An Adaptive Crawler for Locating Hidden-Web Entry Points

Luciano Barbosa
University of Utah
lbarbosa@cs.utah.edu

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

ABSTRACT

In this paper we describe new adaptive crawling strategies to efficiently locate the entry points to hidden-Web sources. The fact that hidden-Web sources are very sparsely distributed makes the problem of locating them especially challenging. We deal with this problem by using the contents of pages to focus the crawl on a topic; by prioritizing promising links within the topic; and by also following links that may not lead to immediate benefit. We propose a new framework whereby crawlers automatically learn patterns of promising links and adapt their focus as the crawl progresses, thus greatly reducing the amount of required manual setup and tuning. Our experiments over real Web pages in a representative set of domains indicate that online learning leads to significant gains in harvest rates—the adaptive crawlers retrieve up to three times as many forms as crawlers that use a fixed focus strategy.

Categories and Subject Descriptors

H.3.3 [**Information Search and Retrieval**]: Search process.

General Terms

Algorithms, Design, Experimentation.

Keywords

Hidden Web, Web crawling strategies, online learning, learning classifiers.

1. INTRODUCTION

The hidden Web has been growing at a very fast pace. It is estimated that there are several million hidden-Web sites [18]. These are sites whose contents typically reside in databases and are only exposed on demand, as users fill out and submit forms. As the volume of hidden information grows, there has been increased interest in techniques that allow users and applications to leverage this information. Examples of applications that attempt to make hidden-Web information more easily accessible include: metasearchers [14, 15, 26, 28], hidden-Web crawlers [2, 21], online-database directories [7, 13] and Web information integration systems [10, 17, 25]. Since for any given domain of interest, there are many hidden-Web sources whose data need to be integrated or searched, a key requirement for these applications is the ability to locate these sources. But doing so at a large scale is a challenging problem.

Given the dynamic nature of the Web—with new sources constantly being added and old sources removed and modified, it is important to *automatically discover the searchable forms* that serve as entry points to the hidden-Web databases. But searchable forms are very sparsely distributed over the Web, even within narrow domains. For example, a topic-focused best-first crawler [9] retrieves only 94 Movie search forms after crawling 100,000 pages related to movies. Thus, to efficiently maintain an up-to-date collection of hidden-Web sources, a crawling strategy must perform a broad search and simultaneously avoid visiting large unproductive regions of the Web.

The crawler must also *produce high-quality results*. Having a homogeneous set of forms that lead to databases in the same domain is useful, and sometimes required, for a number of applications. For example, the effectiveness of form integration techniques [16, 25] can be greatly diminished if the set of input forms is noisy and contains forms that are not in the integration domain. However, an automated crawling process invariably retrieves a diverse set of forms. A focus topic may encompass pages that contain searchable forms from many different database domains. For example, while crawling to find Airfare search interfaces a crawler is likely to retrieve a large number of forms in different domains, such as Rental Cars and Hotels, since these are often co-located with Airfare search interfaces in travel sites. The set of retrieved forms also includes many non-searchable forms that do not represent database queries such as forms for login, mailing list subscriptions, quote requests, and Web-based email forms.

The Form-Focused Crawler (FFC) [3] was our first attempt to address the problem of automatically locating online databases. The FFC combines techniques for focusing the crawl on a topic with a link classifier which identifies and prioritizes links that are likely to lead to searchable forms in one or more steps. Our preliminary results showed that the FFC is up to an order of magnitude more efficient, with respect to the number of searchable forms it retrieves, than a crawler that focuses the search on topic only. This approach, however, has important limitations. First, it requires substantial manual tuning, including the selection of appropriate features and the creation of the link classifier. In addition, the results obtained are highly-dependent on the quality of the set of forms used as the training for the

WWW 2007, May 8–12, 2007, Banff, Alberta, Canada.
ACM 978-1-59593-654-7/07/0005.

link classifier. If this set is not representative, the crawler may drift away from its target and obtain low harvest rates. Given the size of the Web, and the wide variation in the hyperlink structure, manually selecting a set of forms that cover a representative set of link patterns can be challenging. Last, but not least, the set of forms retrieved by the FFC is very heterogeneous—it includes all searchable forms found during the crawl, and these forms may belong to distinct database domains. For a set of representative database domains, on average, only 16% of the forms retrieved by the FFC are actually relevant. For example, in a crawl to locate airfare search forms, the FFC found 12,893 searchable forms, but among these, only 840 were airfare search forms.

In this paper, we present *ACHE* (**A**daptive **C**rawler for **H**idden-Web **E**ntries),a new framework that addresses these limitations. Given a set of Web forms that are entry points to online databases,[1] *ACHE* aims to efficiently and automatically locate other forms in the same domain. Our main contributions are:

- We frame the problem of searching for forms in a given database domain as a learning task, and present a new framework whereby crawlers adapt to their environments and *automatically* improve their behavior by learning from previous experiences. We propose and evaluate two crawling strategies: a completely automated online search, where a crawler builds a link classifier from scratch; and a strategy that combines offline and online learning.
- We propose a new algorithm that selects discriminating features of links and uses these features to automatically construct a link classifier.
- We extend the crawling process with a new module that accurately determines the relevance of retrieved forms with respect to a particular database domain. The notion of relevance of a form is user-defined. This component is essential for the effectiveness of online learning and it greatly improves the quality of the set of forms retrieved by the crawler.

We have performed an extensive performance evaluation of our crawling framework over real Web data in eight representative domains. This evaluation shows that the *ACHE* learning strategy is *effective*—the crawlers are able to adapt and significantly improve their harvest rates as the crawl progresses. Even starting from scratch (without a link classifier), *ACHE* is able to obtain harvest rates that are comparable to those of crawlers like the FFC, that are constructed using prior knowledge. The results also show that *ACHE* is effective and obtains harvest rates that are substantially higher than a crawler whose focus is only on page content—these differences are even more pronounced when only relevant forms (i.e., forms belong to the target database domain) are considered. Finally, the results also indicate that the automated feature selection is able to identify *good* features, which for some domains were more effective than features identified manually.

The remainder of the paper is organized as follows. Since *ACHE* extends the focus strategy of the FFC, to make the paper self-contained, in Section 2 we give a brief overview of the FFC and discuss its limitations. In Section 3, we present the adaptive-learning framework of *ACHE* and describe the underlying algorithms. Our experimental evaluation is discussed in Section 4. We compare our approach to related work in Section 5 and conclude in Section 6, where we outline directions for future work.

[1]In this paper, we use the terms 'online database' and 'hidden-Web source' interchangeably.

2. BACKGROUND: THE FORM-FOCUSED CRAWLER

The FFC is trained to efficiently locate forms that serve as the entry points to online databases—it focuses its search by taking into account both the *contents of pages* and *patterns in and around the hyperlinks in paths to a Web page*. The main components of the FFC are shown in white in Figure 1 and are briefly described below.

- 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 [9]. Once the crawler retrieves a page P, if P is classified as being on-topic, its forms and links are extracted.
- 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 the links to the crawling frontier in the order of their predicted reward.
- The *frontier manager* maintains a set of priority queues with links that are yet to be visited. At each crawling step, it selects the link with the highest priority.
- The *searchable form classifier* filters out non-searchable forms and ensures only searchable forms are added to the *Form Database*. This classifier is domain-independent and able to identify searchable forms with high accuracy. The *crawler* also employs stopping criteria to deal with the fact that sites, in general, contain few searchable forms. It leaves a site after retrieving a pre-defined number of distinct forms, or after it visits a pre-defined number of pages in the site.

These components and their implementation are described in [3]. Below we discuss the aspects of the link classifier and frontier manager needed to understand the adaptive learning mechanism of *ACHE*.

2.1 Link Classifier

Since forms are sparsely distributed on the Web, by prioritizing only links that bring immediate return, i.e., links whose patterns are similar to those that point to pages containing searchable forms, the crawler may miss target pages that can only be reached with additional steps. The link classifier aims to also identify links that have *delayed benefit* and belong to paths that will *eventually* lead to pages that contain forms. It learns to estimate the distance (the length of the path) between a link and a target page based on link patterns: given a link, the link classifier assigns a score to the link which corresponds to the distance between the link and a page that contains a relevant form.

In the FFC, the link classifier is built as follows. Given a set of URLs of pages that contain forms in a given database domain, paths to these pages are obtained by crawling backwards from these pages, using the `link:` facility provided by search engines such as Google and Yahoo! [6]. The backward crawl proceeds in a breadth-first manner. Each level

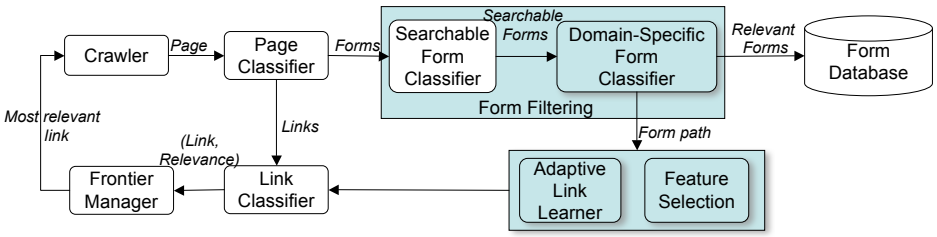


Figure 1: Architecture of *ACHE*. The new modules that are responsible for the online focus adaptation are shown in blue; and the modules shown in white are used both in the FFC and in *ACHE*.

l+1 is constructed by retrieving all documents that point to documents in level *l*. From the set of paths gathered, we *manually* select the best features. Using these data, the classifier is trained to estimate the distance between a given link and a target page that 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.

2.2 Frontier Manager

The goal of the frontier manager is to maximize the expected reward for the crawler. Each link in the frontier is represented by a tuple $(link, Q)$, where Q reflects the expected reward for $link$:

$$Q(state, link) = reward \quad (1)$$

Q maps a *state* (the current crawling frontier) and a link $link$ to the expected *reward* for following $link$. The value of Q is approximated by discretization and is determined by: (1) the distance between $link$ and the target pages—links that are closer to the target pages have a higher Q value and are placed in the highest priority queues; (2) the likelihood of $link$ belonging to a given level.

The frontier manager is implemented as a set of N queues, where each queue corresponds to a link classifier level: a link l is placed on queue i if the link classifier estimates l is i steps from a targe page. Within a queue, links are ordered based on their likelihood of belonging to the level associated with the queue.

Although the goal of frontier manager is to maximize the expected reward, if it only chooses links that give the best expected rewards, it may forgo links that are sub-optimal but that lead to high rewards in the future. To ensure that links with delayed benefit are also selected, the crawling frontier is updated in batches. When the crawler starts, all seeds are placed in queue 1. At each step, the crawler selects the link with the highest relevance score from the first non-empty queue. If the page it downloads belongs to the target topic, its links are classified by link classifier and added to a separate *persistent frontier*. Only when the queues in the crawling frontier become empty, the crawler loads the queues from the persistent frontier.

2.3 Limitations of FFC

An experimental evaluation of the FFC [3] showed that FFC is more efficient and retrieves up to an order of magnitude more searchable forms than a crawler that focuses only on topic. In addition, FFC configurations with a link classifier that uses multiple levels performs uniformly better than their counterpart with a single level (i.e., a crawler that focuses only on immediate benefit). The improvements in harvest rate for the multi-level configurations varied between 20% and 110% for the three domains we considered. This confirms results obtained in other works which underline the importance of taking delayed benefit into account for sparse concepts [11, 22].

The strategy used by the FFC has two important limitations. The set of *forms retrieved by the FFC is highly heterogeneous*. Although the Searchable Form Classifier is able to filter out non-searchable forms with high accuracy, a qualitative analysis of the searchable forms retrieved by the FFC showed that the set contains forms that belong to many different database domains. The average percentage of relevant forms (i.e., forms that belong to the target domain) in the set was low—around 16%. For some domains the percentage was as low as 6.5%. Whereas it is desirable to list only relevant forms in online database directories, such as BrightPlanet [7] and the Molecular Biology Database Collection [13], for some applications this is a requirement. Having a homogeneous set of the forms that belong to the same database domain is critical for techniques such as statistical schema matching across Web interfaces [16], whose effectiveness can be greatly diminished if the set of input forms is noisy and contains forms from multiple domains.

Another limitation of the FFC is that *tuning the crawler and training the link classifier can be time consuming*. The process used to select the link classifier features is manual: terms deemed as representative are manually selected for each level. The quality of these terms is highly-dependent on knowledge of the domain and on whether the set of paths obtained in the back-crawl is representative of a wider segment of the Web for that database domain. If the link classifier is not built with a representative set of paths for a given database domain, because the FFC uses a fixed focus strategy, the crawler will be confined to a possibly small subset of the promising links in the domain.

3. DYNAMICALLY ADAPTING THE CRAWLER FOCUS

With the goal of further improving crawler efficiency, the quality of its results, and automating the process of crawler setup and tuning, we use a learning-agent-based approach to the problem of locating hidden-Web entry points.

Learning agents have four components [23]:

- The *behavior generating element* (BGE), which based on the current state, selects an action that tries to maximize the expected reward taking into account its goals (*exploitation*);
- The *problem generator* (PG) that is responsible for suggesting actions that will lead to new experiences, even if the

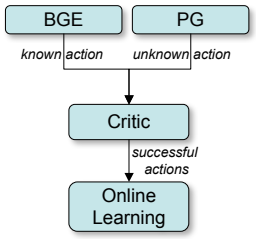


Figure 2: Highlight of the main components involved in the adaptive aspect of a learning agent.

benefit is not immediate, i.e., the decision is locally suboptimal (*exploration*);

- The *critic* that gives the online learning element feedback on the success (or failure) of its actions; and
- The *online learning element* which takes the critic's feedback into account to update the policy used by the BGE.

A learning agent must be able to learn from new experiences and, at the same time, it should be robust with respect to biases that may be present in these experiences [20, 23]. An agent's ability to learn and adapt relies on the successful interaction among its components (see Figure 2). Without exploration, an agent may not be able to correct biases introduced during its execution. If the BGE is ineffective, the agent is not able to exploit its acquired knowledge. Finally, a high-quality critic is crucial to prevent the agent from drifting away from its objective. As we discuss below, *ACHE* combines these four elements to obtain all the advantages of using a learning agent.

3.1 The *ACHE* Architecture

Figure 1 shows the high-level architecture of *ACHE*. The components that we added to enable the crawler to learn from its experience are highlighted (in blue). The *frontier manager* (Section 2.2) acts as both the BGE and PG and balances the trade-off between exploration and exploitation. It does so by using a policy for selecting unvisited links from the crawling frontier which considers links with both immediate and delayed benefit. The Q function (Equation 1) provides the exploitation component (BGE). It ensures the crawler exploits the acquired knowledge to select actions that yield high reward, i.e., links that lead to relevant forms. By also selecting links estimated to have delayed reward, the frontier manager provides an exploratory component (PG), which enables the crawler to explore actions with previously unknown patterns. This exploratory behavior makes *ACHE* robust and enables it to correct biases that may be introduced in its policy. We discuss this issue in more detail in Section 4.

The *form filtering* component is the critic. It consists of two classifiers: the *searchable form classifier* (SFC)[2]; and the *domain-specific form classifier* (DSFC). Forms are processed by these classifiers in a sequence: each retrieved form is first classified by the SFC as searchable or non-searchable; the DSFC then examines the searchable forms and indicates whether they belong to the target database domain (see Section 3.4 for details).

[2]The SFC is also used in the FFC.

Algorithm 1 Adaptive Link Learner

1: **if** $learningThresholdReached$ **then**
2: $paths = collectPaths(relevantForms, length)$
{Collect paths of a given length to pages that contain relevant forms.}
3: $features = FeatureSelector(paths)$
{Select the features from the neighborhood of links in the paths.}
4: $linkClassifier = createClassifier(features, paths)$
{Create new link classifier.}
5: $updateFrontier(linkClassifier)$
{Re-rank links in the frontier using the new link classifier.}
6: **end if**

The policy used by the frontier manager is set by the *link classifier*. In *ACHE*, we employ the *adaptive link learner* as the learning element. It dynamically learns features automatically extracted from successful paths by the *feature selection* component, and updates the link classifier. The effectiveness of the adaptive link learner depends on the accuracy of the form-filtering process; on the ability of the feature selector to identify 'good' features; and on the efficacy of the frontier manager in balancing exploration and exploitation. Below we describe the components and algorithms responsible for making *ACHE* adaptive.

3.2 Adaptive Link Learner

In the FFC, link patterns are learned offline. As described in Section 2.1, these patterns are obtained from paths derived by crawling backwards from a set of pages that contain relevant forms. The adaptive link learner, in contrast, uses features of paths that are gathered during the crawl. *ACHE* keeps a repository of successful paths: when it identifies a relevant form, it adds the path it followed to that form to the repository. Its operation is described in Algorithm 1. The adaptive link learner is invoked periodically, when the learning threshold is reached (line 1). For example, after the crawler visits a pre-determined number of pages, or after it is able to retrieve a pre-defined number of relevant forms.

Note that if the threshold is too low, the crawler may not be able to retrieve enough new samples to learn effectively. On the other hand, if the value is too high, the learning rate will be slow. In our experiments, learning iterations are triggered after 100 new relevant forms are found.

When a learning iteration starts, features are automatically extracted from the new paths (Section 3.3). Using these features and the set of path instances, the adaptive link learner generates a new link classifier.[3] As the last step, the link learner updates the frontier manager with the new link classifier. The frontier manager then updates the Q values of the links using the new link classifier, i.e., it re-ranks all links in the frontier using the new policy.

3.3 Automating the Feature Selection Process

The effectiveness of the link classifier is highly-dependent on the ability to identify discriminating features of links. In *ACHE*, these features are automatically extracted, as described in Algorithm 2. The *Automatic Feature Selection* (*AFS*) algorithm extracts features present in the anchor, URL, and text around links that belong to paths which lead to relevant forms.

[3]The length of the paths considered depends on the number of levels used in the link classifier.

Algorithm 2 *Automatic Feature Selection*

```
1: Input: set of links at distance d from a relevant form
2: Output: features selected in the three feature spaces—
   anchor, URL and around
3: for each featureSpace do
4:   termSet = getTermSet(featureSpace, paths)
     {From the paths, obtain terms in specified feature space.}
5:   termSet = removeStopWords(termSet)
6:   stemmedSet = stem(termSet)
7:   if featureSpace == URL then
8:     topKTerms= getMostFrequentTerms(stemmedSet, k)
       {Obtain the set of k most frequent terms.}
9:     for each term t ∈ topKTerms do
10:      for each term t' ∈ stemmedSet that contains the sub-
         string t do
11:        addFrequency(stemmedSet,t,t')
           {Add frequency of t' to t in stemmedSet.}
12:      end for
13:    end for
14:  end if
15:  selectedFeatures = getNMostFrequentTerms(termSet)
     {Obtain a set of the top n terms.}
16: end for
```

Initially, all terms in anchors are extracted to construct the *anchor feature set*. For the *around feature set*, *AFS* selects the n terms that occur before and the n terms that occur after the anchor (in textual order). Because the number of extracted terms in these different contexts tends to be large, stop-words are removed (line 5) and the remaining terms are stemmed (line 6). The most frequent terms are then selected to construct the feature set (line 15).

The URL feature space requires special handling. Since there is little structure in a URL, extracting terms from a URL is more challenging. For example, "jobsearch" and "usedcars" are terms that appear in URLs of the Job and Auto domains, respectively. To deal with this problem, we try to identify meaningful sub-terms using the following strategy. After the terms are stemmed, the k most frequent terms are selected (topKTerms in line 8). Then, if a term in this set appears as a substring of another term in the URL feature set, its frequency is incremented. Once this process finishes, the k most frequent terms are selected.

The feature selection process must produce features that are suitable for the learning scheme used by the underlying classifier. For text classification Zheng et al. [29] show that the Naïve Bayes model obtains better results with a much lower number of features than linear methods such as Support Vector Machines [20]. As our link classifier is built using the Naïve Bayes model, we performed an aggressive feature selection and selected a small number of terms for each feature space. The terms selected are the ones with highest document frequency (DF)[4]. Experiments conducted by Yang and Pedersen [27] show that DF obtains results comparable to task-sensitive feature selection approaches, as information gain [20] and Chi-square [12].

AFS is very simple to implement, and as our experimental results show, it is very effective in practice.

3.4 Form Filtering

The form filtering component acts as a critic and is responsible for identifying relevant forms gathered by *ACHE*. It assists *ACHE* in obtaining high-quality results and it also enables the crawler to adaptively update its focus strategy, as it identifies new paths to relevant forms during a crawl. Therefore, the overall performance of the crawler agent is highly-dependent on the accuracy of the form-filtering process. If the classifiers are inaccurate, crawler efficiency can be greatly reduced as it drifts way from its objective through unproductive paths.

The form-filtering process needs to identify, among the set of forms retrieved by the crawler, forms that belong to the target database domain. Even a focused crawler retrieves a highly-heterogeneous set of forms. A focus topic (or concept) may encompass pages that contain many different database domains. For example, while crawling to find airfare search interfaces the FFC also retrieves a large number of forms for rental car and hotel reservation, since these are often co-located with airfare search interfaces in travel sites. The retrieved forms also include non-searchable forms that do not represent database queries such as forms for login, mailing list subscriptions, and Web-based email forms.

ACHE uses HIFI, a hierarchical classifier ensemble proposed in [4], to filter out irrelevant forms. Instead of using a single, complex classifier, HIFI uses two simpler classifiers that learn patterns of different subsets of the form feature space. The *Generic Form Classifier* (*GFC*) uses structural patterns which determine whether a form is searchable. Empirically, we have observed that these structural characteristics of a form are a good indicator as to whether the form is searchable or not [3]. To identify searchable forms that belong to a given domain, HIFI uses a more specialized classifier, the *Domain-Specific Form Classifier* (*DSFC*). The DSFC uses the textual content of a form to determine its domain. Intuitively, the form content is often a good indicator of the database domain—it contains metadata and data that pertain to the database.

By partitioning the feature space, not only can simpler classifiers be constructed that are more accurate and robust, but this also enables the use of learning techniques that are more effective for each feature subset. Whereas decision trees [20] gave the lowest error rates for determining whether a form is searchable based on structural patterns, SVMs [20] proved to be the most effective technique to identify forms that belong to the given database domain based on their textual content.

The details of these classifiers are out of the scope of this paper. They are described in [4], where we show that the combination of the two classifiers leads to very high precision, recall and accuracy. The effectiveness of the form filtering component is confirmed by our experimental evaluation (Section 4): significant improvements in harvest rates are obtained by the adaptive crawling strategies. For the database domains used in this evaluation, the combination of these two classifiers results in accuracy values above 90%.

4. EXPERIMENTS

We have performed an extensive performance evaluation of our crawling framework over real Web data in eight representative domains. Besides analyzing the overall performance of our approach, our goals included: evaluating the effectiveness of *ACHE* in obtaining high-quality results (i.e., in retrieving relevant forms); the quality of the features automatically selected by *AFS*; and assessing the effectiveness of online learning in the crawling process.

[4]Document frequency represents the number of documents in a collection where a given term occurs.

Domain	Description	Density	Norm. Density
Airfare	airfare search	0.132%	1.404
Auto	used cars	0.962%	10.234
Book	books search	0.142%	1.510
Hotel	hotel availability	1.857%	19.755
Job	job search	0.571%	6.074
Movie	movie titles and DVDs	0.094%	1.000
Music	music CDs	0.297%	3.159
Rental	car rental availability	0.148%	1.574

Table 1: Database domains used in experiments and density of forms in these domains. The column labeled *Norm. Density* shows the density values normalized with respect to the lowest density value (for the Movie domain).

4.1 Experimental Setup

Database Domains. We evaluated our approach over the eight online database domains described in Table 1. This table also shows the density of relevant forms in the domains. Here, we measure density as the number of distinct relevant forms retrieved by a topic-focused crawler (the baseline crawler described below) divided by the total number of pages crawled. Note that not only are forms very sparsely distributed in these domains, but also that there is a large variation in density across domains. In the least dense domain (Movie), only 94 forms are found after the baseline crawler visits 100,000 pages; whereas in the densest domain (Hotel), the same crawler finds 19 times as many forms (1857 forms).

Crawling Strategies. To evaluate the benefit of online learning in *ACHE*, we ran the following crawler configurations:

- *Baseline*, a variation of the best-first crawler [9]. The page classifier guides the search and the crawler follows all links that belong to a page whose contents are classified as being on-topic. One difference between baseline and the best-first crawler is that the former uses the same stopping criteria as the FFC; [5]
- *Offline Learning*, the crawler operates using a fixed policy that remains unchanged during the crawling process—this is the same strategy used by the FFC [3];
- *Offline-Online Learning*, *ACHE* starts with a pre-defined policy, and this policy is dynamically updated as the crawl progresses;
- *Online Learning*, *ACHE* starts using the baseline strategy and builds its policy dynamically, as pages are crawled.

All configurations were run over one hundred thousand pages; and the link classifiers were configured with three levels.

Effectiveness measure. Since our goal is to find searchable forms that serve as entry points to a given database domain, it is important to measure harvest rate of the crawlers based on the number of *relevant* forms retrieved per pages crawled. It is worthy of note that harvest rates reported in [3] for the FFC (offline learning) took into account all searchable forms retrieved—a superset of the relevant forms. Below, as a point of comparison, we also show the harvest rates for the different crawlers taking all searchable forms into account.

[5]In earlier experiments, we observed that without the appropriate stopping criteria, the best-first crawler gets *trapped* in some sites, leading to extremely low harvest rates [3].

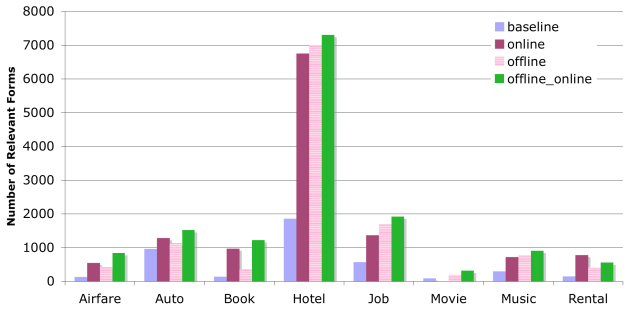


Figure 3: Number of relevant forms returned by the different crawler configurations.

4.2 Focus Adaptation and Crawler Efficiency

Figure 3 gives, for each domain, the number of relevant forms retrieved by the four crawler configurations. Online learning leads to substantial improvements in harvest rates when applied to both the *Baseline* and *Offline* configurations. The gains vary from 34% to 585% for *Online* over *Baseline*, and from 4% to 245% for *Offline-Online* over *Offline*. These results show that the adaptive learning component of *ACHE* is able to automatically improve its focus based on the feedback provided by the form filtering component. In addition, *Online* is able to obtain substantial improvements over *Baseline* in a completely automated fashion—requiring no initial link classifier and greatly reducing the effort to configure the crawler. The only exception is the Movie domain. For Movie, the most sparse domain we considered, the *Online* configuration was not able to learn patterns with enough support from the 94 forms encountered by *Baseline*.

Effect of Prior Knowledge. Having background knowledge in the form of a 'good' link classifier is beneficial. This can be seen from the fact that *Offline-Online* retrieved the largest number of relevant forms in all domains (except for Rental, see discussion below). This knowledge is especially useful for very sparse domains, where the learning process can be prohibitively expensive due to the low harvest rates.

There are instances, however, where the prior knowledge limits the crawler to visit a subset of the productive links. If the set of patterns in the initial link classifier is too narrow, it will prevent the crawler from visiting other relevant pages reachable through paths that are not represented in the link classifier. Consider, for example the Rental domain, where *Online* outperforms *Offline-Online*. This behavior may sound counter-intuitive, since both configurations apply online learning and *Offline-Online* starts with an advantage. The initial link classifier used by *Offline-Online* was biased, and the adaptive process was slow at correcting this bias. A closer examination of the features used by *Offline-Online* shows that, over time, they converge to the same set of features of *Online*. The *Online*, in contrast, started with no bias and was able to outperform *Offline-Online* in a window of 100,000 pages.

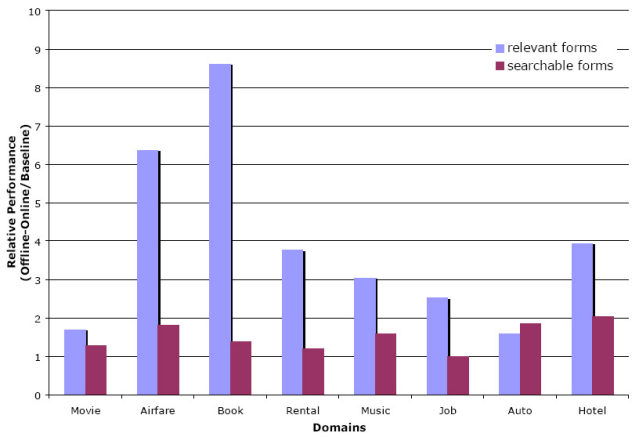


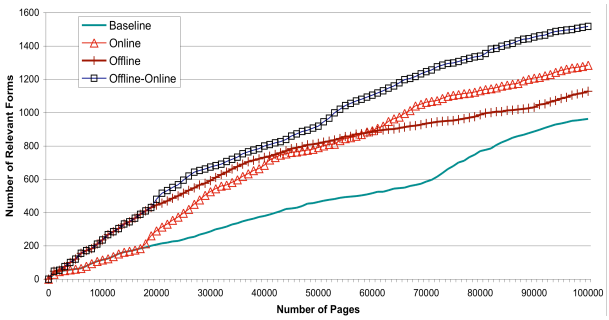
Figure 4: Relative performance of *Offline-Online* over *Baseline*. The domains are ordered with respect to their densities.

The presence of bias in the link classifier also explains the poor performance of *Offline* in Rental, Book and Airfare. For these domains, *Offline-Online* is able to eliminate the initial bias. *ACHE* automatically adapts and learns new patterns, leading to a substantial increase the number of relevant forms retrieved. In the Book domain, for instance, the initial link classifier was constructed using manually gathered forms from online bookstores. Examining the forms obtained by the *Offline-Online*, we observed that forms for online bookstores are only a subset of the relevant forms in this domain. A larger percentage of relevant forms actually appear in library sites. *ACHE* successfully learned patterns to these sites (see Table 2).

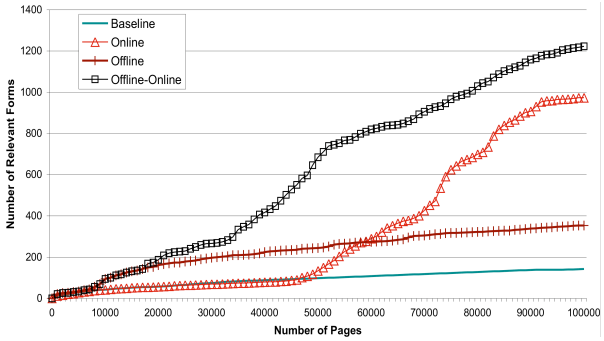
Another evidence of the effectiveness of the adaptive learning strategy is the fact that *Online* outperforms *Offline* for four domains: Airfare, Auto, Book, and Rental. For the latter two, *Online* retrieved 275% and 190% (resp.) more forms than *Offline*. This indicates that a completely automated approach to learning is effective and able to outperform a manually configured crawler.

The Link Classifier and Delayed Benefit. Figure 4 shows the relative performance between the *Offline-Online* configuration of *ACHE* and *Baseline*, with respect to both relevant forms and searchable forms. Here, the domains are ordered (in the x axis) by increasing order of density. Note that for the sparser domains, the performance difference between *ACHE* and *Baseline* is larger. Also note that the gains from delayed benefit are bigger when the performance is measured with respect to relevant forms. For example, in the Book domain, *Offline-Online* retrieves almost 9 times more relevant forms than *Baseline*. The performance difference is much smaller for searchable forms—*Offline-Online* retrieves only 10% more searchable forms than *Baseline*. This can be explained due to the fact that searchable forms are much more prevalent than relevant forms within a focus topic. The numbers in Figure 4 underline the importance of taking delayed benefit into account while searching for sparse concepts.

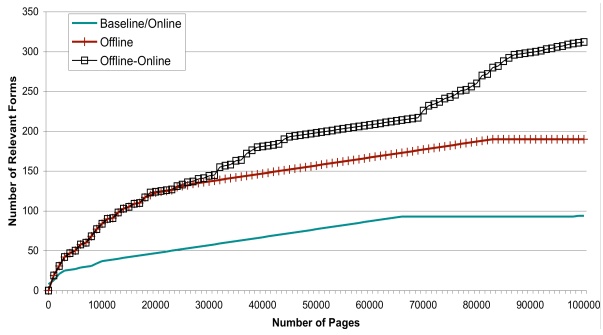
Delayed benefit also plays an important role in the effectiveness of the adaptive learning component of *ACHE*. The use of the link classifier forces *ACHE* to explore paths with previously unknown patterns. This exploratory behavior is key to adaptation. For example, in the Book domain (see



(a) Auto



(b) Book



(c) Movie

Figure 5: Number of forms retrieved over time.

Table 2), since the initial link classifier has a bias towards online bookstores, if *ACHE* only followed links predicted to yield immediate benefit, it would not be able to reach the library sites. Note, however, that the exploratory behavior can potentially lead the crawler to lose its focus. But as our experimental results show, *ACHE* is able to obtain a good balance, being able to adapt to new patterns while maintaining its focus.

Crawler Performance over Time. To give insight about the behavior of the different crawler configurations, it is useful to analyze how their harvest rates change over time. Here, we only show these results for Book, Auto and Movie in Figure 5. Similar results were obtained for the other domains.

Note that the number of relevant forms retrieved by *Online* and *Baseline* coincide initially, and after the crawler starts the learning process, the performance of *Online* improves substantially. A similar trend is observed for *Offline* and *Offline-Online*—the number of relevant forms retrieved by *Offline-Online* increases after its policy is first updated.

Another interesting observation is that the rate of increase in the number of relevant forms retrieved is higher for the configurations that use online-learning.

The increase in the number of forms retrieved over time is a good indication of the effectiveness of online learning for a particular domain. For Movie (Figure 5(c)), after crawling 66,000 pages, *Baseline* retrieves only 93 relevant forms. After that, the number of forms remain (almost) constant (a single additional form is found in the last 30,000 pages). Because so few relevant forms are found, *Online* is not able to learn due to insufficient evidence for the link patterns.Note that the lines for *Baseline* and *Online* coincide for the Movie domain.

In the Book domain, which is also sparse but less so than Movie, *Online* was able to learn useful patterns and substantially improve the harvest rate. As shown in Figure 5(b), the first learning iteration happened after 50,000 pages had been crawled—much later than for the other denser domains. For example, for Auto the first iteration occurs at 17,000 pages (Figure 5(a)). The Auto domain provides a good example of the adaptive behavior of *ACHE* in denser domains.

These results show that, even starting with no information about patterns that lead to relevant forms, these patterns can be learned dynamically and crawler performance can be improved in an automatic fashion. However, the sparser the domain is, the harder it is for the crawler to learn. For *Online* to be effective in a very sparse domain, a crawler that is more focused than *Baseline* is needed initially.

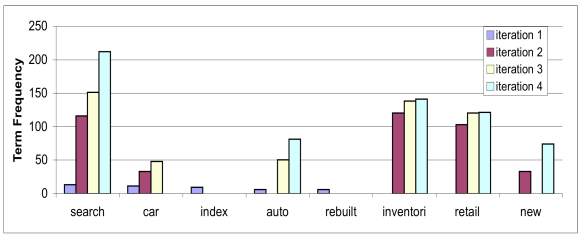
4.3 Feature Selection

The performance improvement obtained by the adaptive crawler configurations provides evidence of the effectiveness of the automatic feature selection described in Section 3.3. As an example of its operation, consider Figure 6, which shows the terms selected by *AFS* in 4 learning iterations for the Auto domain using the *Online* configuration. Note that *AFS* is able to identify good features from scratch and without any manual intervention. For both Anchor and Around feature sets, already in the first iteration, relevant terms are discovered which remain in subsequent iterations (e.g., car, auto), although their frequency changes over time. For instance, the term "auto" has the highest frequency in the first iteration, whereas "car" has the highest frequency after the second iteration.

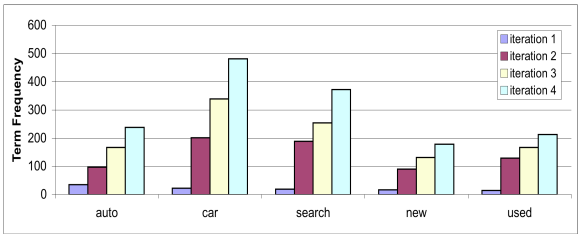
Unlike Anchor and Around, the URL feature set is not so well-behaved. Because URLs contain uncommon terms and more variability, the patterns take longer to converge. As Figure 6 shows, after the first iteration the *AFS* selects terms that disappear in subsequent iterations (e.g., "index" and "rebuilt"). In addition, the frequencies of terms in the URL are much lower than in the other feature spaces.

A final observation about the automatic feature selection is that by analyzing how the features evolve over time, and change at each learning iteration, insights can be obtained about the adaptive behavior of *ACHE*. For example, as Table 2 illustrates, for the *Offline-Online* in the Book domain, the features selected for the initial link classifier are clearly related to online bookstores (e.g., book, search and bookstor). As new relevant forms are encountered, new terms are introduced that are related to library sites (e.g., ipac,[6] librari, book, search, and catalog).

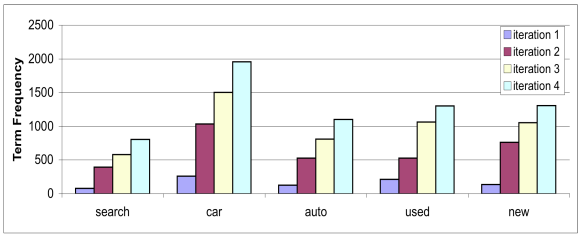
[6]ipac is a system used by some library sites to search their catalogs.



(a) URL



(b) Anchor



(c) Around

Figure 6: Features automatically extracted in different iterations of adaptive learning for *Online* in the Auto domain.

5. RELATED WORK

There is a rich literature in the area of focused crawlers (see e.g., [1, 3, 8, 9, 11, 22, 24, 19]). Closely related to our work are strategies that apply online-learning and that take delayed benefit into account. We discuss these below.

Delayed Benefit. Rennie and McCallum [22] frame the problem of creating efficient crawlers as a reinforcement learning task. They describe an algorithm for learning a function that maps hyperlinks to future discounted reward: the expected number of relevant pages that can be found as a result of following that link. They show that by taking delayed rewards into account, their RL Spider is able to efficiently locate sparse concepts on a predetermined universe of URLs. There are several important differences between the RL Spider and *ACHE*. First and foremost, the RL Spider does not perform a broad search over the Web. It requires as input the URLs of all sites to be visited and performs a focused search only within these pre-determined sites. Second, it is not adaptive—the classifier maintains a fixed policy during the crawl. Finally, although their classifier also learns to prioritize links and it considers links that have delayed benefit, the learning function used in *ACHE* is different: the link classifier estimates the distance between a link and a relevant form (see Section 2.1).

The importance of considering delayed benefit in a focused crawl was also discussed by Diligenti et al. [11]. Their

Iteration	Selected Features		
	URL	Anchor	Around
0 (initial features)	book,search,addal,natur,hist	book,search,addal,bookstor,link	book,search,onlin,new,bookstor
1	search,index,book	search,book,advanc,librari,engin	book,search,titl,onlin,author
2	search,adv,lib,index,ipac	search,book,advanc,librari,catalog	book,search,librari,titl,author
3	search,lib,ipac,profil,catalog	search,librari,catalog,advanc,book	librari,search,book,catalog,titl
4	search,lib,ipac,profil,catalog	librari,search,catalog,advanc,book	librari,search,catalog,book,titl
5	search,lib,ipac,profil,catalog	librari,search,catalog,advanc,book	librari,search,book,catalog,public

Table 2: Features selected during the execution of the Offline-Online in Book domain.

Context Focused Crawler (CFC) learns a hierarchy of concepts related to a search topic. The intuition behind their approach is that topically relevant pages can be found by using a context graph which encodes topics that are directly or indirectly related to the target topic. By performing a backward crawl from a sample set of relevant pages, they create classifiers that identify these related topics and estimate, for a given page r, the distance between r and a topic-relevant page. Unlike *ACHE*, the focus of the CFC is solely based on the contents of pages—it does not prioritize links. If a page is considered relevant, all links in that page will be followed. Like the RL Spider, the CFC uses a fixed focus strategy.

The FFC [3] combines ideas from these two approaches. It employs two classifiers: one that uses page contents that focuses the search on a given topic; and another that identifies promising links within the focus topic. An experimental evaluation over three distinct domains showed that combining the page contents and hyperlink structure leads to substantial improvement in crawler efficiency. The differences between FFC and *ACHE* are discussed in Section 3.

Unlike these approaches, *ACHE* adaptively updates its focus strategy as it learns from new experience. There is an important benefit derived from combining delayed benefit and online-learning: following links that have delayed benefit forces the crawler to explore new paths and enables it to learn new patterns. This is in contrast to strategies based on immediate benefit that exploit actions it has already learned will yield high reward. This exploratory behavior makes our adaptive learning strategy robust and enables it to correct biases created in the learning process. This was observed in several of the domains we considered in our experimental evaluation (Section 4). In the Book domain, for instance, the crawler with a fixed policy (*Offline*) was trapped in a subset of promising links related to online bookstores, whereas *ACHE* was able to eliminate this bias in the first learning iteration and learn new patterns that allowed it to also obtain relevant forms from online library sites.

Online Learning Policies. Chakrabarti et al. [8] proposed an online learning strategy that, similar to *ACHE* uses two classifiers to focus the search: a baseline page classifier that learns to classify pages as belonging to topics in a taxonomy [9]; and the apprentice, a classifier that learns to identify the most promising links in a topic-relevant page. Their motivation to use the apprentice comes from the observation that, even in domains that are not very narrow, the number of links that are irrelevant to the topic can be very high. Thus, following all the links in a topic-relevant page can be wasteful. 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 *ACHE* also attempts to estimate the benefit of following a particular link based on the crawler experience, there is an important difference. Because the apprentice only follows links that give immediate benefit, biases that are introduced by the online learning process are reinforced as the crawl progresses—the crawler will repeatedly exploit actions it has already learned will yield high reward. In contrast, as discussed above, by considering links that may lead to delayed benefit, *ACHE* has a more *exploratory* behavior and will visit unknown states and actions.

It is worthy of note that the goal of our link classifier is complementary to that of the apprentice—it aims to learn which links lead to pages that contain relevant forms, whereas the goal of the apprentice is to avoid off-topic pages. In addition, we are dealing with concepts that are much sparser than the ones considered by Chakrabarti et al.. For example, the density of the domains considered in [8] varied from 31% to 91%, whereas for concepts we considered density values range between 0.094% and 1.857%. Nonetheless, our approach is likely to 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.

Aggarwal et al. [1] proposed an online-learning strategy to learn features of pages that satisfies a user-defined predicate. They start the search with a generic crawler. As new pages that satisfy the user-defined predicates are encountered, the crawler gradually constructs its focus policy. The method of identifying relevant documents is composed by different predictors for content and link structure. Manual tuning is required to determine contribution of each predictor to the final result. In addition, similar to Chakrabarti et al. their strategy only learns features that give immediate benefit. Another drawback of this approach is its use of a generic crawler at the beginning of its execution. Because a generic crawler may need to visit a very large number of pages in order to obtain a significant sample, the learning costs may be prohibitive for sparse domains. As a point of comparison, consider the behavior of the online crawler for the Movie domain (Section 4). Even using a focused crawler, only 94 relevant forms are retrieved in a 100,000 page crawl, and these were not sufficient to derive useful patterns. A much larger number of pages would have to be crawled by a general crawler to obtain the same 94 forms.

6. CONCLUSION AND FUTURE WORK

We have presented a new adaptive focused crawling strategy for efficiently locating hidden-Web entry points. This strategy effectively balances the exploitation of acquired knowledge with the exploration of links with previously unknown

patterns, making it robust and able to correct biases introduced in the learning process. We have shown, through a detailed experimental evaluation, that substantial increases in harvest rates are obtained as crawlers learn from new experiences. Since crawlers that learn from scratch are able to obtain harvest rates that are comparable to, and sometimes higher than manually configured crawlers, this framework can greatly reduce the effort to configure a crawler. In addition, by using the form classifier, *ACHE* produces high-quality results that are crucial for a number information integration tasks.

There are several important directions we intend to pursue in future work. As discussed in Section 5, we would like to integrate the apprentice of [8] into the *ACHE* framework. To accelerate the learning process and better handle very sparse domains, we will investigate the effectiveness and trade-offs involved in using back-crawling during the learning iterations to increase the number of sample paths. Finally, to further reduce the effort of crawler configuration, we are currently exploring strategies to simplify the creation of the domain-specific form classifiers. In particular, the use of form clusters obtained by the online-database clustering technique described in [5] as the training set for the classifier.

Acknowledgments. This work is partially supported by the National Science Foundation (under grants IIS-0513692, CNS-0524096, IIS-0534628) and a University of Utah Seed Grant.

7. REFERENCES

[1] C. C. Aggarwal, F. Al-Garawi, and P. S. Yu. Intelligent crawling on the world wide web with arbitrary predicates. In *Proceedings of WWW*, pages 96–105, 2001.

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

[3] L. Barbosa and J. Freire. Searching for Hidden-Web Databases. In *Proceedings of WebDB*, pages 1–6, 2005.

[4] L. Barbosa and J. Freire. Combining classifiers to identify online databases. In *Proceedings of WWW*, 2007.

[5] L. Barbosa, J. Freire, and A. Silva. Organizing hidden-web databases by clustering visible web documents. In *Proceedings of ICDE*, 2007. To appear.

[6] 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.

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

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

[9] 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.

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

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

[12] T. Dunnin. Accurate methods for the statistics of surprise and coincidence. *Computational Linguistics*, 19(1):61–74, 1993.

[13] M. Galperin. The molecular biology database collection: 2005 update. *Nucleic Acids Res*, 33, 2005.

[14] Google Base. http://base.google.com/.

[15] L. Gravano, H. Garcia-Molina, and A. Tomasic. Gloss: Text-source discovery over the internet. *ACM TODS*, 24(2), 1999.

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

[17] H. He, W. Meng, C. Yu, and Z. Wu. Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In *Proceedings of VLDB*, pages 357–368, 2003.

[18] W. Hsieh, J. Madhavan, and R. Pike. Data management projects at Google. In *Proceedings of ACM SIGMOD*, pages 725–726, 2006.

[19] H. Liu, E. Milios, and J. Janssen. Probabilistic models for focused web crawling. In *Proceedings of WIDM*, pages 16–22, 2004.

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

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

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

[23] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2002.

[24] 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 *Proceedings of CIDR*, 2003.

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

[26] J. Xu and J. Callan. Effective retrieval with distributed collections. In *Proceedings of SIGIR*, pages 112–120, 1998.

[27] Y. Yang and J. O. Pedersen. A Comparative Study on Feature Selection in Text Categorization. In *International Conference on Machine Learning*, pages 412–420, 1997.

[28] C. Yu, K.-L. Liu, W. Meng, Z. Wu, and N. Rishe. A methodology to retrieve text documents from multiple databases. *TKDE*, 14(6):1347–1361, 2002.

[29] Z. Zheng, X. Wu, and R. Srihari. Feature selection for text categorization on imbalanced data. *ACM SIGKDD Explorations Newsletter*, 6(1):80–89, 2004.