crawler regards the page as relevant.

3.3 Form Classifier

Since our goal is to find hidden-Web databases, we need to filter out non-searchable forms, *e.g.,* forms for login, discussion groups interfaces, mailing list subscriptions, purchase forms, Web-based email forms. The form classifier is a general (domain-independent) classifier that uses a decision tree to determine whether a form is searchable or not.

The decision tree was constructed as follows. For positive examples we extracted 216 searchable forms from the UIUC repository [22], and we manually gathered 259 non-searchable forms for the negative examples. For each form in the sample set, we obtained the following features: number of hidden tags; number of checkboxes; number of radio tags; number of file inputs; number of submit tags; number of image inputs; number of buttons; number of resets; number of password tags; number of textboxes; number of items in selects; sum of text sizes in textboxes; submission method (post or get); and the presence of the string "search" within the form tag.

Algorithm	Error test rate
C4.5	8.02%
Support Vector Machine	14.19%
Naive Bayes	10.49%
MultiLayer Perceptron	9.87%

Table 2: Test error rates for different learning algorithms.

We performed the learning task using two thirds of this corpus, and the remaining one third was used for testing. We selected decision trees (the C4.5 classifier) because it had the lowest error rate among the different learning algorithms we evaluated [23]. The error test rates are shown in Table 2.

Cope et al [9] also used a decision tree to classify searchable and non-searchable forms. Their strategy considers over 550 features, whereas we use a much smaller number of features (only 14); and their best error rate is 15%, almost twice the error rate of our form classifier.

3.4 Crawling

The search frontier consists of N queues, where N is the number of levels used by the link classifier; the i-th queue is associated to the i-th level. The crawler prioritizes links that are closer to the target pages, *i.e.,* links that are placed in the queues corresponding to the lowest levels. Within a queue, links are ordered by the likelihood of belonging to the respective level. However, links that belong to pages close to the root of a Web site are given higher priority in the queue. Our decision to prioritize such links comes from the observation that forms often occur close to the main pages of

Web sites [7]. Note, however, that "just" prioritizing these pages is not enough – as we discuss in Section 4, a strategy that simply fixes the search depth is not effective.

Before the crawl starts, the seed links are placed in queue 1. At each crawl step, the crawler gets the most relevant link in the queues, *i.e.,* it pops the link with the highest relevance score from the first non-empty queue. If the page it downloads belongs to the domain, its links are classified by link classifier and added to the *persistent frontier*. When the queues in the crawling frontier become empty, the crawler loads a subset of the queues in the persistent frontier (the most relevant links are given priority). By keeping the persistent frontier separate, we ensure some fairness – all links in the crawling frontier will eventually be followed.

Stopping Criteria. Due to the sparseness of searchable forms, it is important for the Form Crawler to determine when to stop crawling a given site to avoid unproductive searches. The Form Crawler uses two stopping criteria: 1) the crawler leaves a site if it retrieves a pre-defined number of distinct forms; or 2) if it visits the maximum number of pages on that site. The intuition behind the first criterion is that there are few searchable forms in a hidden-Web site. Chang et al [7] observed that deep Web sites contain a small number of query interfaces. They estimate that, on average, a deep Web site has 4.2 query interfaces. Thus, after the crawler finds these forms, it can stop since it is unlikely to find additional forms. Since the Form Crawler performs a broad search, it visits many sites that may contain fewer than 4.2 forms, and sites that do not contain searchable forms. The second criterion ensures that the crawler will not waste resources in such sites. As we discuss below, these stopping criteria are key to achieving a high crawling efficiency.

4. EXPERIMENTAL EVALUATION

The key distinguishing feature of the Form Crawler is that it performs broad crawls to locate forms which are sparsely distributed over the Web. In our experiments, we compare the efficiency of our Form Crawler against that of three other crawlers:

- *Baseline*, a variation of the best-first crawler [6]. The page classifier guides the search: the crawler follows all links that belong to a page classified as being on-topic;
- *Fixed depth* follows the strategy adopted by the Database Crawler [8]. It performs a breadth-first search starting from the root page of a site up to a fixed depth. In our experiments, we set the depth to 3, since according to [7], most query interfaces (91.6%) appear within this depth;
- *Baseline SC* is an extension of the baseline crawler which adopts the stopping criteria described in Section 3.4.

In order to verify the effectiveness of using the distance between a link and a relevant target page as a predictor of link importance, we used different configurations for our Form Crawler:

- *Form Crawler with 1 level*, which corresponds to considering only links that give immediate benefit, *i.e.,* which lead to a form page in a single step;
- *Form Crawler with multiple levels*, which besides links that give immediate benefit also considers links that are multiple steps away from a target page. We ran experiments using from 2 to 4 levels; since the improvements obtained from level 4 are small, we only show the results for configurations with 2 and 3 levels.

We ran these crawlers over three distinct domains: jobs, cars and books. Seed pages for the actual crawl were obtained from the categories in the Google directory that correspond to these domains.

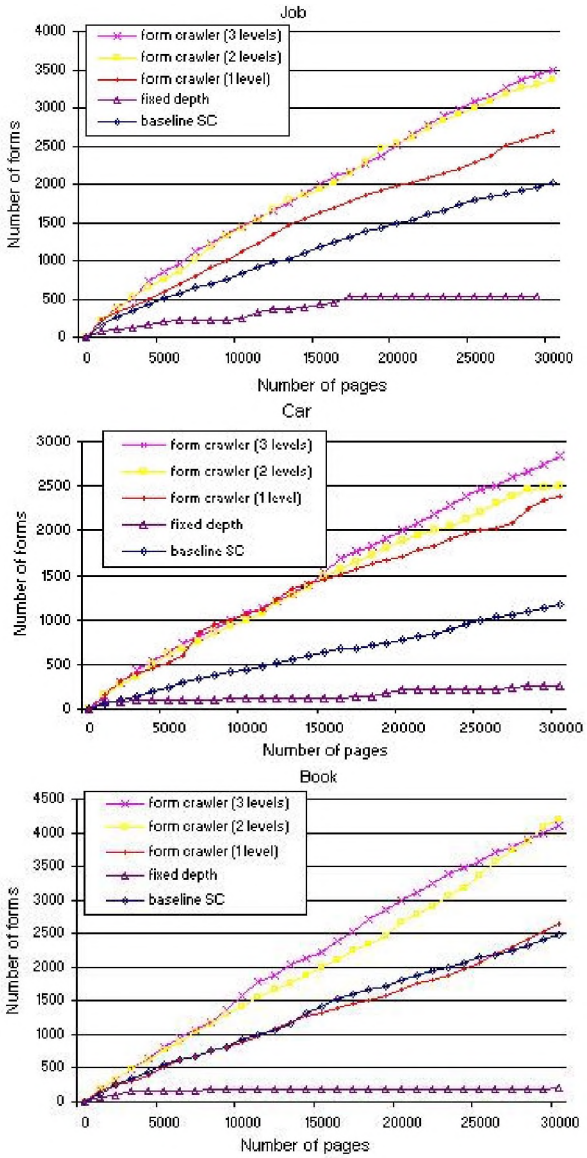

Figure 2: Performance of different crawlers for 3 domains.

For each domain, we created instances of the link and page classifiers. In order to train the link classifier, we obtained a sample of URLs of pages that contain forms (level 1) from the UIUC repository [22], and from these links we performed a backward crawl up to level 4. For the page and form classifiers we followed the procedures described in Sections 3.2 and 3.3, respectively.

An accepted measure for the effectiveness of a focused crawler is the amount of *useful work* it performs. For our crawler, this corresponds to the number of *distinct* relevant forms it retrieves as a function of the number of pages visited. Recall that the relevance of a form is determined by the form classifier (Section 3.3). Figure 2 shows the performance of the different crawlers we considered for each domain. The multi-level Form Crawler performed uniformly better than the other crawlers for all domains. In particular, multi-level always beats Form Crawler with only 1 level. Note that the amount of improvement varies with the domain. Considering the total number of forms retrieved from crawling 30,000 pages, using 3 versus 1 level leads to improvements that range between 20% and 110%. This indicates that the use multiple levels in the link classifier results in an effective strategy to search for

forms. The two multi-level configurations (with 2 and 3 levels) have similar performance for both the jobs and books domains. The reason was that for these domains, the sample links in level 3 contain many *empty* features. This is illustrated in Table 1: very few of the selected features (shown in bold) are present in level 3. Note, however, that using 3 levels in the cars domain leads to a marked improvement – for 30,000 pages, the 3-level crawler retrieves 2833 forms, whereas the 2-level retrieves 2511 forms. The feature table for cars, unlike the ones for the other domains, contains many more of the selected features in level 3.

While running the baseline crawler, we noticed that it remained for a long time in certain sites, overloading these sites without retrieving any new forms. For example, in the jobs domain, after crawling 10000 pages it had retrieved only 214 pages. The baseline SC crawler avoids this problem by employing the stopping conditions we described in Section 3.4. The stopping conditions lead to a significant improvement in crawling efficiency compared to the standard baseline. Nonetheless, as Figure 2 indicates, by further focusing the search, our multi-level strategies retrieve a substantially larger number of forms than baseline SC.

The performance of the fixed-depth crawler was similar to that of the baseline crawler (without stopping conditions). As the density of forms in a site is very low, even performing a shallow crawl (using depth 3) can be inefficient. Our multi-level strategies outperform the fixed-depth crawler by over 1 order of magnitude for both cars and books, and for jobs, the gain is 5-fold.

5. CONCLUSION

In this paper we described a new crawling strategy to automatically discover hidden-Web databases. Our Form Crawler is able to efficiently perform a broad search by focusing the search on a given topic; by learning to identify promising links; and by using appropriate stop criteria that avoid unproductive searches within individual sites. Our experimental results show that our strategy is effective and that the efficiency of the Form Crawler is significantly higher than that of a representative set of crawlers.

Our initial prototype makes use of a decision-tree-based classifier to identify searchable forms. Although the test error rate for this classifier is low, it is hard to determine how well it performs with the actual forms retrieved by the Form Crawler. Since our crawls retrieve thousands of forms, it is not feasible to manually check all these forms. In future work, we plan to investigate automated techniques for evaluating the quality of the forms harvested by the Form Crawler.

Since our system uses learning algorithms to control the search, it can be used as a general framework to build form crawlers for different domains. We are currently using the Form Crawler to build a hidden-Web database directory – because it focuses the crawl on a topic, the Form Crawler is naturally suitable for this task.

Acknowledgments. This work was partially supported by the National Science Foundation under grant EIA 0323604, and by the Oregon University System.

6. REFERENCES

[1] L. Barbosa and J. Freire. Siphoning Hidden-Web Data through Keyword-Based Interfaces. In *Proc. of SBBD*, pages 309–321, 2004.

[2] M. K. Bergman. The Deep Web: Surfacing Hidden Value (White Paper). *Journal of Electronic Publishing*, 7(1), August 2001.

[3] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian. The connectivity server: Fast access to linkage information on the Web. *Computer Networks*, 30(1-7):469–477, 1998.

[4] Brightplanet's searchable databases directory. http://www.completeplanet.com.

[5] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proc. of WWW*, pages 148–159, 2002.

[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[7] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured Databases on the Web: Observations and Implications. *SIGMOD Record*, 33(3):61–70, 2004.

[8] K. C.-C. Chang, B. He, and Z. Zhang. Toward Large-Scale Integration: Building a MetaQuerier over Databases on the Web. In *Proc. of CIDR*, pages 44–55, 2005.

[9] J. Cope, N. Craswell, and D. Hawking. Automated Discovery of Search Interfaces on the Web. In *Proc. of ADC*, pages 181–189, 2003.

[10] M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori. Focused Crawling Using Context Graphs. In *Proc. of VLDB*, pages 527–534, 2000.

[11] L. Gravano, P. G. Ipeirotis, and M. Sahami. QProber: A system for automatic classification of hidden-Web databases. *ACM TOIS*, 21(1):1–41, 2003.

[12] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In *Proc. of SIGMOD*, pages 217–228, 2003.

[13] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. *VLDB Journal*, 13(3):256–273, 2004.

[14] P. Lyman and H. R. Varian. How Much Information? Technical report, UC Berkeley, 2003. http://www.sims.berkeley.edu/research/projects/how-much-info-2003/internet.htm.

[15] A. McCallum. Rainbow. http://www-2.cs.cmu.edu/ mccallum/bow/rainbow/.

[16] T. M. Mitchell. *Machine Learning*. McGraw Hill, 1997.

[17] Profusion's search engine directory. http://www.profusion.com/nav.

[18] S. Raghavan and H. Garcia-Molina. Crawling the Hidden Web. In *Proc. of VLDB*, pages 129–138, 2001.

[19] J. Rennie and A. McCallum. Using Reinforcement Learning to Spider the Web Efficiently. In *Proc. of ICML*, pages 335–343, 1999.

[20] Search engines directory. http://www.searchengineguide.com/searchengines.html.

[21] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, and P. Zimmer. The BINGO! System for Information Portal Generation and Expert Web Search. In *Proc. of CIDR*, 2003.

[22] The UIUC Web integration repository. http://metaquerier.cs.uiuc.edu/repository.

[23] Weka 3: Data Mining Software in Java. http://www.cs.waikato.ac.nz/ ml/weka.

[24] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. In *Proc. of SIGMOD*, pages 95–106, 2004.