IEEE Transactions on Services Computing Volume: PP Year: 2015
IEEE Transactions on Services Computing, Vol 99 Year 2015

# SmartCrawler: A Two-stage Crawler for Efficiently Harvesting Deep-Web Interfaces

Feng Zhao, Jingyu Zhou, Chang Nie, Heqing Huang, Hai Jin

**Abstract**—As deep web grows at a very fast pace, there has been increased interest in techniques that help efficiently locate deep-web interfaces. However, due to the large volume of web resources and the dynamic nature of deep web, achieving wide coverage and high efficiency is a challenging issue. We propose a two-stage framework, namely *SmartCrawler*, for efficient harvesting deep web interfaces. In the first stage, *SmartCrawler* performs site-based searching for center pages with the help of search engines, avoiding visiting a large number of pages. To achieve more accurate results for a focused crawl, *SmartCrawler* ranks websites to prioritize highly relevant ones for a given topic. In the second stage, *SmartCrawler* achieves fast in-site searching by excavating most relevant links with an adaptive link-ranking. To eliminate bias on visiting some highly relevant links in hidden web directories, we design a link tree data structure to achieve wider coverage for a website. Our experimental results on a set of representative domains show the agility and accuracy of our proposed crawler framework, which efficiently retrieves deep-web interfaces from large-scale sites and achieves higher harvest rates than other crawlers.

**Index Terms**—Deep web, two-stage crawler, feature selection, ranking, adaptive learning

## 1 Introduction

The *deep* (or *hidden*) web refers to the contents lie behind searchable web interfaces that cannot be indexed by searching engines. Based on extrapolations from a study done at University of California, Berkeley, it is estimated that the deep web contains approximately 91,850 terabytes and the surface web is only about 167 terabytes in 2003 [1]. More recent studies estimated that 1.9 zettabytes were reached and 0.3 zettabytes were consumed worldwide in 2007 [2], [3]. An IDC report estimates that the total of all digital data created, replicated, and consumed will reach 6 zettabytes in 2014 [4]. A significant portion of this huge amount of data is estimated to be stored as structured or relational data in web databases — deep web makes up about 96% of all the content on the Internet, which is 500-550 times larger than the surface web [5], [6]. These data contain a vast amount of valuable information and entities such as Infomine [7], Clusty [8], BooksInPrint [9] may be interested in building an index of the deep web sources in a given domain (such as book). Because these entities cannot access the proprietary web indices of search engines (e.g., Google and Baidu), there is a need for an efficient crawler that is able to accurately and quickly explore the deep web databases.

It is challenging to locate the deep web databases, because they are not registered with any search engines, are usually sparsely distributed, and keep constantly changing. To address this problem, previous work has proposed two types of crawlers, *generic crawlers* and *focused crawlers*. Generic crawlers [10], [11], [12], [13], [14] fetch all searchable forms and cannot focus on a specific topic. Focused crawlers such as Form-Focused Crawler (FFC) [15] and Adaptive Crawler for Hidden-web Entries (ACHE) [16] can automatically search online databases on a specific topic. FFC is designed with link, page, and form classifiers for focused crawling of web forms, and is extended by ACHE with additional components for form filtering and adaptive link learner. The link classifiers in these crawlers play a pivotal role in achieving higher crawling efficiency than the best-first crawler [17]. However, these link classifiers are used to predict the distance to the page containing searchable forms, which is difficult to estimate, especially for the delayed benefit links (links eventually lead to pages with forms). As a result, the crawler can be inefficiently led to pages without targeted forms.

Besides efficiency, quality and coverage on relevant deep web sources are also challenging. Crawler must produce a large quantity of high-quality results from the most relevant content sources [15], [16], [18], [19], [20], [21]. For assessing source quality, SourceRank ranks the results from the selected sources by computing the agreement between them [20], [21]. When selecting a relevant subset from the available content sources, FFC and ACHE prioritize links that bring immediate return (links directly point to pages con-

- *Feng Zhao, Change Nie and Hai Jin are with the Department of Computer Science and Technolgoy, HuaZhong University of Science and Technology, Wuhan, China.*
  *E-mail: zhaof@hust.edu.cn, changer328@gmail.com, hjin@hust.edu.cn*
- *Jingyu Zhou is with Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China.*
  *E-mail: zhou-jy@cs.sjtu.edu.cn*
- *Heqing Huang is with Department of Computer Science and Engineering, The Pennsylvania State University University Park, Pennsylvania, USA.*
  *E-mail: hhuang@cse.psu.edu*

taining searchable forms) and delayed benefit links. But the set of retrieved forms is very heterogeneous. For example, from a set of representative domains, on average only 16% of forms retrieved by FFC are relevant [15], [16]. Furthermore, little work has been done on the source selection problem when crawling more content sources [19], [22]. Thus it is crucial to develop smart crawling strategies that are able to quickly discover relevant content sources from the deep web as much as possible.

In this paper, we propose an effective deep web harvesting framework, namely *SmartCrawler*, for achieving both wide coverage and high efficiency for a focused crawler. Based on the observation that deep websites usually contain a few searchable forms and most of them are within a depth of three [23], [10], our crawler is divided into two stages: *site locating* and *in-site exploring*. The site locating stage helps achieve wide coverage of sites for a focused crawler, and the in-site exploring stage can efficiently perform searches for web forms within a site. Our main contributions are:

- We propose a novel two-stage framework to address the problem of searching for hidden-web resources. Our site locating technique employs a *reverse searching* technique (e.g., using Google's "link:" facility to get pages pointing to a given link) and incremental two-level site prioritizing technique for unearthing relevant sites, achieving more data sources. During the in-site exploring stage, we design a link tree for balanced link prioritizing, eliminating bias toward webpages in popular directories.
- We propose an adaptive learning algorithm that performs online feature selection and uses these features to automatically construct link rankers. In the site locating stage, high relevant sites are prioritized and the crawling is focused on a topic using the contents of the root page of sites, achieving more accurate results. During the in-site exploring stage, relevant links are prioritized for fast in-site searching.

We have performed an extensive performance evaluation of *SmartCrawler* over real web data in 12 representative domains and compared with ACHE [16] and a site-based crawler. Our evaluation shows that our crawling framework is very effective, achieving substantially higher harvest rates than the state-of-the-art ACHE crawler. The results also show the effectiveness of the reverse searching and adaptive learning.

The rest of the paper is organized as follows. We start in Section 2 by discussing related work. Section 3 presents the design of our two-stage *SmartCrawler* and Section 4 discusses feature selection and ranking. Section 5 reports our experimental results. Finally, Section 6 concludes the paper with future work.

## 2 Related Work

To leverage the large volume information buried in deep web, previous work has proposed a number of techniques and tools, including deep web understanding and integration [10], [24], [25], [26], [27], hidden-web crawlers [18], [28], [29], and deep web samplers [30], [31], [32]. For all these approaches, the ability to crawl deep web is a key challenge. Olston and Najork systematically present that crawling deep web has three steps: locating deep web content sources, selecting relevant sources and extracting underlying content [19]. Following their statement, we discuss the two steps closely related to our work as below.

**Locating deep web content sources**. A recent study shows that the harvest rate of deep web is low — only 647,000 distinct web forms were found by sampling 25 million pages from the Google index (about 2.5%) [27], [33]. Generic crawlers are mainly developed for characterizing deep web and directory construction of deep web resources, that do not limit search on a specific topic, but attempt to fetch all searchable forms [10], [11], [12], [13], [14]. The Database Crawler in the MetaQuerier [10] is designed for automatically discovering query interfaces. Database Crawler first finds root pages by an IP-based sampling, and then performs shallow crawling to crawl pages within a web server starting from a given root page. The IP-based sampling ignores the fact that one IP address may have several virtual hosts [11], thus missing many websites. To overcome the drawback of IP-based sampling in the Database Crawler, Denis et al. propose a stratified random sampling of hosts to characterize national deep web [13], using the Host-graph provided by the Russian search engine Yandex. I-Crawler [14] combines pre-query and post-query approaches for classification of searchable forms.

**Selecting relevant sources**. Existing hidden web directories [34], [8], [7] usually have low coverage for relevant online databases [23], which limits their ability in satisfying data access needs [35]. Focused crawler is developed to visit links to pages of interest and avoid links to off-topic regions [17], [36], [15], [16]. Soumen et al. describe a best-first focused crawler, which uses a page classifier to guide the search [17]. The classifier learns to classify pages as topic-relevant or not and gives priority to links in topic relevant pages. However, a focused best-first crawler harvests only 94 movie search forms after crawling 100,000 movie related pages [16]. An improvement to the best-first crawler is proposed in [36], 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 gives its choice as feedback so that the apprentice can learn the features of good links and prioritize links in the frontier. The FFC [15] and ACHE [16] are focused crawlers used for searching

interested deep web interfaces. The FFC contains three classifiers: a page classifier that scores the relevance of retrieved pages with a specific topic, a link classifier that prioritizes the links that may lead to pages with searchable forms, and a form classifier that filters out non-searchable forms. ACHE improves FFC with an adaptive link learner and automatic feature selection. SourceRank [20], [21] assesses the relevance of deep web sources during retrieval. Based on an agreement graph, SourceRank calculates the stationary visit probability of a random walk to rank results.

Different from the crawling techniques and tools mentioned above, *SmartCrawler* is a domain-specific crawler for locating relevant deep web content sources. *SmartCrawler* targets at deep web interfaces and employs a two-stage design, which not only classifies sites in the first stage to filter out irrelevant websites, but also categorizes searchable forms in the second stage. Instead of simply classifying links as relevant or not, *SmartCrawler* first ranks sites and then prioritizes links within a site with another ranker.

## 3 Design

### 3.1 Two-Stage Architecture

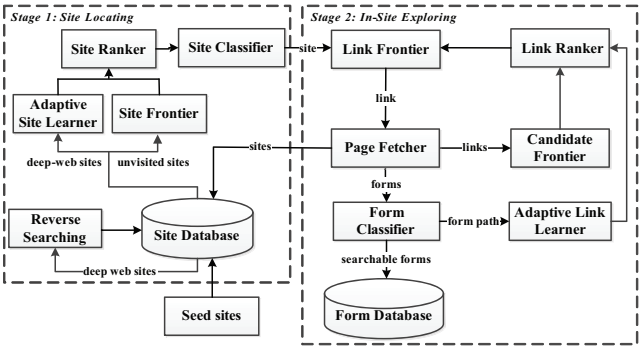



Fig. 1: The two-stage architecture of *SmartCrawler*.

To efficiently and effectively discover deep web data sources, *SmartCrawler* is designed with a two-stage architecture, *site locating* and *in-site exploring*, as shown in Figure 1. The first site locating stage finds the most relevant site for a given topic, and then the second in-site exploring stage uncovers searchable forms from the site.

Specifically, the site locating stage starts with a seed set of sites in a site database. Seeds sites are candidate sites given for *SmartCrawler* to start crawling, which begins by following URLs from chosen seed sites to explore other pages and other domains. When the number of unvisited URLs in the database is less than a threshold during the crawling process, *SmartCrawler* performs "reverse searching" of known deep web sites for *center pages* (highly ranked pages that have many links to other domains) and feeds these pages back to the site database. `Site Frontier` fetches homepage URLs from the site database, which are ranked by `Site Ranker` to prioritize highly relevant sites. The `Site Ranker` is improved during crawling by an `Adaptive Site Learner`, which adaptively learns from features of deep-web sites (web sites containing one or more searchable forms) found. To achieve more accurate results for a focused crawl, `Site Classifier` categorizes URLs into relevant or irrelevant for a given topic according to the homepage content.

After the most relevant site is found in the first stage, the second stage performs efficient in-site exploration for excavating searchable forms. Links of a site are stored in `Link Frontier` and corresponding pages are fetched and embedded forms are classified by `Form Classifier` to find searchable forms. Additionally, the links in these pages are extracted into `Candidate Frontier`. To prioritize links in `Candidate Frontier`, *SmartCrawler* ranks them with `Link Ranker`. Note that site locating stage and in-site exploring stage are mutually intertwined. When the crawler discovers a new site, the site's URL is inserted into the `Site Database`. The `Link Ranker` is adaptively improved by an `Adaptive Link Learner`, which learns from the URL path leading to relevant forms.

### 3.2 Site Locating

The site locating stage finds relevant sites for a given topic, consisting of site collecting, site ranking, and site classification.

#### 3.2.1 Site Collecting

The traditional crawler follows all newly found links. In contrast, our *SmartCrawler* strives to minimize the number of visited URLs, and at the same time maximizes the number of deep websites. To achieve these goals, using the links in downloaded webpages is not enough. This is because a website usually contains a small number of links to other sites, even for some large sites. For instance, only 11 out of 259 links from webpages of aaronbooks.com pointing to other sites; amazon.com contains 54 such links out of a total of 500 links (many of them are different language versions, e.g., amazon.de). Thus, finding out-of-site links from visited webpages may not be enough for the `Site Frontier`. In fact, our experiment in Section 5.3 shows that the size of `Site Frontier` may decrease to zero for some sparse domains.

To address the above problem, we propose two crawling strategies, *reverse searching* and *incremental two-level site prioritizing*, to find more sites.

- **Reverse searching.**
  The idea is to exploit existing search engines, such as Google, Baidu, Bing etc., to find center pages of unvisited sites. This is possible because search engines rank webpages of a site and center

pages tend to have high ranking values. Algorithm 1 describes the process of reverse searching. A reverse search is triggered:

- When the crawler bootstraps.
- When the size of site frontier decreases to a pre-defined threshold.

We randomly pick a known deep website or a seed site and use general search engine's facility to find center pages and other relevant sites, Such as Google's "link:" , Bing's "site:", Baidu's "domain:". For instance, [link:www.google.com] will list web pages that have links pointing to the Google home page. In our system, the result page from the search engine is first parsed to extract links. Then these pages are downloaded and analyzed to decide whether the links are relevant or not using the following heuristic rules:

- If the page contains related searchable forms, it is relevant.
- If the number of seed sites or fetched deep-web sites in the page is larger than a user-defined threshold, the page is relevant.

Finally, the found relevant links are output. In this way, we keep `Site Frontier` with enough sites.

**Algorithm 1:** Reverse searching for more sites.

```
input : seed sites and harvested deep websites
output: relevant sites
while # of candidate sites less than a threshold do
    // pick a deep website
    site = getDeepWebSite(siteDatabase, seedSites)
    resultPage = reverseSearch(site)
    links = extractLinks(resultPage)
    foreach link in links do
        page = downloadPage(link)
        relevant = classify(page)
        if relevant then
            relevantSites = extractUnvisitedSite(page)
            Output relevantSites
        end
    end
end
```

- **Incremental site prioritizing.**
To make crawling process resumable and achieve broad coverage on websites, an incremental site prioritizing strategy is proposed. The idea is to record learned patterns of deep web sites and form paths for incremental crawling. First, the prior knowledge (information obtained during past crawling, such as deep websites, links with searchable forms, etc.) is used for initializing `Site Ranker` and `Link Ranker`. Then, unvisited sites are assigned to `Site Frontier` and are prioritized by `Site Ranker`, and visited sites are added to fetched site list. The detailed incremental site prioritizing process is described in Algorithm 2.
While crawling, *SmartCrawler* follows the out-of-site links of relevant sites. To accurately classify out-of-site links, `Site Frontier` utilizes two queues to save unvisited sites. The high priority queue is for out-of-site links that are classified as relevant by `Site Classifier` and are judged by `Form Classifier` to contain searchable forms. The low priority queue is for out-of-site links that only judged as relevant by `Site Classifier`. For each level, `Site Ranker` assigns relevant scores for prioritizing sites. The low priority queue is used to provide more candidate sites. Once the high priority queue is empty, sites in the low priority queue are pushed into it progressively.

**Algorithm 2:** Incremental Site Prioritizing.

```
input : siteFrontier
output: searchable forms and out-of-site links
HQueue=SiteFrontier.CreateQueue(HighPriority)
LQueue=SiteFrontier.CreateQueue(LowPriority)
while siteFrontier is not empty do
    if HQueue is empty then
        HQueue.addAll(LQueue)
        LQueue.clear()
    end
    site = HQueue.poll()
    relevant = classifySite(site)
    if relevant then
        performInSiteExploring(site)
        Output forms and OutOfSiteLinks
        siteRanker.rank(OutOfSiteLinks)
        if forms is not empty then
            HQueue.add (OutOfSiteLinks)
        end
        else
            LQueue.add(OutOfSiteLinks)
        end
    end
end
```

### 3.2.2 Site Ranker

Once the `Site Frontier` has enough sites, the challenge is how to select the most relevant one for crawling. In *SmartCrawler*, `Site Ranker` assigns a score for each unvisited site that corresponds to its relevance to the already discovered deep web sites. The ranking mechanism is discussed in Section 4.3.

#### 3.2.3 Site Classifier

After ranking `Site Classifier` categorizes the site as topic relevant or irrelevant for a focused crawl, which is similar to page classifiers in FFC [15] and ACHE [16]. If a site is classified as topic relevant, a site crawling process is launched. Otherwise, the site is ignored and a new site is picked from the frontier.

In *SmartCrawler*, we determine the topical relevance of a site based on the contents of its homepage. When a new site comes, the homepage content of the site is extracted and parsed by removing stop words and stemming. Then we construct a feature vector for the site (Section 4.1) and the resulting vector is fed into a Naïve Bayes classifier to determine if the page is topic-relevant or not.

### 3.3 In-Site Exploring

Once a site is regarded as topic relevant, in-site exploring is performed to find searchable forms. The goals are to quickly harvest searchable forms and to cover web directories of the site as much as possible. To achieve these goals, in-site exploring adopts two crawling strategies for high efficiency and coverage. Links within a site are prioritized with `Link Ranker` and `Form Classifier` classifies searchable forms.

#### 3.3.1 Crawling Strategies

Two crawling strategies, *stop-early* and *balanced link prioritizing*, are proposed to improve crawling efficiency and coverage.

- **Stop-early.**
  Previous work [18] observed that 72% interfaces and 94% web databases are found within the depth of three. Thus, in-site searching is performed in breadth-first fashion to achieve broader coverage of web directories. Additionally, in-site searching employs the following stopping criteria to avoid unproductive crawling:

  **SC1:** The maximum depth of crawling is reached.
  **SC2:** The maximum crawling pages in each depth are reached.
  **SC3:** A predefined number of forms found for each depth is reached.
  **SC4:** If the crawler has visited a predefined number of pages without searchable forms in one depth, it goes to the next depth directly.
  **SC5:** The crawler has fetched a predefined number of pages in total without searchable forms.

  SC1 limits the maximum crawling depth. Then for each level we set several stop criteria (SC2, SC3, SC4). A global one (SC5) restricts the total pages of unproductive crawling.
- **Balanced link prioritizing.**
  The simple breadth-first visit of links is not efficient, whose results are in omission of highly relevant links and incomplete directories visit, when combined with above stop-early policy. We solve this problem by prioritizing highly relevant links with link ranking. However, link ranking may introduce bias for highly relevant links in certain directories. Our solution is to build a link-tree for a balanced link prioritizing.

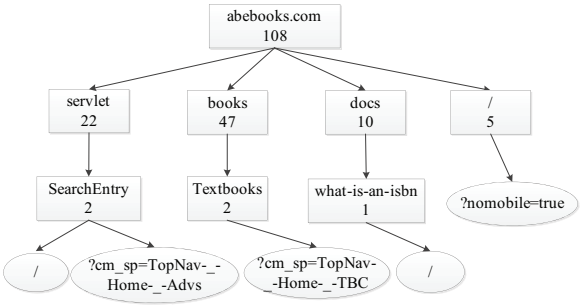



Fig. 2: Part of the link tree extracted from the homepage of http://www.abebooks.com, where ellipses represent leaf nodes and the number in a rectangle denotes the number of leaf nodes in its decedents.

Figure 2 illustrates an example of a link tree constructed from the homepage of http://www.abebooks.com. Internal nodes of the tree represent directory paths. In this example, `servlet` directory is for dynamic request; `books` directory is for displaying different catalogs of books; and `docs` directory is for showing help information. Generally each directory usually represents one type of files on web servers and it is advantageous to visit links in different directories. For links that only differ in the query string part, we consider them as the same URL.

Because links are often distributed unevenly in server directories, prioritizing links by the relevance can potentially bias toward some directories. For instance, the links under `books` might be assigned a high priority, because "book" is an important feature word in the URL. Together with the fact that most links appear in the `books` directory, it is quite possible that links in other directories will not be chosen due to low relevance score. As a result, the crawler may miss searchable forms in those directories.

To avoid such bias and cover more directories, we merge directories with few links. Specifically, let $V$ denote the maximum number of links that can be visited for a site; and let $N$ denote the number links for a link tree. We compute $\gamma = V/N$. If $\gamma \geq 1$, the crawler visits all links. Otherwise, for each node we compute the number of links to be visited as $C = M \times \gamma$, where $M$ is the number of links of this node. Then for each level we merge those directories with $C$ less than one into a new node. Figure 3 is the merged link tree derived from the above example in Figure 2. The leftmost nodes in the lower two levels are merged

nodes and the number of visited links are evenly distributed among sibling nodes. Finally, highly relevant links are selected from each directory for fetching.

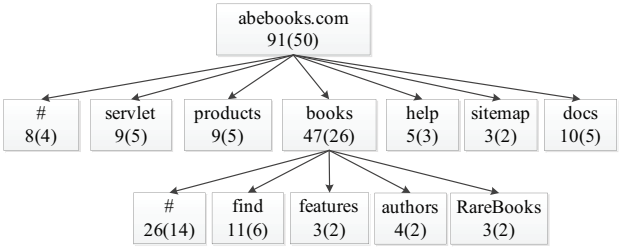



Fig. 3: The merged link tree of the homepage of http://www.abebooks.com/, where symbol # represents the merged node. The two numbers of each internal node represent the count of links and the actual visiting count under the node.

#### 3.3.2 Link Ranker

`Link Ranker` prioritizes links so that *SmartCrawler* can quickly discover searchable forms. A high relevance score is given to a link that is most similar to links that directly point to pages with searchable forms (see Section 4.3).

#### 3.3.3 Form Classifier

Classifying forms aims to keep form focused crawling, which filters out non-searchable and irrelevant forms. For instance, an airfare search is often co-located with rental car and hotel reservation in travel sites. For a focused crawler, we need to remove off-topic search interfaces.

*SmartCrawler* adopts the HIFI strategy to filter relevant searchable forms with a composition of simple classifiers [37]. HIFI consists of two classifiers, a searchable form classifier (SFC) and a domain-specific form classifier (DSFC). SFC is a domain-independent classifier to filter out non-searchable forms by using the structure feature of forms. DSFC judges whether a form is topic relevant or not based on the text feature of the form, that consists of domain-related terms. The strategy of partitioning the feature space allows selection of more effective learning algorithms for each feature subset. In our implementation, SFC uses decision tree based C4.5 algorithm [15], [38] and DSFC employs SVM [39]. The details of these classifiers are out of the scope of this paper (see [37] for details).

## 4 Feature Selection and Ranking

*SmartCrawler* encounters a variety of webpages during a crawling process and the key to efficiently crawling and wide coverage is ranking different sites and prioritizing links within a site. This section first discusses the online feature construction of feature space and adaptive learning process of *SmartCrawler*, and then describes the ranking mechanism.

### 4.1 Online Construction of Feature Space

In *SmartCrawler*, patterns of links to relevant sites and searchable forms are learned online to build both site and link rankers. The ability of online learning is important for the crawler to avoid biases from initial training data and adapt to new patterns.

The feature space of deep web sites ($FSS$) is defined as:

$$FSS = \{U, A, T\}, \tag{1}$$

where $U$, $A$, $T$ are vectors corresponding to the feature context of URL, anchor, and text around URL of the deep web sites.

The feature space of links of a site with embedded forms ($FSL$) is defined as:

$$FSL = \{P, A, T\}, \tag{2}$$

where $A$ and $T$ are the same as defined in $FSS$ and $P$ is the vector related to the path of the URL, since all links of a specific site have the same domain.

Each feature context can be represented as a vector of terms with a specific weight. The weight $w$ of term $t$ can be defined as:

$$w_{t,d} = 1 + \log tf_{t,d}, \tag{3}$$

where $tf_{t,d}$ denotes the frequency of term $t$ appears in document $d$, and $d$ can be $U$, $P$, $A$, or $T$. We use term frequency (TF) as feature weight for its simplicity and our experience shows that TF works well for our application.

To automatically construct $FSS$ and $FSL$ online, we use the following feature selection method using top-$k$ features:

- When computing a feature set for $P$, $A$, and $T$, words are first stemmed after removing stop words. Then the top-$k$ most frequent terms are selected as the feature set.
- When constructing a feature set for $U$, a partition method based on term frequency is used to process URLs, because URLs are not well structured. Firstly, the top-level domain of URL (e.g. com, co.uk) is excluded. Secondly, after stemming terms, the most frequent $k$ terms are selected from the URL features. Thirdly, if a term in the frequent set appears as a substring of the URL, the URL is split by the frequent term. For example, "abebooks" and "carbuyer" are terms that appear in URL of the book and auto domains, as "book" and "car" are frequent terms, the URL is split into "abe", "book" and "car", "buyer", respectively.

### 4.2 Adaptive Learning

*SmartCrawler* has an adaptive learning strategy that updates and leverages information collected successfully during crawling. As shown in Figure 1, both `Site Ranker` and `Link Ranker` are controlled by

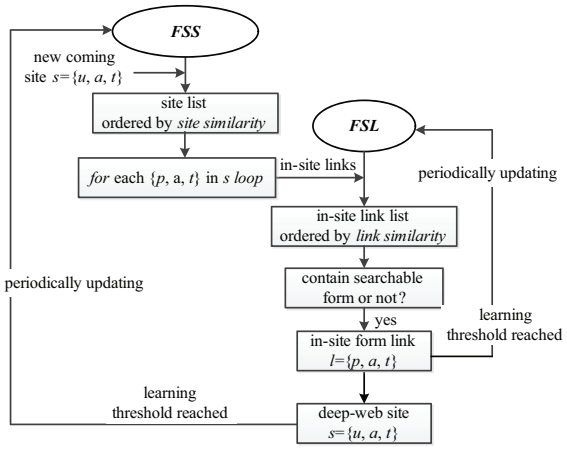



Fig. 4: Adaptive learning process in *SmartCrawler*.

adaptive learners. Periodically, $FSS$ and $FSL$ are adaptively updated to reflect new patterns found during crawling. As a result, `Site Ranker` and `Link Ranker` are updated. Finally, `Site Ranker` re-ranks sites in `Site Frontier` and `Link Ranker` updates the relevance of links in `Link Frontier`.

Figure 4 illustrates the adaptive learning process that is invoked periodically. For instance, the crawler has visited a pre-defined number of deep web sites or fetched a pre-defined number of forms. In the implementation, the learning thresholds are 50 new deep websites or 100 searchable forms. When a site crawling is completed, feature of the site is selected for updating $FSS$ if the site contains relevant forms. During in-site exploring, features of links containing new forms are extracted for updating $FSL$.

### 4.3 Ranking Mechanism

#### 4.3.1 Site Ranking

*SmartCrawler* ranks site URLs to prioritize potential deep sites of a given topic. To this end, two features, *site similarity* and *site frequency*, are considered for ranking. Site similarity measures the topic similarity between a new site and known deep web sites. Site frequency is the frequency of a site to appear in other sites, which indicates the popularity and authority of the site — a high frequency site is potentially more important. Because seed sites are carefully selected, relatively high scores are assigned to them.

Given the homepage URL of a new site $s = \{U_s, A_s, T_s\}$, the site similarity to known deep web sites $FSS$, can be defined as follows:

$$ST(s) = Sim(U, U_s) + sim(A, A_s) + sim(T, T_s), \quad (4)$$

where function $Sim$ scores the similarity of the related feature between $s$ and known deep web sites. The function $Sim(\cdot)$ is computed as the cosine similarity between two vectors $V_1$ and $V_2$:

$$Sim(V_1, V_2) = \frac{V_1 \cdot V_2}{|V_1| \times |V_2|}. \quad (5)$$

The site frequency measures the number of times a site appears in other sites. In particular, we consider the appearance in known deep sites to be more important than other sites. The site frequency is defined as:

$$SF(s) = \sum_{known\ sites\ list} I_i, \quad (6)$$

where $I_i = 1$ if $s$ appeared in known deep web sites, otherwise $I_i = 0$.

Finally, the rank of a new coming site $s$ is a function of site similarity and site frequency, and we use a simple linear combination of these two features:

$$Rank(s) = \alpha \times ST(s) + (1 - \alpha) \times \log(1 + SF(s)), \quad (7)$$

where $0 \leq \alpha \leq 1$.

#### 4.3.2 Link Ranking

For prioritizing links of a site, the link similarity is computed similarly to the site similarity described above. The difference includes: 1) link prioritizing is based on the feature space of links with searchable forms ($FSL$); 2) for URL feature $U$, only path part is considered since all links have the same domain; and 3) the frequency of links is not considered in link ranking.

Given a new link $l = \{P_l, A_l, T_l\}$, the link similarity to the feature space of known links with searchable forms $FSL$ is defined as:

$$LT(l) = Sim(P, P_l) + sim(A, A_l) + sim(T, T_l), \quad (8)$$

where function $Sim(\cdot)$ (Equation 5) scores the similarity of the related feature between $l$ and the known in-site links with forms.

Finally, we use the link similarity for ranking different links.

## 5 Evaluation

We have performed an extensive performance evaluation of our crawling framework over real web data in 12 representative domains. Our goals include:

- evaluating the efficiency of *SmartCrawler* in obtaining relevant deep websites and searchable forms,
- analyzing the effectiveness of site collecting, and
- assessing the performance of adaptive learning.

### 5.1 Experimental Setup

We have implemented *SmartCrawler* in Java and evaluated our approach over 12 different domains described in Table 1.

To evaluate the performance of our crawling framework, we compare *SmartCrawler* to the SCDI (site-based crawler for deep web interfaces) and ACHE [16]:

TABLE 1: Twelve domains for experiments.

| Domain | Description |
|---|---|
| Airfare | airfare search |
| Auto | used cars search |
| Book | books search |
| Hotel | hotel search |
| Job | job search |
| Movie | movie titles and DVDs search |
| Music | music CDs search |
| Rental | car rental search |
| Apartment | apartment search |
| Route | map and airline search |
| Product | household product search |
| People | sports stars search |

- **ACHE.** We implemented the ACHE, which is an adaptive crawler for harvesting hidden-web entries with offline-online learning to train link classifiers. We adapt the similar stopping criteria as *SmartCrawler*, i.e., the maximum visiting pages and a predefined number of forms for each site.
- **SCDI.** We designed an experimental system similar to *SmartCrawler*, named SCDI, which shares the same stopping criteria with *SmartCrawler*. Different from *SmartCrawler*, SCDI follows the out-of-site links of relevant sites by site classifier without employing incremental site prioritizing strategy. It also does not employ reverse searching for collecting sites and use the adaptive link prioritizing strategy for sites and links.
- ***SmartCrawler.*** *SmartCrawler* is our proposed crawler for harvesting deep web interfaces. Similar to ACHE, *SmartCrawler* uses an offline-online learning strategy, with the difference that *SmartCrawler* leverages learning results for site ranking and link ranking. During in-site searching, more stop criteria are specified to avoid unproductive crawling in *SmartCrawler*.

In our implementation of *SmartCrawler*, `Site Classifier` uses the Naïve Bayes classifier from Weka [40], which is trained with samples from the topic taxonomy of the Dmoz directory [41]. `Form Classifier` is trained with dataset extended from TEL-8 dataset of the UIUC repository [42]. The TEL-8 dataset contains 447 deep web sources with 477 query interfaces, because a source may contain multiple interfaces. We construct our dataset by collecting 216 actively searchable forms from the UIUC repository [42], and manually gathering 473 non-searchable forms for the negative examples. Our final dataset includes 689 query forms covering 12 domains shown in Table 1.

### 5.2 Crawler Efficiency

In this experiment, we compare the efficiency of ACHE, SCDI, and *SmartCrawler* for fetching 100,000 pages from different domains. The results of the numbers of retrieved relevant deep websites and searchable forms are illustrated in Figure 5 and Figure 6, respectively.

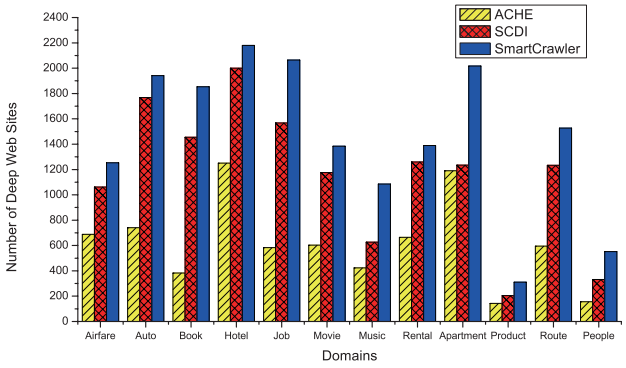



Fig. 5: The numbers of relevant deep websites harvested by ACHE, SCDI and *SmartCrawler*.

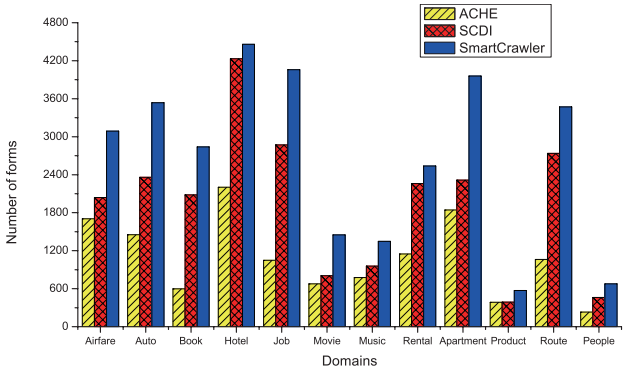



Fig. 6: The numbers of relevant forms harvested by ACHE, SCDI and *SmartCrawler*.

Figure 5 shows that *SmartCrawler* finds more relevant deep websites than ACHE and SCDI for all domains. Figure 6 illustrates that *SmartCrawler* consistently harvests more relevant forms than both ACHE and SCDI. SCDI is significantly better than ACHE because our two-stage framework can quickly discover relevant sites rather than being trapped by irrelevant sites. By prioritizing sites and in-site links, *SmartCrawler* harvests more deep websites than SCDI, because potential deep websites are visited earlier and unproductive links in in-site searching are avoided. Most of bars present a similar trend in Figure 5 and Figure 6, because the harvested sites are often proportional to harvested searchable forms.

To better understand the efficiency of three crawlers, Figure 7 illustrates the number of relevant forms harvested during the crawling process. We can observe that *SmartCrawler* and SCDI consistently harvest more relevant forms than ACHE, because our two-stage approach prioritizes more relevant sites, avoiding visiting many pages of irrelevant sites. Additionally, *SmartCrawler* and SCDI use more stopping criteria such that they can visit less pages for a site. With the increase of visited pages, *SmartCrawler* keeps a higher crawling rate of relevant forms than SCDI, mainly due to site ranking and link ranking for

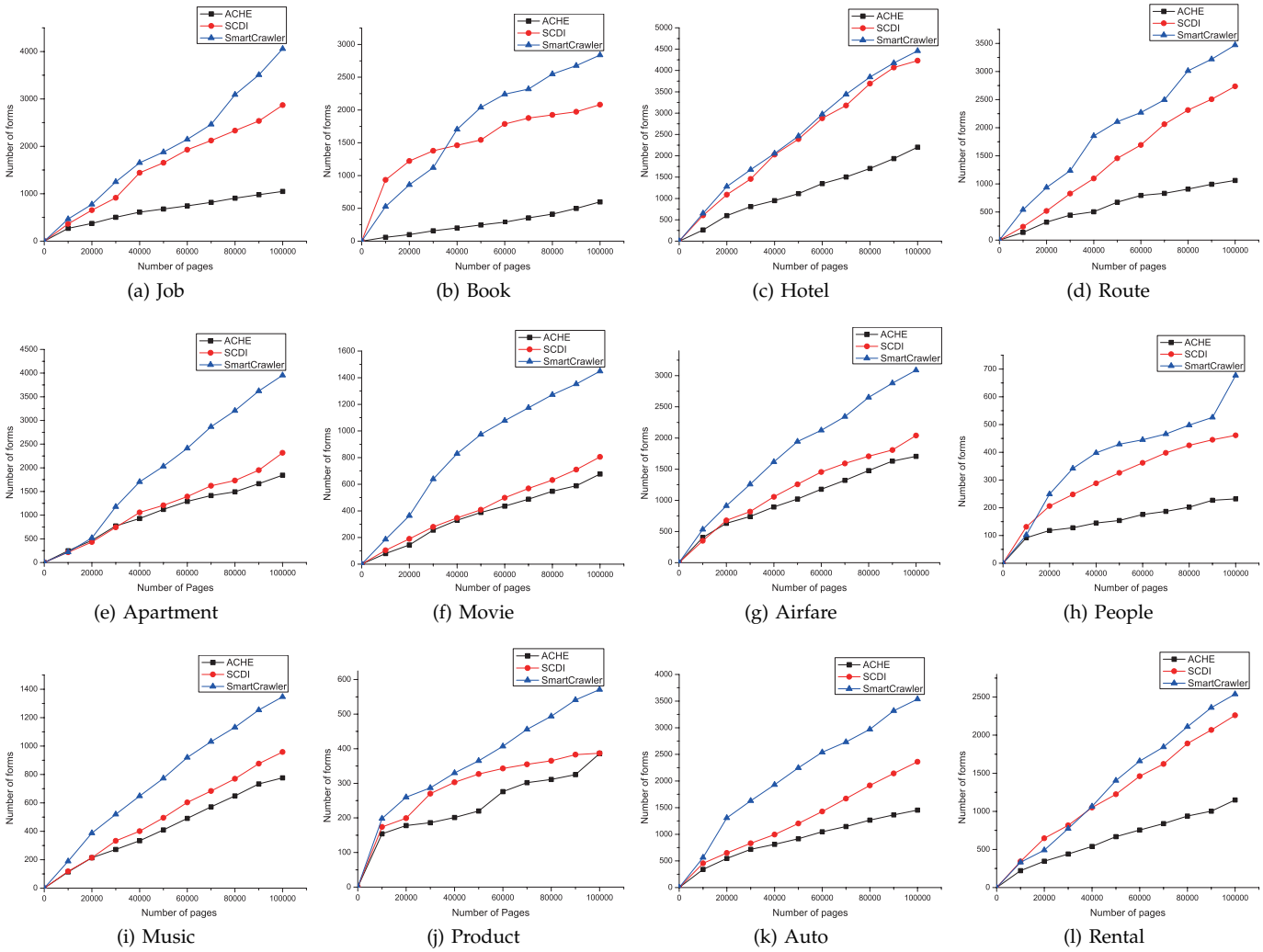


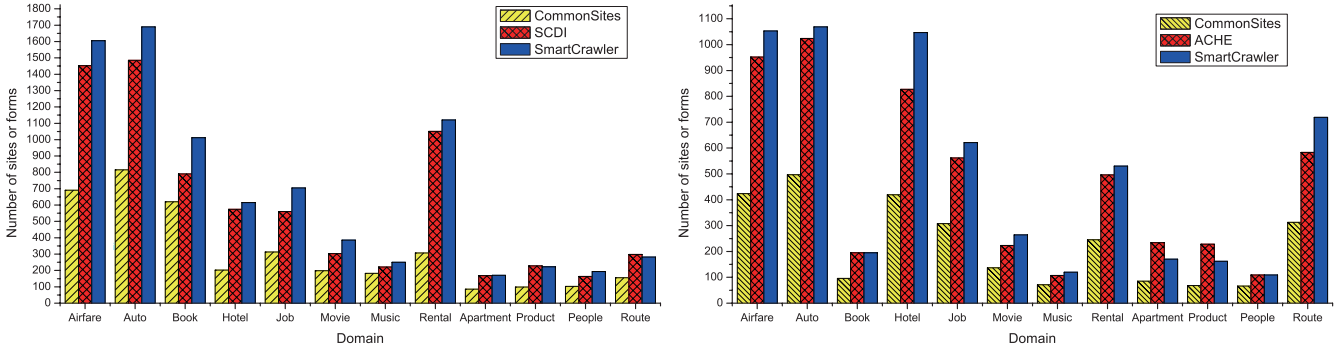

Fig. 7: The comparison of harvest rate of forms during the crawl for three crawlers.

quickly visiting relevant pages. We further compare the running time and achieved searchable forms on twelve online database domains. Table 2 illustrates the effectiveness of proposed strategy in terms of searchable forms obtained for ACHE and *SmartCrawler*. With the proposed strategies, *SmartCrawler* can avoid spending too much time crawling unproductive sites. Using the saved time, *SmartCrawler* can visit more relevant web directories and get many more relevant searchable forms.

We also compared the number of forms harvested by *SmartCrawler*, ACHE and SCDI under common sites (the sites that both SCDI and *SmartCrawler* or both ACHE and *SmartCrawler* accessed during crawling). The results of the number of common sites and searchable forms found in these common sites are showed in Figure 8. Since SCDI, ACHE and *SmartCrawler* shares the same 100 seed sites, the forms fetched in the same sites can reflect the effectiveness of proposed crawling strategies. Furthermore, SCDI and *SmartCrawler* shares the same stop criteria and form classifier, the forms fetched in the same sites can reflect the effectiveness of the balanced link prioritizing strategy. Figure 8 shows *SmartCrawler* can retrieve more searchable forms than ACHE and SCDI when

TABLE 2: The comparison of running time and number of searchable forms found for ACHE and *SmartCrawler*.

| Domain | Running Time | | Searchable Forms | |
|---|---|---|---|---|
| | ACHE | *SmartCrawler* | ACHE | *SmartCrawler* |
| Airfare | 7h59min | 6h59min | 1705 | 3087 |
| Auto | 8h11min | 6h32min | 1453 | 3536 |
| Book | 8h21min | 7h32min | 599 | 2838 |
| Job | 8h50min | 8h8min | 1048 | 4058 |
| Hotel | 8h37min | 6h54min | 2203 | 4459 |
| Movie | 10h9min | 6h26min | 677 | 1449 |
| Music | 7h59min | 6h29min | 776 | 1347 |
| Rental | 8h1min | 6h38min | 1149 | 2538 |
| Product | 7h50min | 6h29min | 386 | 571 |
| Apartment | 8h7min | 6h7min | 1844 | 3958 |
| Route | 8h0min | 6h36min | 1061 | 3471 |
| People | 8h3min | 6h43min | 232 | 677 |




(a) The Comparison of SCDI and *SmartCrawler*

(b) The Comparison of ACHE and *SmartCrawler*

Fig. 8: The number of common sites and forms in the common sites harvested by SCDI, ACHE and *SmartCrawler*.

searching the same set of sites in almost all domains.

TABLE 3: The accuracy of site classification, form classification and searchable forms classification in *SmartCrawler*.

| Domain | Site Classification | Form Classification | Searchable Forms Classification |
|---|---|---|---|
| Airfare | 92.08% | 94.79% | 95.25% |
| Auto | 92.93% | 97.52% | 94.48% |
| Book | 92.42% | 97.27% | 98.46% |
| Hotel | 98.00% | 90.25% | 97.82% |
| Job | 97.47% | 93.30% | 93.92% |
| Movie | 90.91% | 91.71% | 97.86% |
| Music | 93.07% | 90.20% | 98.47% |
| Rental | 97.00% | 90.07% | 97.88% |
| Apartment | 92.82% | 88.43% | 97.86% |
| Route | 91.13% | 90.18% | 97.62% |
| Product | 97.07% | 93.54% | 95.19% |
| People | 97.07% | 93.54% | 95.19% |

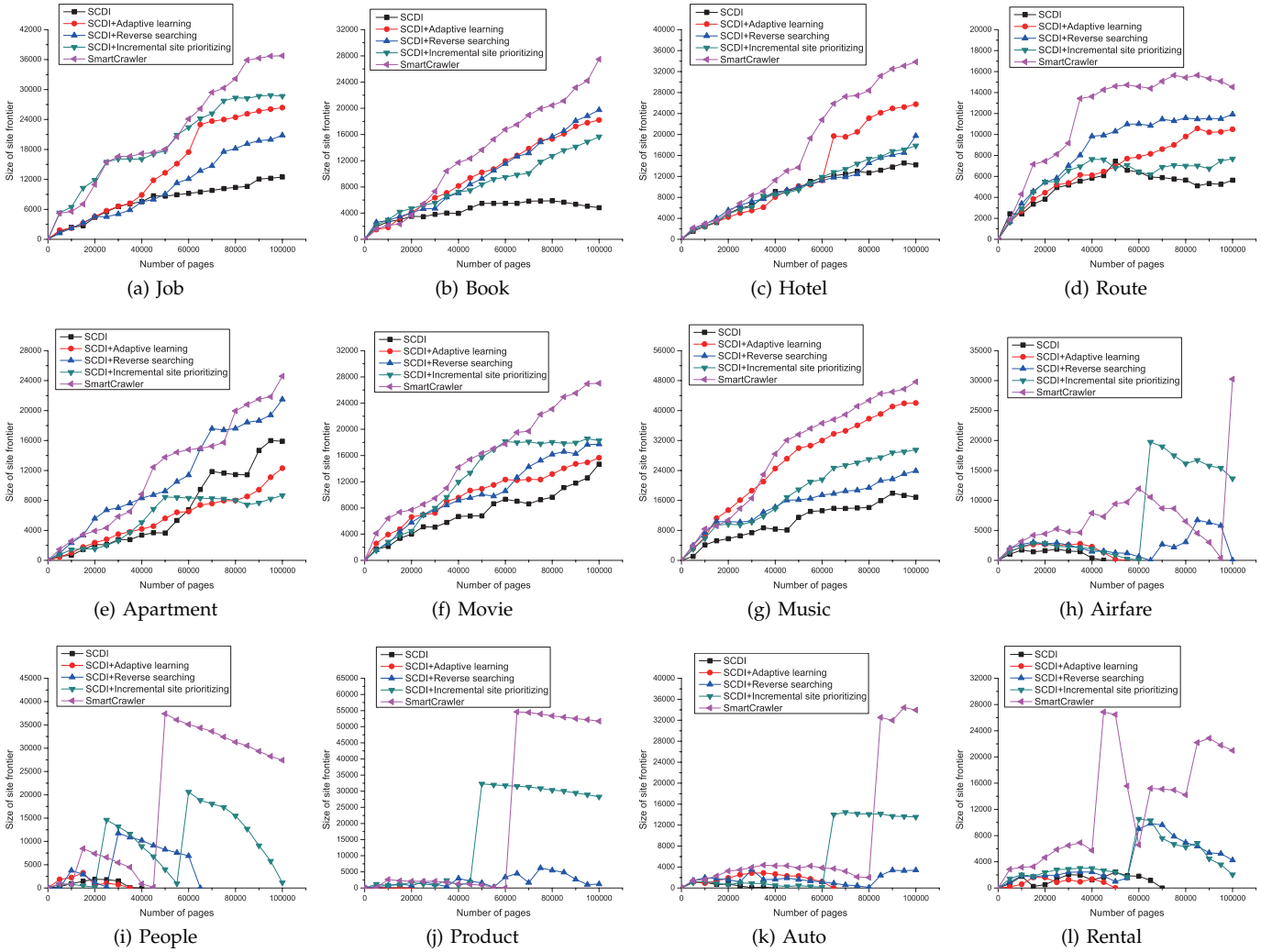
Moreover, to validate the effectiveness of `Site Classifier` and `Form Classifier`, we train our classifier with expanded TEL-8 dataset that covering twelve online database domains. The 10-fold cross validation is used to evaluate the accuracy of site classifier and C4.5 algorithm is used for filtering non-searchable forms. After filtering out non-searchable forms, DSFC of HIFI is used to judge whether the form is topic relevant or not. Both the text among the form tags and default values of textbox, radio control and checkbox are considered as basis for classification. Furthermore, we also write an additional script program to verify whether the discovered forms are searchable or not, which queries the form for answers. The accuracy of site classification, form classification and accuracy of searchable form are shown in Table 3. We also compared the accuracy of SFC for ACHE and *SmartCrawler* under the same seed sites and the same training dataset, *SmartCrawler* can achieve higher accuracy on finding searchable forms in total twelve domains. The average accuracy of *SmartCrawler* is 93.17%, while ACHE is 90.24%. This is because *SmartCrawler* avoids crawling unproductive forms.

### 5.3 Effectiveness of Site Collecting

This experiment studies the effectiveness of the site collecting mechanism in our *SmartCrawler*. A prime fact for harvesting searchable forms is that the *SmartCrawler* fetch links from the high priority queue of `Site Frontier`.

The proposed crawling strategy can mitigate the draining of `Site Frontier` for twelve online domains. Figure 9 compares the high priority queue sizes in `Site Frontier` of *SmartCrawler* (crawler employs all these strategies) and the approach combing SCDI with different strategies, such as reverse searching, adaptive learning, and incremental site prioritizing. Figure 9 shows that SCDI achieves the least queue size because it has not used any optimized strategies. When combined with a crawling strategy, the site numbers in high priority queue of `Site Frontier` increased. *SmartCrawler* is the aggregation of SCDI and all proposed strategies, achieving the maximum site sizes. For "job", "book", "hotel", "route", "apartment", "movie" and "music" domains, the sizes keep increasing. For "airfare", "people", "product", "auto" and "rental" domains, the queue size of SCDI initially increases and then gradually falls off because it does not have adequate sites for crawling. The reverse searching strategy, incremental site prioritizing strategy and adaptive learning strategy can separately increase the sites in working queue, so *SCDI+reverse searching*, *SCDI+incremental site prioritizing* and *SCDI+adaptive learning* are higher than SCDI. However, even combined with a single strategy, site numbers in high priority queue still fall off in "airfare", "people" domain. In contrast, *SmartCrawler* in these five domains keeps above zero and obtains the largest site sizes in twelve domains.

Take reverse searching strategy in Figure 9 as a special example, Table 4 illustrates the effectiveness of reverse searching in terms of center pages collected in a crawl. Reverse searching will be triggered when the




(a) Job (b) Book (c) Hotel (d) Route

(e) Apartment (f) Movie (g) Music (h) Airfare

(i) People (j) Product (k) Auto (l) Rental

Fig. 9: The Comparison of site frontier's high priority queue sizes during crawling with different strategies.

size of the `Site Frontier` is below the threshold, where a reverse searching thread will add sites in the center pages to the `Site Frontier`. Table 4 shows that reverse searching strategy is triggered many more times and gets more center pages in *SmartCrawler* than *SCDI+reverse searching*, because *SmartCrawler* can preferably utilize reverse searching strategy under the joint efforts of other proposed strategies.

TABLE 4: The comparison of the number of triggered times and center pages found for reverse searching.

| Domain | Times | | Center Pages | |
|---|---|---|---|---|
| | SCDI+RS | *SmartCrawler* | SCDI+RS | *SmartCrawler* |
| Airfare | 14 | 29 | 45 | 81 |
| Auto | 24 | 28 | 64 | 82 |
| Book | 1 | 3 | 1 | 0 |
| Job | 2 | 2 | 11 | 11 |
| Hotel | 2 | 40 | 2 | 153 |
| Movie | 1 | 21 | 4 | 34 |
| Music | 1 | 18 | 4 | 32 |
| Rental | 9 | 13 | 23 | 24 |
| Product | 35 | 22 | 20 | 18 |
| Apartment | 29 | 39 | 3 | 34 |
| Route | 2 | 7 | 2 | 18 |
| People | 6 | 9 | 10 | 15 |

### 5.4 Effectiveness of Adaptive Learning

The adaptive learning in *SmartCrawler* is critical for finding relevant sites and links during the crawl. To illustrate the effectiveness of our adaptive learning mechanism, Table 5 shows the top features of deep websites ($FSS$) and links with embedded forms ($FSL$) for the book domain. The table shows the results for initial 25 learning iterations.

Table 5 shows that after 10 learning iterations, important new terms related to library and university sites (e.g., "lib", "librari", "colleg" and "univers") are introduced to site features. After 25 iterations, the new terms such as "librari", "colleg", "univers" have the highest frequency and another new term "journal" is introduced. Because no library and university sites are provided in the seed set, this experiment demonstrates that our *SmartCrawler* can quickly learn

TABLE 5: The top features of deep websites and links with forms in the book domain. 25 iterations of learning are shown and the number in bracket denotes term frequency. The highlighted terms are important new features learnt during the crawling process.

| No. | Deep Website Features | | | Link Features | | |
|---|---|---|---|---|---|---|
| | URL | Anchor | Text | Path | Anchor | Text |
| 0 | (search,3) (book,3) (author,1) (index,1) | (advanc,1) (search,1) (book,1) (author,1) | (home,1) (search,1) (titl,1) (book,1) | (search,3) (book,3) (author,1) (index,1) | (advanc,1) (search,1) (book,1) (author,1) | (home,1) (search,1) (titl,1) (book,1) |
| 5 | (book,197) (search,3) **(librari,3)** (biblio,2) (alibri,2) | (book,91) (visit,78) (seller,77) (bookstor,9) **(univers,8)** | (book,135) (seller,78) (bookstor,20) (print,14) (onlin,13) | (search,46) (book,43) (advanc,9) (form,6) (index,4) | (search,103) (book,67) (advanc,65) (brows,29) (detail,6) (catalog,5) | (book,104) (search,52) (brows,39) (cart,32) (shop,30) (author,18) |
| 10 | (book,401) **(lib,6) (librari,5)** (search,3) (biblio,2) (alibri,2) | (bookstor,140) (book,134) (visit,78) (seller,77) **(colleg,68)** **(univers,56)** | (book,172) (bookstor,152) (seller,78) **(colleg,68)** **(univers,57)** | (search,95) (book,67) (index,24) (advanc,18) (shop,15) | (search,186) (advanc,120) (book,107) (brows,36) **(librari,34)** (textbook,19) | (book,174) (search,117) (brows,68) (textbook,61) (cart,57) (shop,57) |
| 15 | (book,588) **(lib,6) (librari,5)** (search,4) (ua,4) (rare,3) (biblio,2) | (book,208) (bookstor,157) (visit,79) (seller,77) **(colleg,74)** **(univers,65)** **(librari,29)** | (book,300) (bookstor,167) **(colleg,82)** (seller,78) **(univers,71)** (rare,47) (websit,47) **(librari,34)** | (search,411) (book,152) (shop,81) (index,76) (advanc,50) (cart,42) | (search,353) (advanc,238) (book,151) **(librari,59)** (brows,59) (catalog,30) (author,27) (textbook,20) | (search,255) (book,204) (brows,101) (shop,82) (cart,79) (textbook,63) (titl,62) **(librari,54)** |
| 20 | (book,642) (lib,6) (search,5) (librari,5) (ua,4) (store,3) | (book,231) (bookstor,161) **(librari,130)** **(univers,96)** (visit,80) **(colleg,80)** (seller,77) | (book,353) **(librari,182)** (bookstor,170) **(univers,108)** **(colleg,95)** (seller,79) | (search,487) (book,175) (index,138) (pg,127) (shop,83) (advanc,67) | (search,482) (advanc,331) (book,195) (brows,80) **(librari,74)** (author,44) (catalog,44) | (search,381) (book,233) (brows,112) (shop,104) (cart,100) (titl,92) **(librari,85)** |
| 25 | (book,774) (search,7) (lib,6) (catalog,5) (librari,5) (store,4) | (book,248) (bookstor,167) **(librari,155)** **(univers,106)** **(colleg,83)** (visit,81) | (book,390) **(librari,229)** (bookstor,175) **(univers,127)** **(colleg,100)** (seller,81) | (search,876) (book,257) (index,167) (advanc,138) (pg,127) (shop,103) | (search,714) (advanc,479) (book,218) **(librari,97)** (brows,85) (author,54) (catalog,50) | (search,635) (book,272) **(journal,181)** (author,151) (brows,133) (advanc,129) **(librari,118)** |

to prioritize related sites all by itself. In comparison, ACHE is only able to learn the library sites for the same number of visited pages. From Table 5, we can see that the top features tend to be stable, especially for the links. For example, the terms such as "book", "search", "advanc", and "shop" keep being frequent features in the URL feature set of links. In contrast, the URL feature set of deep websites changes more frequently, because domains in URLs have more variability.

Table 6 shows the final features of deep websites (*FSS*) after visiting 1966 deep websites for the *auto* domain. Table 6 illustrates that *SmartCrawler* can learn not only the common patterns (e.g. , "auto", "car", "vhicl"), but also auto brand terms (e.g., "ford", "nissan", "honda"). Since no auto brand terms were initialized as features of deep websites, this experiment shows that *SmartCrawler* can effectively learn new patterns by itself.

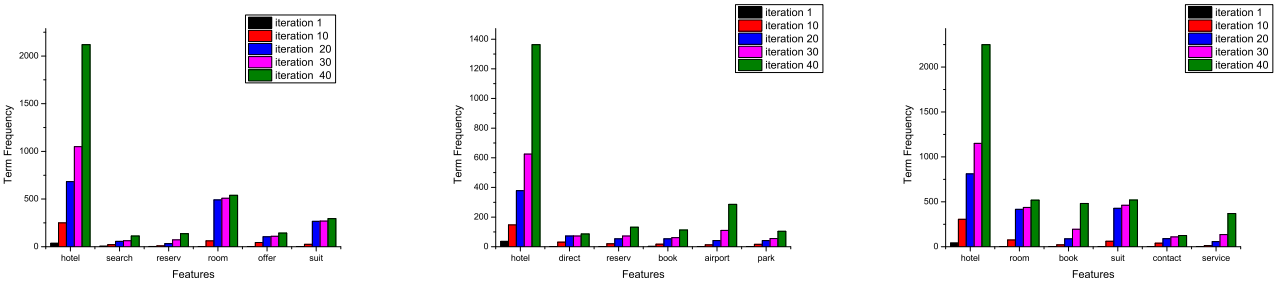
Figure 10 illustrates the features of form links over different learning iterations in the hotel domain. From the figure, we can see that *SmartCrawler* can identify relevant features (e.g., "hotel" and "book" in URLs, anchors, and texts around links) from the initial iteration and the frequency of these terms maintain a stable increment over iterations. On the other hand, new features such as "reserv" in anchors and "destin" in around texts are adaptively learned after a few iterations.

## 6 Conclusion and Future Work

In this paper, we propose an effective harvesting framework for deep-web interfaces, namely *SmartCrawler*. We have shown that our approach achieves both wide coverage for deep web interfaces and maintains highly efficient crawling. *SmartCrawler* is a focused crawler consisting of two stages: efficient site locating and balanced in-site exploring. *SmartCrawler* performs site-based locating by reversely searching the known deep web sites for center pages, which can effectively find many data sources for sparse

TABLE 6: The top features of deep websites in *auto* domain after visiting 1966 deep websites

| Attribute | Deep Website Features |
|---|---|
| URL | (**auto,358**) (**car,196**) (**ford,83**) (**nissan,73**) (acura,67) (**honda,51**) (**toyota,49**) (**motor,47**) (warranti,38) (kopen,35) (forum,23) (**benz,16**) (onlin,16) (van,15) (vw,15) (**mitsubishi,14**) (**kia,12**) (truck,11) |
| Anchor | (**warranti,263**) (websit,215) (view,188) (**dealer,184**) (**car,162**) (**auto,126**) (extend,79) (world,77) (camp,75) (part,75) (sale,62) (**ford,56**) (acura,52) (rv,51) (**nissan,50**) (servic,46) (forum,46) (kopen,40) (special,37) |
| Text | (**auto,260**) (**dealer,238**) (**vehicl,231**) (**car,225**) (**warranti,223**) (part,188) (view,174) (sale,149) (servic,108) (acura,104) (special,103) (world,99) (extend,99) (camp,94) (kopen,85) (**toyota,79**) (forum,78) (**honda,74**) (rv,73) |




(a) Path (b) Anchor (c) Text

Fig. 10: Features of links with embedded forms ($FSL$) extracted in different iterations of adaptive learning in the hotel domain.

domains. By ranking collected sites and by focusing the crawling on a topic, *SmartCrawler* achieves more accurate results. The in-site exploring stage uses adaptive link-ranking to search within a site; and we design a link tree for eliminating bias toward certain directories of a website for wider coverage of web directories. Our experimental results on a representative set of domains show the effectiveness of the proposed two-stage crawler, which achieves higher harvest rates than other crawlers. In future work, we plan to combine pre-query and post-query approaches for classifying deep-web forms to further improve the accuracy of the form classifier.

## REFERENCES


[1] Peter Lyman and Hal R. Varian. How much information? 2003. Technical report, UC Berkeley, 2003.

[2] Roger E. Bohn and James E. Short. How much information? 2009 report on american consumers. Technical report, University of California, San Diego, 2009.

[3] Martin Hilbert. How much information is there in the "information society"? *Significance*, 9(4):8–12, 2012.

[4] Idc worldwide predictions 2014: Battles for dominance – and survival – on the 3rd platform. http://www.idc.com/research/Predictions14/index.jsp, 2014.

[5] Michael K. Bergman. White paper: The deep web: Surfacing hidden value. *Journal of electronic publishing*, 7(1), 2001.

[6] Yeye He, Dong Xin, Venkatesh Ganti, Sriram Rajaraman, and Nirav Shah. Crawling deep web entity pages. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 355–364. ACM, 2013.

[7] Infomine. UC Riverside library. http://lib-www.ucr.edu/, 2014.

[8] Clusty's searchable database dirctory. http://www.clusty.com/, 2009.

[9] Booksinprint. Books in print and global books in print access. http://booksinprint.com/, 2015.

[10] Kevin Chen-Chuan Chang, Bin He, and Zhen Zhang. Toward large scale integration: Building a metaquerier over databases on the web. In *CIDR*, pages 44–55, 2005.

[11] Denis Shestakov. Databases on the web: national web domain survey. In *Proceedings of the 15th Symposium on International Database Engineering & Applications*, pages 179–184. ACM, 2011.

[12] Denis Shestakov and Tapio Salakoski. Host-ip clustering technique for deep web characterization. In *Proceedings of the 12th International Asia-Pacific Web Conference (APWEB)*, pages 378–380. IEEE, 2010.

[13] Denis Shestakov and Tapio Salakoski. On estimating the scale of national deep web. In *Database and Expert Systems Applications*, pages 780–789. Springer, 2007.

[14] Shestakov Denis. On building a search interface discovery system. In *Proceedings of the 2nd international conference on Resource discovery*, pages 81–93, Lyon France, 2010. Springer.

[15] Luciano Barbosa and Juliana Freire. Searching for hidden-web databases. In *WebDB*, pages 1–6, 2005.

[16] Luciano Barbosa and Juliana Freire. An adaptive crawler for locating hidden-web entry points. In *Proceedings of the 16th international conference on World Wide Web*, pages 441–450. ACM, 2007.

[17] Soumen Chakrabarti, Martin Van den Berg, and Byron Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[18] Jayant Madhavan, David Ko, Łucja Kot, Vignesh Ganapathy, Alex Rasmussen, and Alon Halevy. Google's deep web crawl. *Proceedings of the VLDB Endowment*, 1(2):1241–1252, 2008.

[19] Olston Christopher and Najork Marc. Web crawling. *Foundations and Trends in Information Retrieval*, 4(3):175–246, 2010.

[20] Balakrishnan Raju and Kambhampati Subbarao. Sourcerank: Relevance and trust assessment for deep web sources based on inter-source agreement. In *Proceedings of the 20th international conference on World Wide Web*, pages 227–236, 2011.

[21] Balakrishnan Raju, Kambhampati Subbarao, and Jha Manishkumar. Assessing relevance and trust of the deep web sources and results based on inter-source agreement. *ACM Transactions on the Web*, 7(2):Article 11, 1–32, 2013.

[22] Mustafa Emmre Dincturk, Guy vincent Jourdan, Gregor V.

Bochmann, and Iosif Viorel Onut. A model-based approach for crawling rich internet applications. *ACM Transactions on the Web*, 8(3):Article 19, 1–39, 2014.

[23] Kevin Chen-Chuan Chang, Bin He, Chengkai Li, Mitesh Patel, and Zhen Zhang. Structured databases on the web: Observations and implications. *ACM SIGMOD Record*, 33(3):61–70, 2004.

[24] Wensheng Wu, Clement Yu, AnHai Doan, and Weiyi Meng. An interactive clustering-based approach to integrating source query interfaces on the deep web. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 95–106. ACM, 2004.

[25] Eduard C. Dragut, Thomas Kabisch, Clement Yu, and Ulf Leser. A hierarchical approach to model web query interfaces for web source integration. *Proc. VLDB Endow.*, 2(1):325–336, August 2009.

[26] Thomas Kabisch, Eduard C. Dragut, Clement Yu, and Ulf Leser. Deep web integration with visqi. *Proceedings of the VLDB Endowment*, 3(1-2):1613–1616, 2010.

[27] Eduard C. Dragut, Weiyi Meng, and Clement Yu. *Deep Web Query Interface Understanding and Integration*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2012.

[28] André Bergholz and Boris Childlovskii. Crawling for domain-specific hidden web resources. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*, pages 125–133. IEEE, 2003.

[29] Sriram Raghavan and Hector Garcia-Molina. Crawling the hidden web. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 129–138, 2000.

[30] Cheng Sheng, Nan Zhang, Yufei Tao, and Xin Jin. Optimal algorithms for crawling a hidden database in the web. *Proceedings of the VLDB Endowment*, 5(11):1112–1123, 2012.

[31] Panagiotis G Ipeirotis and Luis Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *Proceedings of the 28th international conference on Very Large Data Bases*, pages 394–405. VLDB Endowment, 2002.

[32] Nilesh Dalvi, Ravi Kumar, Ashwin Machanavajjhala, and Vibhor Rastogi. Sampling hidden objects using nearest-neighbor oracles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 1325–1333. ACM, 2011.

[33] Jayant Madhavan, Shawn R. Jeffery, Shirley Cohen, Xin Dong, David Ko, Cong Yu, and Alon Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proceedings of CIDR*, pages 342–350, 2007.

[34] Brightplanet's searchable database dirctory. http://www.completeplanet.com/, 2001.

[35] Mohamamdreza Khelghati, Djoerd Hiemstra, and Maurice Van Keulen. Deep web entity monitoring. In *Proceedings of the 22nd international conference on World Wide Web companion*, pages 377–382. International World Wide Web Conferences Steering Committee, 2013.

[36] Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th international conference on World Wide Web*, pages 148–159, 2002.

[37] Luciano Barbosa and Juliana Freire. Combining classifiers to identify online databases. In *Proceedings of the 16th international conference on World Wide Web*, pages 431–440. ACM, 2007.

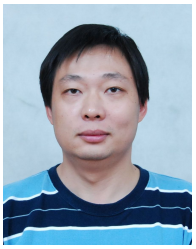
[38] Jared Cope, Nick Craswell, and David Hawking. Automated discovery of search interfaces on the web. In *Proceedings of the 14th Australasian database conference-Volume 17*, pages 181–189. Australian Computer Society, Inc., 2003.

[39] Dumais Susan and Chen Hao. Hierarchical classification of Web content. In *Proceedings of the 23rd Annual International ACM SIGIR conference on Research and Development in Information Retrieval*, pages 256–263, Athens Greece, 2000.

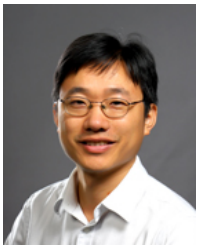
[40] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The weka data mining software: an update. *SIGKDD Explorations Newsletter*, 11(1):10–18, November 2009.

[41] Open directory project. http://www.dmoz.org/, 2013.

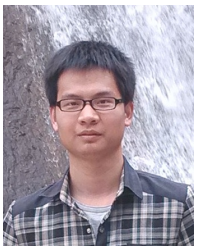
[42] The UIUC web integration repository. http://metaquerier.cs.uiuc.edu/repository/, 2003.



**Feng Zhao** received his M.S. and Ph.D degrees in Computer Science from Huazhong University of Science and Technology in 2003 and 2006. He is currently an associate professor of College of Computer Science and Technology, Huazhong University of Science and Technology, China. He is the vice dean of innovation institute of the same university. His research interests include information retrieval, data mining, security and distributed computing.



**Jingyu Zhou** received the B.S. degree in Computer Science from Zhejiang University, China, in 1999. He received the M.S. and Ph.D. degrees in Computer Science from University of California at Santa Barbara in 2003 and 2006. He is currently an associate professor at Shanghai Jiao Tong University. He is generally interested in distributed systems, information retrieval, and security. He has published more than 30 papers at various conferences and journals, including WWW, INFOCOM, ICS, TPDS, DSN, FSE, CIKM, and IPDPS. He has served as PC members for more than 20 international conferences.



**Chang Nie** received the B.S. degree in Software Engineer from Northeastern University, China, in 2011. He received his M.S. degree in computer science from Huazhong University of Science and Technology in 2014. His research interests include information retrieval and web crawling.

**Heqing Huang** received the B.S. degree in Computer Science from Huazhong University of Science and Technology, China, in 2010. He is currently a Ph.D candiate of department of Computer Science and Engineering at the Pennsylvania State University, University Park. His research interests include information retrieval, data mining, software security, mobile system security and risk analysis.

**Hai Jin** received his M.S. and Ph.D degrees in computer science from Huazhong University of Science and Technology in 1991 and 1994. He is currently a professor of College of Computer Science and Technology, Huazhong University of Science and Technology, China. He is the Dean of College of Computer Science and Technology of the same university. His current research interests include cloud computing, virtual computing and computer architecture. He has published more than 200 papers at various conferences and journals, including VLDB, WWW, ICDCS, SC, TC, TKDE, TPDS and TSC.