Web Intelligence and Agent Systems: An International Journal 10 (2012) 75–88
DOI 10.3233/WIA-2012-0232
IOS Press

Selecting queries from sample to crawl deep web data sources

Yan Wang [a,*], Jianguo Lu [a,c], Jie Liang [a], Jessica Chen [a] and Jiming Liu [b]
[a] *School of Computer Science, University of Windsor, Windsor, Ontario, Canada*
E-mail: {jlu,wang16c,liangr,xjchen}@uwindsor.ca
[b] *Department of Computer Science, Hong Kong Baptist University, Hong Kong, China*
E-mail: jiming@comp.hkbu.edu.hk
[c] *State Key Lab for Novel Software Technology, Nanjing University, Nanjing, China*

Abstract. This paper studies the problem of selecting queries to efficiently crawl a deep web data source using a set of sample documents. Crawling deep web is the process of collecting data from search interfaces by issuing queries. One of the major challenges in crawling deep web is the selection of the queries so that most of the data can be retrieved at a low cost. We propose to learn a set of queries from a sample of the data source. To verify that the queries selected from a sample also produce a good result for the entire data source, we carried out a set of experiments on large corpora including Gov2, newsgroups, wikipedia and Reuters. We show that our sampling-based method is effective by empirically proving that 1) The queries selected from samples can harvest most of the data in the original database; 2) The queries with low overlapping rate in samples will also result in a low overlapping rate in the original database; and 3) The size of the sample and the size of the terms from where to select the queries do not need to be very large. Compared with other query selection methods, our method obtains the queries by analyzing a small set of sample documents, instead of learning the next best query incrementally from all the documents matched with previous queries.

Keywords: Deep web, hidden web, invisible web, crawling, query selection, sampling, set covering, web service

1. Introduction

The deep web [7] is the content that is dynamically generated from data sources such as databases or file systems. Unlike surface web where web pages are collected by following the hyperlinks embedded inside collected pages, data from a deep web are guarded by a search interface such as HTML form, web services, or programmable web API [36], and can be retrieved by queries. The amount of data in deep web exceeds by far that of the surface web. This calls for deep web crawlers to excavate the data so that they can be used, indexed, and searched upon in an integrated environment. With the proliferation of publicly available web services that provide programmable interfaces, where input and output data formats are explicitly specified, automated extraction of deep web data becomes more practical.

The deep web crawling has been studied in two perspectives. One is the study of the macroscopic views of the deep web, such as the number of the deep web data sources [10,15,29], the shape of such data sources (e.g., the attributes in the html form) [29], and the total number of pages in the deep web [7].

When surfacing or crawling the deep web that consists of tens of millions of HTML forms, usually the focus is on the coverage of those data sources rather than the exhaustive crawling of the content inside one specific data source [29]. That is, the breadth, rather than the depth, of the deep web is preferred when the computing resource of a crawler is limited. In this kind of breadth oriented crawling, the challenges are locating the data sources [15], learning and understanding the interface and the returning result so that

*Corresponding author.

query submission and data extraction can be automated [1,6,19,37].

Another category of crawling is depth oriented. It focuses on one designated deep web data source, with the goal to garner most of the documents from the given data source with minimal cost [5,9,22,33,40]. In this realm, the crucial problem is the selection of queries to cover most of the documents in a data source. Let the set of documents in a data source be the universe. Each query represents the documents it matches, i.e., a subset of the universe. The query selection problem is thus cast as a set covering problem. Unlike the traditional set covering problem where the universe and all the subsets are known, the biggest challenge in query selection is that before the queries are selected and documents are downloaded, there are no subsets to select from.

One approach taken by Ntoulas et al. to solving this problem is to learn the global picture by starting with a random query, downloading the matched documents, and learning the next query from the current documents [33]. This process is repeated until all the documents are downloaded. A shortcoming of this method is the requirement of downloading and analyzing all the documents covered by current queries in order to select the next query to be issued, which is highly inefficient. In addition, in applications where only the links are the target of the crawling, downloading the entire documents is unnecessary. Even when our final goal is to download the documents instead of the URLs, it would be more efficient to separate the URLs collection from the document downloading itself. Usually, the implementation of a downloader should consider factors such as multi-threading and network exceptions, and should not be coupled with link collection.

Because of those practical considerations, we propose an efficient sampling-based method for collecting the URLs of a deep web data source. We first collect from the data source a set of documents as a sample that represents the original data source. From the sample data, we select a set of queries that cover most of the sample documents with a low cost. Then we use this set of queries to extract data from the original data source.

The main contribution of this paper is the hypothesis that the queries working well on the sample will also induce satisfactory results on the total data base (i.e., the original data source). More precisely, this paper conjectures that:

1. The vocabulary learnt from the sample can cover most of the total data base;
2. The overlapping rate in the sample can be projected to the total data base;
3. The sizes of the sample and the query pool do not need to be very large.

While the first result can be derived from [8], the last two are not reported in the literature as far as we are aware of. As our method is dependent on the sample size and query pool size, we have empirically determined the appropriate sizes for the sample and the query pool for effective crawling of a deep web.

In this paper, we focus on querying textual data sources, i.e., the data sources that contain plain text documents only. This kind of data sources usually provides a simple keywords-based query interface instead of multiple attributes as studied by Wu et al. [40]. Madhavan et al.'s study [29] shows that the vast majority of the html forms found by Google deep web crawler contain only one search attribute. Hence we focus on such search interfaces.

2. Related work

There has been a flurry of research of data extraction from web [19], and more recently on deep web [9,29,33]. The former focuses on extracting information from HTML web pages, especially on the problem of turning un-structured data into structured data. The latter concentrates on locating deep web entries [6,15,29], automated form filling [9,37], and query selection [5,22,33,40]. Olston and Najork provided a good summary on deep web crawling [34] in general. Khare, An, and Song surveyed the work on automated query interface understanding and form filling [18].

A naive approach to selecting queries to cover a textual data source is to choose words randomly from a dictionary. In order to reduce the network traffic, queries to be sent need to be selected carefully. Various approaches [5,23,33,40] have been proposed to solve the query selection problem, with the goal of maximizing the coverage of the data source while minimizing the communication costs. Their strategy is to minimize the number of queries issued, by maximizing the unique returns of each query.

One of the most elaborate query selection methods along this line was proposed by Ntoulos et al. [33]. The authors used an adaptive method to select the next query to issue based on the documents downloaded from previous queries. The query selection problem is modeled as a set covering problem [39]. A greedy algorithm for set-covering problem is used to select

an optimal query based on the documents downloaded so far and the prediction of document frequencies of the queries on the entire corpus. The focus is to minimize the number of queries sent out, which is important when data sources impose the limit for the number of queries that can be accepted for each user account. Our focus is minimizing the network traffic, which is the overlapping rate.

Wu et al. [40] propose an iterative method to crawl structured hidden web. Unlike our simple keyword-based search interface, it considers interfaces with multiple attributes. Also, the data sources are considered structured (such as relational database), instead of text documents as we discussed.

Barbosa and Freire [5] pointed out the high overlapping problem in data extraction, and proposed a method trying to minimize the number of queries. Liddle et al. [22] gave several evaluation criteria for the cost of data extraction algorithms, and presented a data extraction method for web forms with multiple attributes.

We reported our preliminary result in a rather short paper [25]. This paper extends the previous work by adding more experiment results and analysis.

3. Problem formalization

3.1. Hit rate

The goal of data extraction is to harvest most of the data items within a data source. This is formalized as the Hit Rate that is defined below.

Let q be a query and DB a database. We use $S(q, DB)$ to denote the set of data items in response to query q on database DB.

Definition 1 (Hit Rate, HR)**.** Given a set of queries $Q = \{q_1, q_2, \ldots, q_i\}$ and a database DB. The hit rate of Q in DB, denoted by $HR(Q, DB)$, is defined as the ratio between the number of unique data items collected by sending the queries in Q to DB and the size of the data base DB, i.e.:

$$u = \left| \bigcup_{j=1}^{i} S(q_j, DB) \right|,$$

$$HR(Q, DB) = \frac{u}{|DB|}.$$

3.2. Overlapping rate

The cost of deep web crawling in our work refers to the redundant links that are retrieved, which can be defined by the overlapping rate.

Definition 2 (Overlapping Rate, OR)**.** Given a set of queries $Q = \{q_1, q_2, \ldots, q_i\}$, the overlapping rate of Q in DB, denoted by $OR(Q, DB)$, is defined as the ratio between the total number of collected links (n) and the number of unique links retrieved by sending queries (u) in Q to DB, i.e.,

$$n = \sum_{j=0}^{i} |S(q_j, DB)|,$$

$$OR(Q, DB) = n/u.$$

Intuitively, the cost can be measured in several aspects, such as the number of queries sent, the number of document links retrieved, and the number of documents downloaded. Ntoulas et al. [33] assigned weights to each factor and use the weighted sum as the total cost. While it is a straightforward modeling of the real world, this cost model is rather complicated to track. In particular, the weights are difficult to verify in different deep web data sources, and the crawling method is not easy to be evaluated against such cost model.

We observe that almost all the deep web data sources return results in pages, instead of a single long list of documents. For example, if there are 1,000 matches, a deep web data source such as a web service or an html form may return one page that consists of only 10 documents. If you want the next 10 documents, a second query needs to be sent. Hence in order to retrieve all the 1000 matches, altogether 100 queries with the same query terms are required.

With this scenario the number of queries is proportional to the total number of documents retrieved. Hence there is no need to separate those two factors when measuring the cost. That is why we simply use n, the total number of retrieved documents, as the indicator of the cost. Since data sources vary in their sizes, a large data source with larger n does not necessarily mean that the cost is higher than a smaller n in a small data source. Therefore we normalize the cost by dividing the total number of documents by the unique ones. When all the documents are retrieved, u is equal to the data source size.

Example 1. Suppose that our data source DB has three documents d_1, d_2, and d_3. d_1 contains two terms t_1 and t_2, d_2 contains t_1 and t_3, and d_3 contains t_2 only. The matrix representation of the data source is shown in Table 1. OR and HR for queries $\{t_1, t_2\}$ and

Table 1
HR and OR example

	d_1	d_2	d_3
t_1	1	1	0
t_2	1	0	1
t_3	0	1	0

$\{t_2, t_3\}$ are calculated as below:

$$OR(\{t_1, t_2\}, DB) = \frac{2+2}{3} = \frac{4}{3},$$
$$HR(\{t_1, t_2\}, DB) = \frac{3}{3} = 1,$$
$$OR(\{t_2, t_3\}, DB) = \frac{2+1}{3} = 1,$$
$$HR(\{t_2, t_3\}, DB) = \frac{3}{3} = 1.$$

Since $\{t_2, t_3\}$ has a lower OR than $\{t_1, t_2\}$ and they produce the same HR, we should use $\{t_2, t_3\}$ instead of $\{t_1, t_2\}$ to retrieve the documents.

3.3. Relationship between HR and OR

Another reason to use HR and OR to evaluate the crawling method is that there is a fixed relationship between HR and OR when documents can be obtained randomly. I.e., if documents can be retrieved randomly with equal capture probability, we have shown in [24,27] that

$$HR = 1 - OR^{-2.1}. \quad (1)$$

When documents are retrieved by random queries, the relationship between HR and OR are roughly

$$HR = 1 - OR^{-1}. \quad (2)$$

As a rule a thumb, when $OR = 2$, most probably we have retrieved 50% of the documents in the deep web with random queries. This provides a convenient way to evaluate the crawling methods.

3.4. Our method

The challenge in selecting appropriate queries is that the actual corpus is unknown to the crawler from the outside, hence the crawler cannot select the most suitable queries without the global knowledge of the underlying documents inside the database.

With our deep web crawling method, we first download a sample set of documents from the total database.

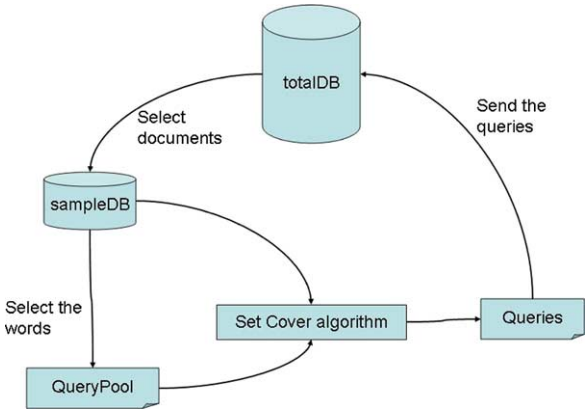

Fig. 1. Crawling method based on sampling.

Algorithm 1 Outline of Deep Web Crawling algorithm $DWC(TotalDB, s, p)$

Input: the original data source $TotalDB$; sample size s, query pool size p.
Output: A set of terms in Queries

1: Create a sample data base $SampleDB$ by randomly selecting s number of documents from the corpus $TotalDB$;
2: Create a query pool $QueryPool$ of size p from the terms that occur in $SampleDB$;
3: Select a set of queries $Queries$ from $QueryPool$ that can cover at least 99% of the $SampleDB$ by running a set covering algorithm;

From this sample, we select a set of queries that can cover most of the documents in the sample set with low cost. This paper shows that the same set of queries can be also used to cover most of the documents in the *original* data source with a low cost. This method is illustrated in Fig. 1 and explained in Algorithm 1.

In the following, we will use $DWC(db, s, p)$ to denote the output obtained by running the algorithms with input the data source db, the sample size s, and the query pool size p.

In our algorithm and experiments random samples are obtained by generating uniformly distributed random numbers within the range of the document IDs in the corpus. However, in practical applications we do not have the direct access to the whole corpus. Instead, only queries can be sent and the matched documents are accessed. In this scenario the random sampling of the documents in a corpus is a challenging task, and has attracted substantial studies (for example in [4]). Since the cost of obtaining such random samples are rather high, our experiments skip the random sampling

process and take the random samples directly from the corpus.

To refine this algorithm, there are several parameters that need to be decided.

One is the number of documents that should be selected into the sample, i.e., the size of $SampleDB$. Although in general the larger sample will always produce a better result, we need to find an appropriate size for the sample so that it is amenable to efficient processing while still large enough to produce a satisfactory query list in $QueryPool$.

The second uncertainty is how to select the terms from the $SampleDB$ in order to form the $QueryPool$. There are several parameters that can influence the selection of terms, typically, the size of the pool and the document frequencies of the selected terms.

Thus the $Queries$ finally selected from the query pool depends on various criteria, such as the size of $Queries$, the algorithm chosen to obtain the terms in $SampleDB$, and the document frequencies of those terms selected.

The soundness of Algorithm 1 relies on the hypothesis that the vocabulary that works well for the sample will also be able to extract the data from the actual database effectively. More precisely, this hypothesis says that

1. the terms selected from $SampleDB$ will cover most documents in the $TotalDB$, and
2. the overlapping rate in $TotalDB$ will be close to the overlapping rate in the $SampleDB$.

Before analyzing the correspondence between the sample and total databases in detail, we first study the problem of query selection from a sample data base.

4. Select queries from SampleDB

4.1. Create the query pool

In order to select the queries to issue, we need to obtain a query pool $QueryPool$ first. $QueryPool$ is built from the terms in a random sample of the corpora. We should be aware that random queries can not produce random documents because large documents have higher probability of being matched. It is a rather challenging task to obtain random documents from a searchable interface [4].

Not every word in the first batch of the search results should be taken into our consideration, due to the time constraint we suffer in order to calculate an effective query set $Queries$ from the $SampleDB$ with high hit rate and low overlapping rate. As we mentioned in the Introduction, searching for an optimal query set can be viewed as a set-covering problem. Set-covering problem is NP-hard, and satisfactory heuristic algorithms in the literature have a time complexity that are at least quadratic to the number of input words. This determines that we are able to calculate $Queries$ only with a query pool of limited size. The first batch of search result, on the contrary, may well-exceed such a limit. For instance, a first-batch of result randomly selected from a newsgroups data contains more than 26,000 unique words. Therefore, we only consider a subset of words from the sample documents as a query pool.

Apparently, the size of this subset will affect the quality of the $Queries$ we generate. Moreover, it should be measured relative to the sample size and the document frequencies of the terms.

Intuitively, when a sample contains only a few documents, very few terms would be enough to jointly cover all of those documents. When the sample size increases, very often we need to add more terms into the $QueryPool$ in order to capture all the new documents.

There is another factor to consider when selecting queries in the query pool, i.e., the document frequency (DF) of the terms in the sample size. There are a few options:

Random terms Randomly selecting the terms in the sample database may be an obvious choice. However, it suffers from low hit rate because most of the randomly selected queries are of low document frequencies. According to Zipf's Law [42], the distribution of words sorted by their frequency (i.e., number of occurrences) is very skewed [42]. In one of our $SampleDB$ there are about 75% of the words that have very low frequencies. Therefore, by randomly polling the words from the vocabulary, we will get many queries with small number of returns.

Popular terms Another possible choice is the popular words in the sample, which are terms with high document frequencies. Popular words such as stop words are not selected for several reasons. One is that many data sources do not index stop words. Hence using stop words to extract documents may not return many results. The second reason is that data sources usually have a restriction on the maximal number of results that can be returned to users. Even if a query can match many documents, data sources will return only part of it. In addition, the words that return more documents also bring in more duplicates.

Terms within certain range of document frequencies Since neither random words nor popular words are good choices, in the experiment described in this section, we will select terms that have document frequency ranging between 2 and 20% of the sample size. For example, if the sample size is 2,000, we use the words with DF values between 2 and 400. Words with DF $= 1$ are most probably rare words. Words that appear in more than 400 documents are too popular to consider.

The size of the query pool should also be measured relative to the document frequencies of its terms: terms with low frequency contribute less to the coverage of the document set. Taking into account the sample size and the document frequencies, we define the following relative query pool size as a factor that influences the quality of the generated queries.

Definition 3 (Relative query pool size)**.** Let Q be a set of queries and DB a database. The relative query pool size of Q on DB, denoted by $poolSize(Q, DB)$, is defined as follows

$$poolSize(Q, DB) = \frac{\sum_{q \in Q} df(q, DB)}{|DB|},$$

where $df(q, DB)$ denotes the document frequency of q in DB, i.e., the number of documents in DB that contain the query q.

$poolSize$ indicates the total number of documents that can be retrieved by the queries in the query pool, normalized by the data collection size. For example, if $poolSize = 20$, on average each document is captured 20 times if all the queries are used. As for exactly what is the best relative size of the query pool, we will analyze that in Section 5.4.2.

4.2. Select queries from the query pool

Once the query pool is established, we need to select from this pool some queries that will be sent to the $TotalDB$.

Let $QueryPool = \{q_1, q_2, \ldots, q_n\}$ be a query pool. We need to find a subset $Queries = \{q^1, \ldots, q^m\}$, where $m < n$, so that

$$HR(Queries, SampleDB) = 1,$$

and

$$OR(Queries, SampleDB)$$

is minimal.

Let $S = \bigcup_{i=1}^{n} S(q_i, SampleDB)$ and $S_i = S(q_i, SampleDB)$. Apparently, we have $S_i \subseteq S$ for $i = 1, \ldots, n$, and $\bigcup_{i=1}^{n} S_i = S$. Thus, the above problem is equivalent to the set-covering problem, i.e. to find a set $S^1, \ldots, S^m$ such that for any $i = 1, \ldots, m$, $S^i = S_j$ for some j, $\bigcup_{i=1}^{m} S^i = S$ and that the cost of the cover as defined by $\sum_{i=1}^{m} |S^i|$ is minimal.

Selecting a collection of appropriate queries can be modeled as a set covering problem as follows:

Definition 4 (Set Covering problem)**.** Given a data source DB whose set of documents is denoted by S, and finite sets $S_1, \ldots, S_n$, where $S_i \subseteq S$, $i = 1, \ldots, n$, each representing the set of documents returned by a query q_i, i.e., $S_i = S(q_i, DB)$. Let $J = \{1, 2, \ldots, n\}$. A subset J^* of J is a cover if

$$\bigcup_{j \in J^*} S_j = S.$$

The cost of the cover is

$$\sum_{j \in J^*} |S_j|.$$

The set covering problem is to find a cover with minimum cost.

Note that in our formulation, the cost function is defined as the sum of the cardinalities of the subsets, with the aim to minimize the overlapping rate. However, the goal defined in [33] is to select minimal number of queries.

In query selection, it is not easy to find a complete cover, especially when the language model of the data source, such as the distribution of the terms in the corpus, is unknown before hand. Hence, the set-covering problem is generalized to the p-partial covering problem, where p is the proportion of the coverage required.

Set covering is an NP-hard problem and it is not easy to find an optimal solution. Various heuristic algorithms have been proposed in both graph theory and various application domains such as crew scheduling. We have adopted the most straightforward greedy algorithm. According to this algorithm, the set of queries are generated by repeatedly selecting the next query which minimizes the cost and maximizes the new documents returned.

Our greedy algorithm for p-partial covering is shown in Algorithm 2.

Algorithm 2 p-partial set covering algorithm

set $J^* = \{\}$;
while $\left|\bigcup_{j \in J*} S_j\right| < p\,|S|$) **do**
  find a k minimizing $\dfrac{|S_k|}{|S_k - \bigcup_{j \in J*} S_j|}$;
  add k to J^*;
end while
for each $k \in J^*$ **do**
  remove S_k if it is redundant;
end for

Example 2. For the matrix in Example 1, a greedy algorithm will select t_1 first. Next it will select t_3. $OR(t_1, t_2) = 4/3$. In this case $HR = 1$, $OR(\{t_1, t_2\}, DB) = 4/3$.

5. Experiments

The purpose of the experiments is to study how well the queries selected from $SampleDB$ perform in the $TotalDB$. If the approach is effective, we want to identify the appropriate sample size and the relative query pool size.

5.1. Data

We run our experiments on a variety of data collected from various domains. The four corpora are Reuters, Gov, Wikipedia, and Newsgroups. They are of different sizes ranging between 30 thousands to 1.4 millions. Their characteristics are summarized in Table 2. These are standard test data that are used by many researchers in information retrieval.

- *Reuters* is a TREC data set that contains 806,790 news stories in English (http://trec.nist.gov/data/reuters/reuters.html).
- *Gov* is a subset of Gov2 that contains 1 million documents. Gov2 is a TREC test data collected from .gov domain during 2004, which contains 25 million documents. We used only a subset of the data for efficiency consideration.
- *Wikipedia* is the corpus provided by wikipedia.org which contains 1.4 million documents.
- *Newsgroups* includes 30k posts in various newsgroups.

In the experiment we built our own search engine using Lucene [14], in order to have details of a data source such as its size. In real deep web data sources, usually the total number of documents is unknown, hence it is impossible to calculate the HR and evaluate the crawling methods.

Table 2
Summary of test corpora

Name	Number of docs	Size in MB	Avg file size (KB)
Reuters	806,791	666	0.83
Wikipedia	1,369,701	1,950	1.42
Gov	1,000,000	5,420	5.42
Newsgroups	30,000	22	0.73

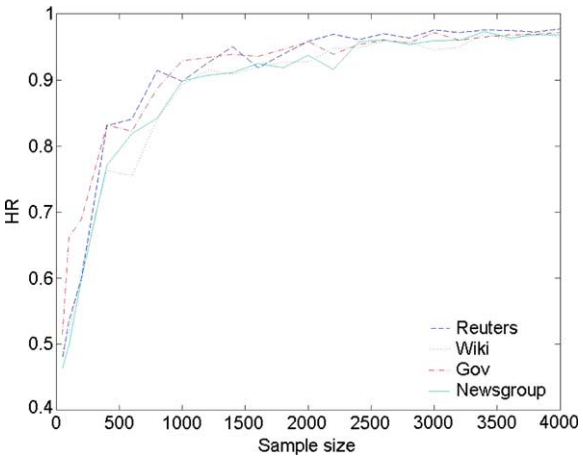

Fig. 2. Impact of sample size on HR. The queries are selected from $SampleDB$ and cover above 99% of the documents in $SampleDB$. The HR in the plot is obtained when those queries are sent to the $TotalDB$. Relative query pool size is 20.

5.2. Hypothesis I

Our first hypothesis is that in general the queries learnt from a small $SampleDB$ can cover most of the data in $TotalDB$, i.e., the queries can be used to retrieve most of the documents in $TotalDB$.

Hypothesis 1. Suppose that $SampleDB$ and the subsequent set of queries Q are created by our algorithm from $TotalDB$. If $|SampleDB| > 1{,}000$ and $poolSize = 20$, then

$$HR(Q, TotalDB) > 0.8.$$

Here we assume that the size of $TotalDB$ is a very large number, i.e., $|TotalDB| \gg 1{,}000$.

We tested the cases where the sample sizes range between 100 to 4,000 documents for the four corpora studied. Figure 2 shows HR in $TotalDB$ as a function of the sample size. It demonstrates that our method quickly finds the queries that can account for 90% of

the documents in $TotalDB$. It is shown that at the beginning when the sample size increases, HR in total database will be higher. After about 1,000 documents, the gain in HR tapers and the increase of sample size has little impact on the HR in $TotalDB$.

This phenomenon can be explained by Heaps' law, which states that when more documents are gathered, there are diminishing returns of new words. When 1,000 documents are checked, most common words are already recovered from the sample. There are very few useful words left outside the 1,000 sample. Hence there is little improvement when sample gets larger.

5.3. Hypothesis II

While the first hypothesis shows that it is easy to select the queries to retrieve most of the documents in $TotalDB$, what we concern more is the cost, i.e., the overlapping rate, to retrieve the data.

Although for $SampleDB$, we make sure that the selected queries have a low cost by applying a set covering algorithm, we are not sure yet whether the cost would be also low for the $TotalDB$. Hence we need to verify our second hypothesis, i.e., the queries selected by Algorithm 1 from $SampleDB$ will also result in low overlapping rate in $TotalDB$. More precisely, it can be described as:

Hypothesis 2. Suppose that queries Q are selected by our method, i.e., $Q = DWC(TotalDB, sampleSize, poolSize)$, where $sampleSize > 1{,}000$, and $poolSize > 10$. Q' is a set of queries selected randomly from the same query pool such that $HR(Q', TotalDB) = HR(Q, TotalDB)$. Then

$$OR(Q, TotalDB) < OR(Q', TotalDB).$$

In order to verify this, we conducted a series of experiments to compare with the cost for random queries. Figure 3 illustrates the effectiveness of our method by comparing it with the performance of random queries. In this experiment our method used greedy algorithm to select the queries from a $QueryPool$ of relative size 20. The query pool is constructed from a sample set of documents of size 3,000. The random method in comparison selects queries randomly from the same query pool. Take the plot for Reuters corpus for example, in order to harvest 90% of the data, random queries will induce approximately 5 overlapping rate, while our method using greedy set-covering algorithm will produce 2.5 overlapping rate. Since Reuters consists of 0.8 million of documents, our method will save $(5 - 2.5) \times 0.8 = 2$ millions of documents compared with the random method.

Although this experiment shows that our method is effective, we need to identify what are the appropriate sizes for $SampleDB$ and $QueryPool$, respectively.

5.4. Hypothesis III

The previous two hypotheses have shown that 1) we can cover most of the documents based on a sample; 2) we can do that with a cost lower than that of a random method. Our next concern is exactly how large the sample and the query pool should be. Our third hypothesis is that in order to download most of the documents with low overlapping, the sample size and the relative query pool size do not need to be very large. This will be elaborated in two aspects, i.e., the sample size and the query pool size.

5.4.1. Sample size

As for the proper sample size, Fig. 2 shows that a few thousands of sample documents are good enough to harvest most of the documents in $TotalDB$. However, we have not shown how OR changes as sample size increases.

The impact of the sample size on OR can be summarized as below:

Hypothesis 3. 1. Suppose that

$$Q_1 = DWC(TotalDB, sampleSize_1, poolSize),$$
$$Q_2 = DWC(TotalDB, sampleSize_2, poolSize).$$

Let $Q'_i \subseteq Q_i$, $i \in \{1, 2\}$, such that

$$HR(Q'_1, TotalDB) = HR(Q'_2, TotalDB).$$

$sampleSize_1 > sampleSize_2$ does not imply that

$$OR(Q'_1, TotalDB) < OR(Q'_2, TotalDB).$$

Intuitively, this hypothesis says that a larger sample size does not guarantee smaller OR.

We conducted a series of experiments to investigate the impact of sample size on the overlapping rate. Figure 4 shows OR in $TotalDB$ as a function of sample size for the four corpora, with hit rate fixed at 89%, and relative query pool size fixed at 20.

It shows that sample size has little effect on OR: sometimes larger sample size may induce higher overlap in $TotalDB$. Although it is counter-intuitive, the reason is that with more documents in the sample,

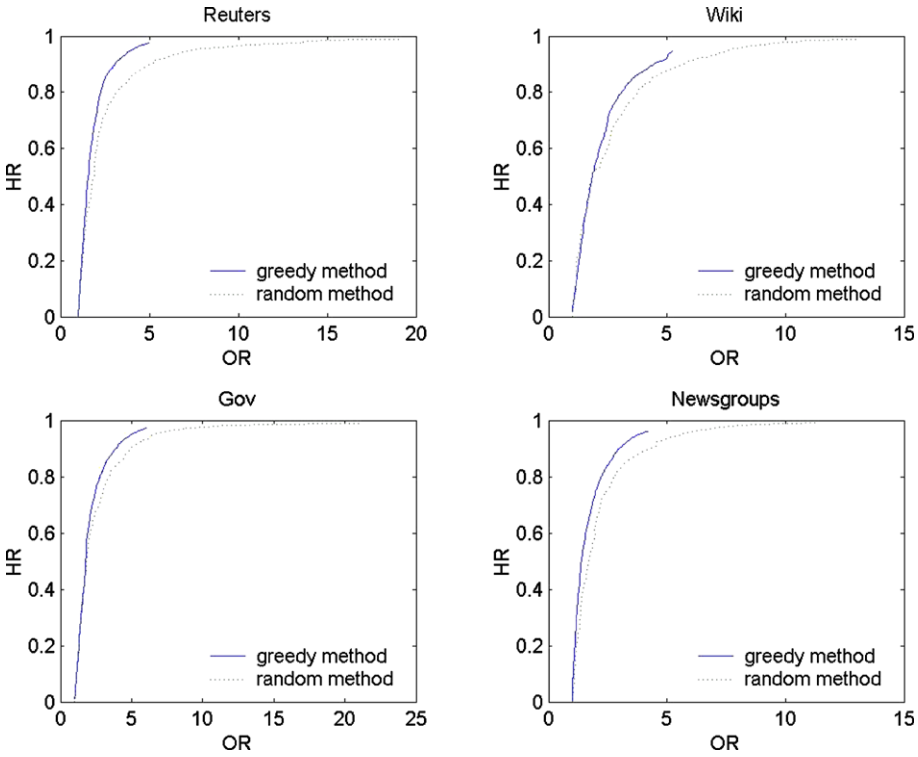

Fig. 3. Comparison of our method on the four corpora with queries selected randomly from sample. X axis is the Overlapping Rate, Y axis is the Hit Rate. Sample size is 3,000, relative query pool size is 20. Our method achieves a much smaller OR when HR is high.

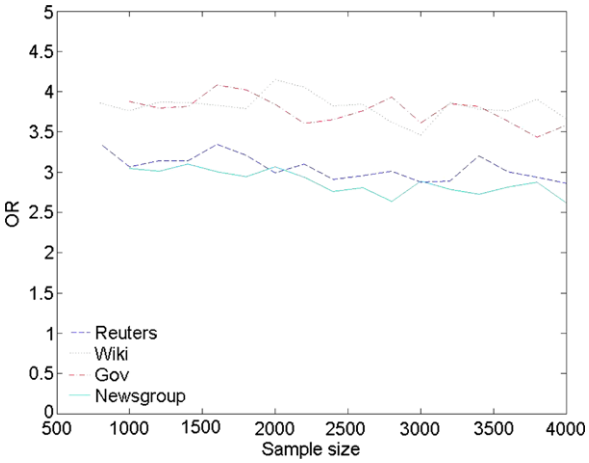

Fig. 4. Impact of sample size on OR. HR is 89%, relative query pool size is 20.

there are more words to choose from, most of them having low frequency according to Zipf's law. When those low frequency words are selected by the set covering algorithm, they result in low OR in $SampleDB$. However, when they are mapped to $TotalDB$, they are most probably still of low frequency, hence do not have much effect on the overall performance.

5.4.2. *Query pool size*

This experiment investigate the impact of query pool size on HR and OR. Since the HR and OR will also vary with different sample size, we include sample size as another dimension of input in our experiments.

First we need to investigate the impact of query pool size on HR. Our hypothesis can be formulated as follows:

Hypothesis 3. 2. Let

$$Q_1 = DWC(TotalDB, sampleSize, poolSize_1),$$
$$Q_2 = DWC(TotalDB, sampleSize, poolSize_2).$$

$poolSize_1 > poolSize_2$ does not imply that

$$HR(Q_1, TotalDB) > HR(Q_2, TotalDB).$$

In particular, when $sampleSize > 1{,}000$, $HR(Q_1, TotalDB)$ achieves the highest value if $poolSize_1$ is between 10 and 20.

Figure 5 shows the HR in $TotalDB$ as a function of relative query pool size and sample size. The pool size is in the range of 5 and 40. We ignored the pool size

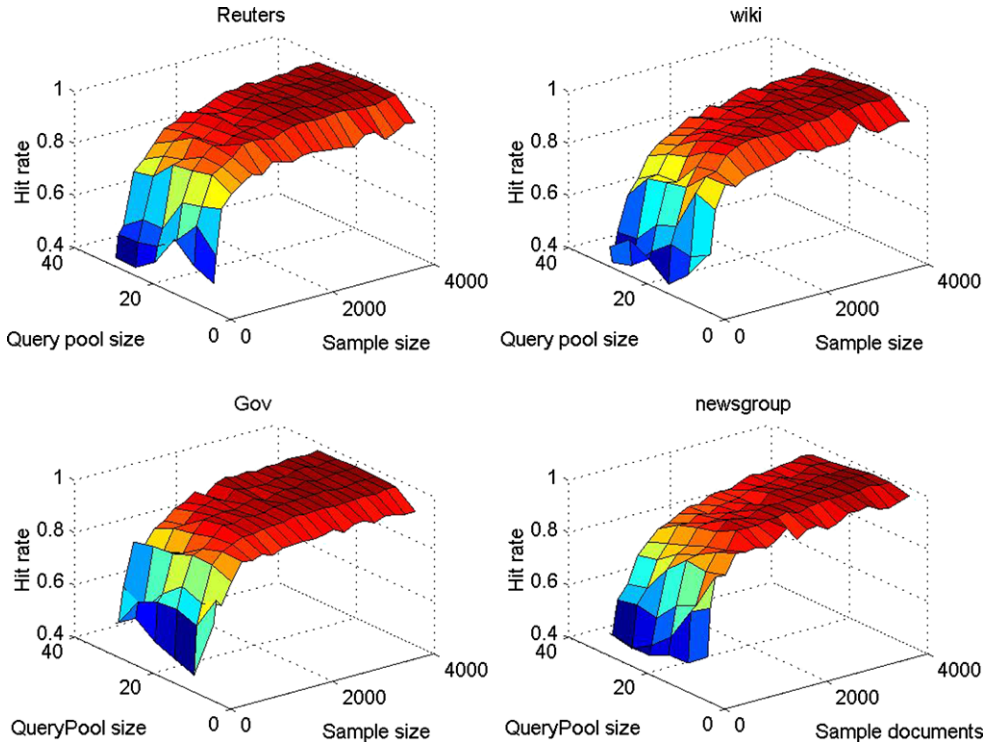

Fig. 5. The impact of QueryPool size on hit rate.

smaller than five because it will not produce enough queries to cover all the documents, let alone select a better cover.

It can be seen that HR is low only when query pool size is below 10. When query pool becomes larger, HR may decrease because of the same reason we explained in the last sub section, i.e., the inclusion of more low frequency words. The conclusion of this experiment is that best performance is achieved when pool size is set between 10 to 20.

Another investigation is the impact of query pool size on OR. OR is dependent of HR, hence it is meaningless to list the OR without the reference to HR. In order to have an objective comparison, we measure the improvement of OR over random method which obtains the same hit rate. Our empirical study shows the following result:

Hypothesis 3. 3. Suppose that Q and Q' have the same hit rate. Q is selected by our algorithm, while Q' is a set of queries randomly selected from the query pool, i.e.,

$$Q = DWC(TotalDB, sampleSize, poolSize),$$
$$HR(Q, TotalDB) = HR(Q', TotalDB).$$

Let OR improvement be

$$OR(Q', TotalDB) - OR(Q, TotalDB).$$

Then OR improvement increases as sample size grows.

The experiment is carried out as follows: we first fire all the selected queries Q to the $TotalDB$ and record the overlapping rate $OR(Q, TotalDB)$ and $HR(Q, TotalDB)$ at the end of querying process. Then we use random queries Q' to reach the same HR, and record the overlapping rate $OR(Q', TotalDB)$ at this point. The improvement of OR is

$$OR(Q', TotalDB) - OR(Q, TotalDB).$$

Figure 6 depicts the improvement of OR while sample size and query pool size vary. It shows that the relative query pool size does not need to be very large– the best result is obtained while relative query pool size is around 20, which can be explained again by the inclusion of rare words.

Figures 6 and 4 seem contradicting with each other– with the increase of sample size, one says that OR improvement increases monotonically while the other says that the OR does not change with a pattern. Ac-

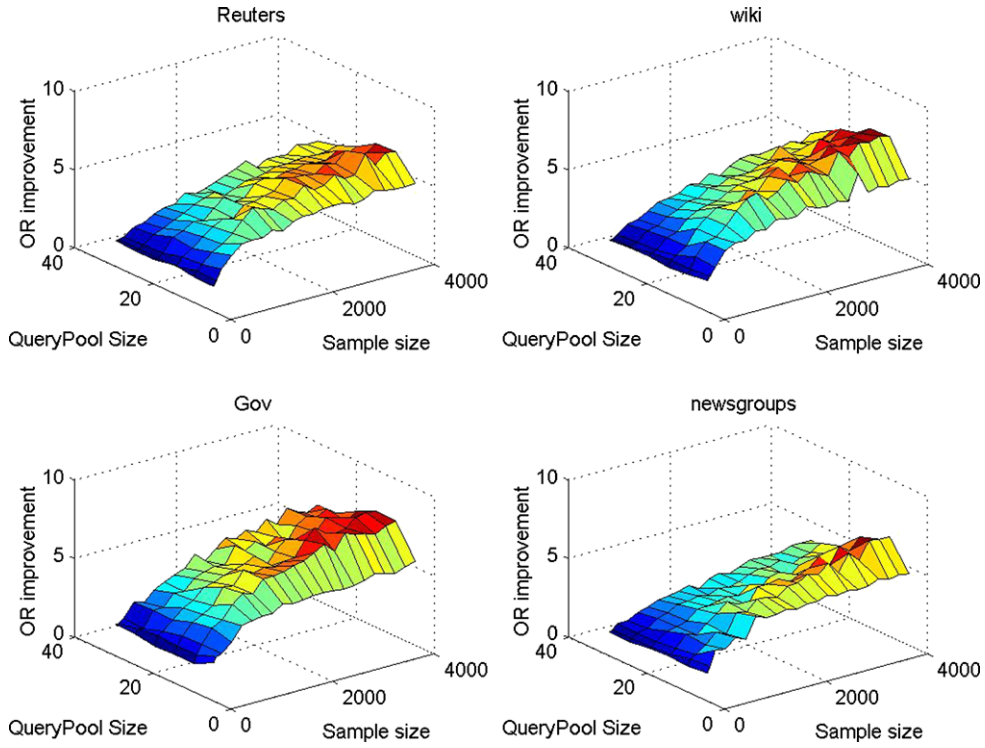

Fig. 6. The impact of QueryPool size on OR.

tually in Fig. 6 the hit rate increases along the growing sample size, resulting in growing OR improvement. On the other hand, experiments described in Fig. 4 have a fixed HR.

Figure 6 also gives us an overall comparison between our method and the random method. First of all, no matter what sizes of the sample and query pool have, the result of our method is always better than that of random method. We can also see that larger samples will introduce more improvements in overlapping, while query pool size does not matter very much.

5.5. Compare queries on other data sources

The results in the preceding experiments showed that the queries selected from the samples can cover most of the total database effectively. However, these experiments do not rule out another possibility, i.e., whether queries selected from any English corpus may work equally well for all the data sources. If that were true, it would imply that the sampling process may not be necessary – we could select appropriate queries once and use those queries for all data crawling tasks.

In order to show that the sampling process is necessary, we need to show that the selected queries from a sample of $TotalDB$ will not work well on another data source $TotalDB'$.

To be precise, suppose that Q is selected from $TotalDB$, i.e.,

$$Q = DWC(TotalDB, sampleSize, poolSize).$$

Suppose that Q' and Q'' are two subsets of Q that can achieve the same HR of two total databases $TotalDB$ and $totalDB'$, i.e., $Q' \subseteq Q, Q'' \subseteq Q$, such that

$$HR(Q', TotalDB) = HR(Q'', TotalDB').$$

We demonstrate that

$$OR(Q', TotalDB) < OR(Q'', TotalDB').$$

Figure 7 shows the results of applying queries to other corpora. For example, the sub figure *From Reuters corpus* shows the OR/HR relationship for queries selected from a sample of Reuters. Those queries are subsequently sent to all the four corpora. The charts show that in three cases queries selected from $SampleDB_A$ will perform better in $TotalDB_A$ than other $TotalDB$s, with the exception for Wikipedia.

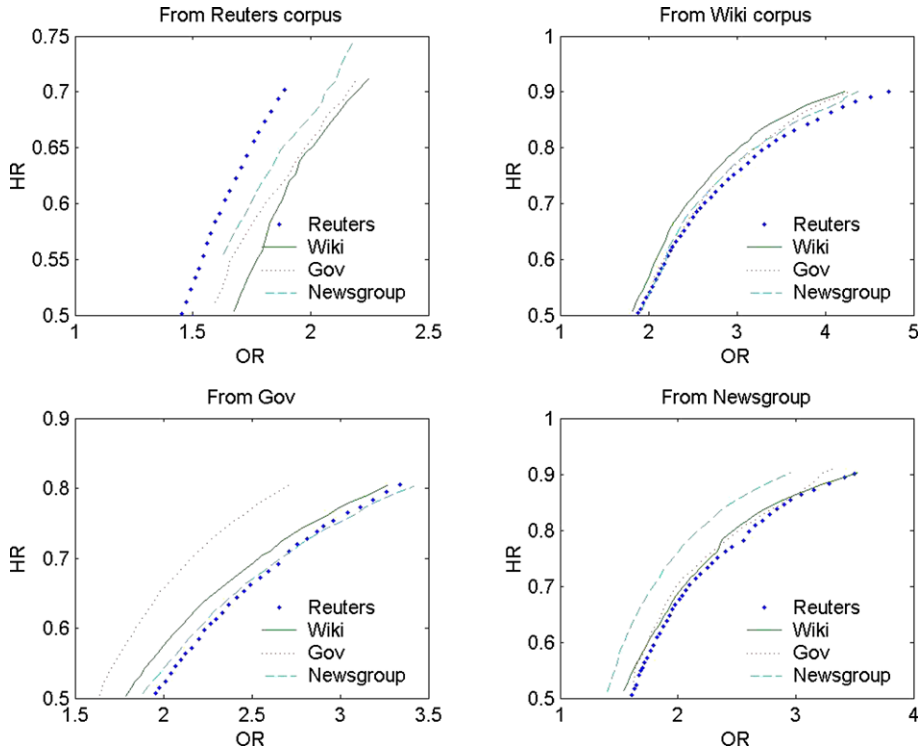

Fig. 7. Apply queries selected in one corpus to other corpora. Sample size is 3,000, relative query pool size is 20. Each sub figure shows the querying results for four corpora with queries selected from one particular corpus.

Wikipedia contains a variety of topics, resulting in samples that are representative for a variety of corpora. Hence the learnt queries works equally well for other corpora.

Comparing Fig. 7 with the random method in Fig. 3, we find that queries learnt from corpus A can also improve the performance of crawling corpus B, albeit not as good as learning from corpus B directly. An explanation for this is that some groups of words tend to co-occur more often in all corpora. By learning less overlapping words in one corpus, we break up those co-occurrent words and resulting better results in another corpus.

5.6. Select queries directly from $TotalDB$

When queries are selected directly from the $TotalDB$ instead of a $SampleDB$, those queries would certainly perform better than our method. In order to learn whether the sampling method is effective, we would like to know how much better the direct selection method is than our sampling method. Figure 8 shows the difference when the queries are selected from $SampleDB$ and $TotalDB$. In this experiment, our method sets the sample size as 4,000, the relative query pool size as 30, and the range of document frequency of the queries as 2 to 800.

The experiment shows that for all the four corpora we investigated, the sample based method performs nearly as good as the direct selection method, especially when the hit rate is close to one. In particular, the 4th sub figure for the Newsgroups corpus shows that the two approaches have almost the same performance. This can be explained that the difference between the sample size and the actual database size is not as big as other corpora because the Newsgroups corpus has 30,000 documents only.

6. Conclusions

This paper proposes an efficient and effective deep web crawling method. It can recover most of the data in a text data source with low overlapping rate. Our empirical study on the four corpora shows that using a sample of around 2,000 documents, we can efficiently select a set of queries that can cover most of the data source with low cost. We also empirically identified

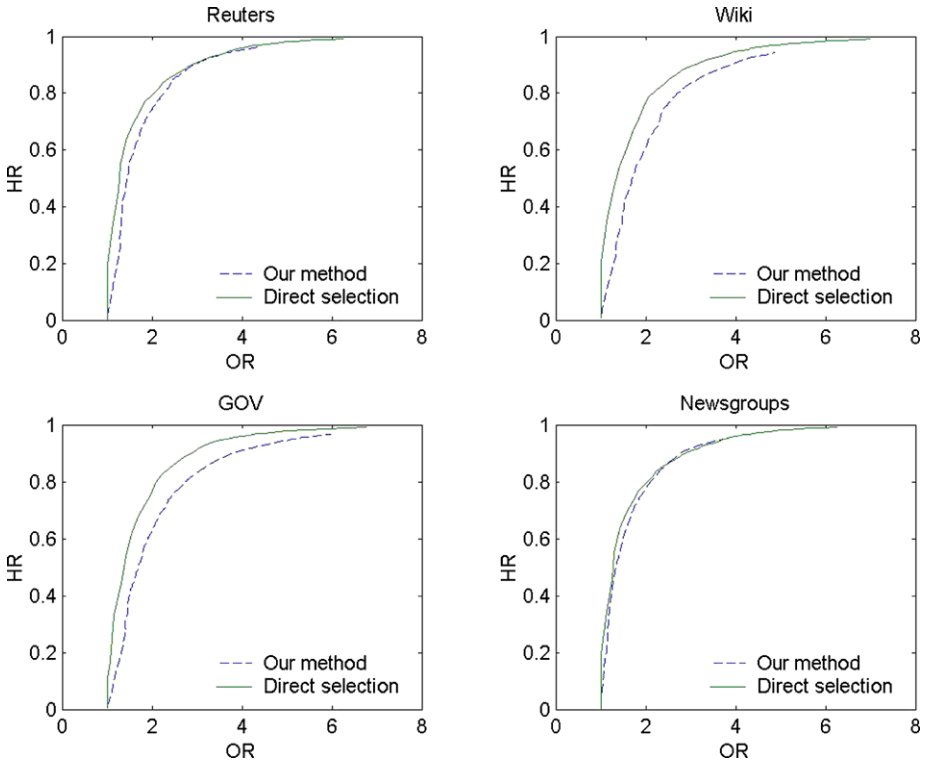

Fig. 8. Comparison with queries selected directly from $TotalDB$. Each sub figure shows the querying results for queries selected from one particular corpus.

the appropriate size for the sample and the size for the query pool.

The main contribution of the paper is that it shows a relatively small set of sample documents can be used to select the queries to efficiently cover most of the data source. Using a sample to predict the characteristics of a total population is widely used in various areas. Sampling a data source is well studied. Our Hypothesis 1 is related with the result by Callan et al. [8], which says that using around 500 documents from a sample, one can predict rather accurately the ctf (total term frequency) ratio for the total DB. That result coincides with our Hypothesis 1. However, Hypotheses 2 and 3 are proposed by us as far as we are aware of.

We will continue to explore:

Ranked data source Our method, as well as many other approaches, assumes that data sources will return all the documents. In reality, many data sources rank the matched result and return only the top k number of matches. For this type of data sources, it is almost impossible to harvest all the data due to high overlapping rate [26], hence the role of query selection is even more critical.

Multiple search attributes The scope of the paper is limited to textual database with simple keywords query interface, which is common for document searching on the web. Query forms with multiple attributes or complex query grammar are not considered in this paper. In particular, a practical data extractor should utilize the query grammar to achieve better result.

We believe that before crawling a deep web data source, there is a need to estimate its size [3,26,27]. The knowledge of size is necessary to evaluate the crawler, to decide when to stop crawling, and to estimate the document frequencies of the terms which is crucial in crawling ranked data sources [26].

Acknowledgements

We would like to thank the anonymous reviewers for their detailed comments and suggestions. The research is supported by Natural Sciences and Engineering Research Council of Canada (NSERC) and State Key Lab for Novel Software Technology at Nanjing University, China.

References

[1] M. Alvarez, et al., Extracting lists of data records from semi-structured web pages, *Data and Knowledge Engineering* **64**(2) (February 2008), 491–509, Elsevier.

[2] M. Alvarez, et al., Crawling the content hidden behind web forms, in: *Computational Science and Its Applications, ICCSA*, 2007, pp. 322–333.

[3] I. Anagnostopoulos and C. Anagnostopoulos, Estimating the size and evolution of categorised topics in web directories, in: *Web Intelligence and Agent Systems* **8**(1) (2010), 53–68, IOS Press.

[4] Z. Bar-Yossef and M. Gurevich, Random sampling from a search engine's index, in: *WWW*, 2006, pp. 367–376.

[5] L. Barbosa and J. Freire, Siphoning hidden-web data through keyword-based interfaces, in: *SBBD*, 2004.

[6] L. Barbosa and J. Freire, An adaptive crawler for locating hidden-web entry points, in: *Proc. of the 16th International Conference on World Wide Web*, 2007, pp. 441–450.

[7] M.K. Bergman, The deep web: Surfacing hidden value, *The Journal of Electronic Publishing* **7**(1) (2001).

[8] J. Callan and M. Connell, Query-based sampling of text databases, *ACM Transactions on Information Systems* (2001), 97–130.

[9] J. Caverlee, L. Liu, and D. Buttler, Probe, cluster, and discover: Focused extraction of QA-pagelets from the deep web, in: *ICDE*, 2004, pp. 103–114.

[10] C.-H. Chang, M. Kayed, M.R. Girgis, and K.F. Shaalan, A survey of web information extraction systems, *IEEE Transactions on Knowledge and Data Engineering* **18**(10) (October 2006), 1411–1428.

[11] Che, et al., *Query Rewriting for Extracting Data Behind HTML Forms*, Springer, 2004.

[12] V. Crescenzi, G. Mecca, and P. Merialdo, RoadRunner: Towards automatic data extraction from large web sites, *VLDB J.* (2001), 109–118.

[13] C. Ferris and J. Farrell, What are web services? *Commun. ACM* **46**(6) (2003), 31.

[14] E. Hatcher and O. Gospodnetic, *Lucene in Action*, Manning Publications, 2004.

[15] B. He, et al., Accessing the deep web: A survey, *Communications of the ACM (CACM)* **50**(2) (May 2007), 94–101.

[16] L. Holst, A unified approach to limit theorems for urn models, *Journal of Applied Probability* **16**(1) (March 1979), 154–162.

[17] P.G. Ipeirotis, et al., *To Search or to Crawl?: Towards a Query Optimizer for Text-Centric Tasks*, ACM Press, New York, NY, USA, 2006, pp. 265–276.

[18] R. Khare, Y. An, and I.-Y. Song, Understanding deep web search interfaces: A survey, *ACM SIGMOD Record* **39**(1) (March 2010), 33–40.

[19] C.A. Knoblock, K. Lerman, S. Minton, and I. Muslea, Accurately and reliably extracting data from the web: A machine learning approach, *IEEE Data Eng. Bull.* **23**(4) (2000), 33–41.

[20] J.P. Lage, et al., *Automatic Generation of Agents for Collecting Hidden Web Pages for Data Extraction*, Elsevier, 2004, pp. 177–196.

[21] K. Lang, Newsweeder: Learning to filter netnews, in: *Twelfth International Conference on Machine Learning*, 1995, pp. 331–339.

[22] S.W. Liddle, D.W. Embley, D.T. Scott, and S.H. Yau, Extracting data behind web forms, in: *Advanced Conceptual Modeling Techniques*, 2002, pp. 402–413.

[23] L. Jiang, Z. Wu, Q. Feng, J. Liu, and Q. Zheng, Efficient deep web crawling using reinforcement learning, in: *Advances in Knowledge Discovery and Data Mining*, Lecture Notes in Computer Science, Vol. 6118/2010, 2010, pp. 428–439.

[24] J. Lu, Efficient estimation of the size of text deep web data source, in: *CIKM*, 2008, pp. 1485–1486.

[25] J. Lu, Y. Wang, J. Liang, J. Chen, and J. Liu, An approach to deep web crawling by sampling, in: *Web Intelligence*, 2008, pp. 718–724.

[26] J. Lu, Ranking bias in deep web size estimation using capture recapture method, *Data and Knowledge Engineering*, accepted.

[27] J. Lu and D. Li, Estimating deep web data source size by capture-recapture method, *Information Retrieval* **13**(1) (2010), 70–95, Springer.

[28] C. Lund and M. Yannakakis, On the hardness of approximating minimization problems, *Journal of the ACM* **41**(5) (September 1994), 960–981.

[29] J. Madhavan, D. Ko, L. Kot, V. Ganapathy, A. Rasmussen, and A. Halevy, Google's deep-web crawl, in: *VLDB*, 2008, pp. 1241–1252.

[30] F. McCown, J.A. Smith, and M.L. Nelson, Lazy preservation: Reconstructing websites by crawling the crawlers, in: *Proc. of the 8th Annual ACM international Workshop on Web Information and Data Management (WIDM)*, 2006, pp. 67–74.

[31] K. Morita, E. Atlam, M. Fuketra, K. Tsuda, M. Oono, and J. Aoe, Word classification and hierarchy using co-occurrence word information, *Inf. Process. Manage.* **40**(6) (November 2004), 957–972.

[32] M.L. Nelson, J.A. Smith, and I.G. del Campo, *Efficient, Automatic Web Resource Harvesting*, ACM Press, New York, NY, USA, pp. 43–50, 2006.

[33] A. Ntoulas, P. Zerfos, and J. Cho, Downloading textual hidden web content through keyword queries, in: *Proc. of the Joint Conference on Digital Libraries (JCDL)*, 2005, pp. 100–109.

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

[35] V.T. Paschos, A survey of approximately optimal solutions to some covering and packing problems, *ACM Comput. Surv.* **29**(2) (June 1997), 171–209.

[36] www.ProgrammableWeb.com, 2007.

[37] S. Raghavan and H. Garcia-Molina, Crawling the hidden web, in: *VLDB*, 2001.

[38] D. Shestakov, S.S. Bhowmick, and E.-P. Lim, DEQUE: Querying the deep web, *Journal of Data Knowl. Eng.* **52**(3) (2005), 273–311.

[39] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein, *Introduction to Algorithms*, 2nd edn, MIT Press and McGraw-Hill, 2001.

[40] P. Wu, J.-R. Wen, H. Liu, and W.-Y. Ma, Query selection techniques for efficient crawling of structured web sources, in: *ICDE*, 2006, pp. 47–56.

[41] S.B. Yao, Approximating block access in database organizations, *ACM Comm.* **20**(4) (1977), 260–261.

[42] G.K. Zipf, *Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology*, Addison-Wesley, Reading, MA, 1949.