Downloading Textual Hidden Web Content Through Keyword Queries

Alexandros Ntoulas
UCLA Computer Science
ntoulas@cs.ucla.edu

Petros Zerfos
UCLA Computer Science
pzerfos@cs.ucla.edu

Junghoo Cho
UCLA Computer Science
cho@cs.ucla.edu

ABSTRACT

An ever-increasing amount of information on the Web today is available only through search interfaces: the users have to type in a set of keywords in a search form in order to access the pages from certain Web sites. These pages are often referred to as the *Hidden Web* or the *Deep Web*. Since there are no static links to the Hidden Web pages, search engines cannot discover and index such pages and thus do not return them in the results. However, according to recent studies, the content provided by many Hidden Web sites is often of very high quality and can be extremely valuable to many users.

In this paper, we study how we can build an effective *Hidden Web crawler* that can autonomously discover and download pages from the Hidden Web. Since the only "entry point" to a Hidden Web site is a query interface, the main challenge that a Hidden Web crawler has to face is how to automatically generate meaningful queries to issue to the site. Here, we provide a theoretical framework to investigate the query generation problem for the Hidden Web and we propose effective policies for generating queries automatically. Our policies proceed iteratively, issuing a different query in every iteration. We experimentally evaluate the effectiveness of these policies on 4 real Hidden Web sites and our results are very promising. For instance, in one experiment, one of our policies downloaded more than 90% of a Hidden Web site (that contains 14 million documents) after issuing fewer than 100 queries.

Categories and Subject Descriptors: H.3.7 [Information Systems]: Digital Libraries; H.3.1 [Information Systems]: Content Analysis and Indexing; H.3.3 [Information Systems]: Information Search and Retrieval.

General Terms: Algorithms, Performance, Design.

Keywords: Hidden Web crawling, Deep Web crawler, keyword queries, adaptive algorithm, query selection.

1. INTRODUCTION

Recent studies show that a significant fraction of Web content cannot be reached by following links [7, 12]. In particular, a large part of the Web is "hidden" behind search forms and is reachable only when users type in a set of keywords, or *queries*, to the forms. These pages are often referred to as the *Hidden Web* [17] or the *Deep Web* [7], because search engines typically cannot index the pages and do not return them in their results (thus, the pages are essentially "hidden" from a typical Web user).

According to many studies, the size of the Hidden Web increases rapidly as more organizations put their valuable content online through an easy-to-use Web interface [7]. In [12], Chang et al. estimate that well over 100,000 Hidden-Web sites currently exist on the Web. Moreover, the content provided by many Hidden-Web sites is often of very high quality and can be extremely valuable to many users [7]. For example, PubMed hosts many high-quality papers on medical research that were selected from careful peer-review processes, while the site of the US Patent and Trademarks Office[1] makes existing patent documents available, helping potential inventors examine "prior art."

In this paper, we study how we can build a *Hidden-Web crawler*[2] that can automatically download pages from the Hidden Web, so that search engines can index them. Conventional crawlers rely on the hyperlinks on the Web to discover pages, so current search engines cannot index the Hidden-Web pages (due to the lack of links). We believe that an effective Hidden-Web crawler can have a tremendous impact on how users search information on the Web:

- *Tapping into unexplored information*: The Hidden-Web crawler will allow an average Web user to easily explore the vast amount of information that is mostly "hidden" at present. Since a majority of Web users rely on search engines to discover pages, when pages are not indexed by search engines, they are unlikely to be viewed by many Web users. Unless users go directly to Hidden-Web sites and issue queries there, they cannot access the pages at the sites.

- *Improving user experience*: Even if a user is aware of a number of Hidden-Web sites, the user still has to waste a significant amount of time and effort, visiting all of the potentially relevant sites, querying each of them and exploring the result. By making the Hidden-Web pages searchable at a central location, we can significantly reduce the user's wasted time and effort in searching the Hidden Web.

- *Reducing potential bias*: Due to the heavy reliance of many Web users on search engines for locating information, search engines influence how the users perceive the Web [28]. Users do *not* necessarily perceive what actually *exists* on the Web, but what is *indexed* by search engines [28]. According to a recent article [5], several organizations have recognized the importance of bringing information of their Hidden Web sites onto the surface, and committed considerable resources towards this effort. Our

JCDL'05, June 7–11, 2005, Denver, Colorado, USA.

[1] US Patent Office: `http://www.uspto.gov`

[2] Crawlers are the programs that traverse the Web automatically and download pages for search engines.

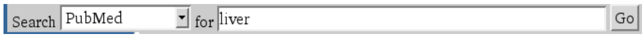

Search PubMed for liver Go

Figure 1: A single-attribute search interface

Hidden-Web crawler attempts to automate this process for Hidden Web sites with textual content, thus minimizing the associated costs and effort required.

Given that the only "entry" to Hidden Web pages is through querying a search form, there are two core challenges to implementing an effective Hidden Web crawler: (a) The crawler has to be able to understand and model a query interface, and (b) The crawler has to come up with meaningful queries to issue to the query interface. The first challenge was addressed by Raghavan and Garcia-Molina in [29], where a method for learning search interfaces was presented. Here, we present a solution to the second challenge, i.e. how a crawler can automatically generate queries so that it can discover and download the Hidden Web pages.

Clearly, when the search forms list all possible values for a query (e.g., through a drop-down list), the solution is straightforward. We exhaustively issue all possible queries, one query at a time. When the query forms have a "free text" input, however, an *infinite* number of queries are possible, so we cannot exhaustively issue all possible queries. In this case, what queries should we pick? Can the crawler automatically come up with meaningful queries without understanding the semantics of the search form?

In this paper, we provide a theoretical framework to investigate the Hidden-Web crawling problem and propose effective ways of generating queries automatically. We also evaluate our proposed solutions through experiments conducted on *real* Hidden-Web sites. In summary, this paper makes the following contributions:

- We present a formal framework to study the problem of Hidden-Web crawling. (Section 2).
- We investigate a number of crawling policies for the Hidden Web, including the optimal policy that can potentially download the maximum number of pages through the minimum number of interactions. Unfortunately, we show that the optimal policy is NP-hard and cannot be implemented in practice (Section 2.2).
- We propose a new adaptive policy that approximates the optimal policy. Our adaptive policy examines the pages returned from previous queries and adapts its query-selection policy automatically based on them (Section 3).
- We evaluate various crawling policies through experiments on real Web sites. Our experiments will show the relative advantages of various crawling policies and demonstrate their potential. The results from our experiments are very promising. In one experiment, for example, our adaptive policy downloaded more than 90% of the pages within PubMed (that contains 14 million documents) after it issued fewer than 100 queries.

2. FRAMEWORK

In this section, we present a formal framework for the study of the Hidden-Web crawling problem. In Section 2.1, we describe our assumptions on Hidden-Web sites and explain how users interact with the sites. Based on this interaction model, we present a high-level algorithm for a Hidden-Web crawler in Section 2.2. Finally in Section 2.3, we formalize the Hidden-Web crawling problem.

2.1 Hidden-Web database model

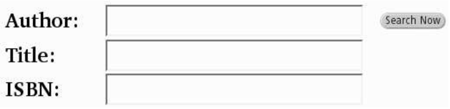


Figure 2: A multi-attribute search interface

There exists a variety of Hidden Web sources that provide information on a multitude of topics. Depending on the type of information, we may categorize a Hidden-Web site either as a *textual database* or a *structured database*. A textual database is a site that mainly contains plain-text documents, such as *PubMed* and *LexisNexis* (an online database of legal documents [1]). Since plain-text documents do not usually have well-defined structure, most textual databases provide a simple search interface where users type a list of keywords in a *single* search box (Figure 1). In contrast, a structured database often contains multi-attribute relational data (e.g., a book on the Amazon Web site may have the fields `title='Harry Potter'`, `author='J.K. Rowling'` and `isbn='0590353403'`) and supports *multi-attribute* search interfaces (Figure 2). In this paper, we will mainly focus on *textual databases* that support *single-attribute* keyword queries. We discuss how we can extend our ideas for the textual databases to multi-attribute structured databases in Section 6.1.

Typically, the users need to take the following steps in order to access pages in a Hidden-Web database:

1. **Step 1.** First, the user issues a query, say "liver," through the search interface provided by the Web site (such as the one shown in Figure 1).
2. **Step 2.** Shortly after the user issues the query, she is presented with a *result index page*. That is, the Web site returns a list of links to potentially relevant Web pages, as shown in Figure 3(a).
3. **Step 3.** From the list in the result index page, the user identifies the pages that look "interesting" and follows the links. Clicking on a link leads the user to the actual Web page, such as the one shown in Figure 3(b), that the user wants to look at.

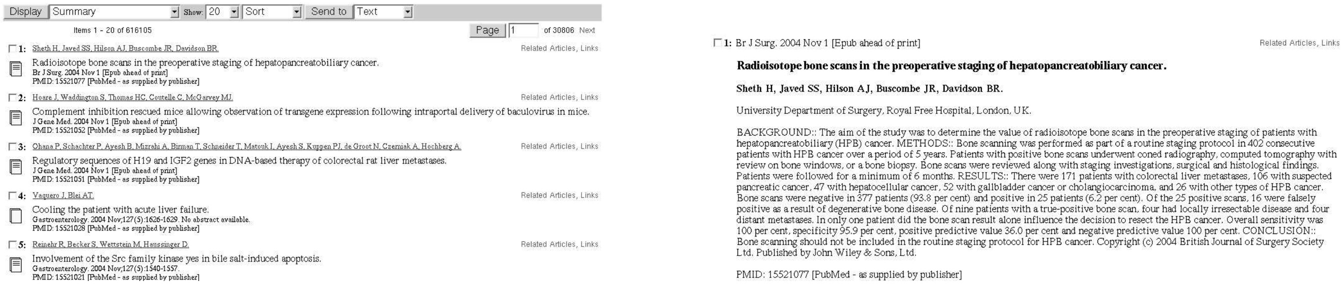
2.2 A generic Hidden Web crawling algorithm

Given that the only "entry" to the pages in a Hidden-Web site is its search from, a Hidden-Web crawler should follow the three steps described in the previous section. That is, the crawler has to generate a query, issue it to the Web site, download the result index page, and follow the links to download the actual pages. In most cases, a crawler has limited time and network resources, so the crawler repeats these steps until it uses up its resources.

In Figure 4 we show the generic algorithm for a Hidden-Web crawler. For simplicity, we assume that the Hidden-Web crawler issues single-term queries only.[3] The crawler first decides which query term it is going to use (Step (2)), issues the query, and retrieves the result index page (Step (3)). Finally, based on the links found on the result index page, it downloads the Hidden Web pages from the site (Step (4)). This same process is repeated until all the available resources are used up (Step (1)).

Given this algorithm, we can see that the most critical decision that a crawler has to make is what query to issue next. If the crawler can issue successful queries that will return many matching pages, the crawler can finish its crawling early on using minimum resources. In contrast, if the crawler issues completely irrelevant queries that do not return any matching pages, it may waste all of its resources simply issuing queries without ever retrieving actual pages. Therefore, how the crawler selects the next query can greatly affect its effectiveness. In the next section, we formalize this query selection problem.

[3] For most Web sites that assume "AND" for multi-keyword queries, single-term queries return the maximum number of results. Extending our work to multi-keyword queries is straightforward.



Items 1 - 20 of 616105

□ 1: Sheth H, Javed SS, Hilson AJ, Buscombe JR, Davidson BR. Related Articles, Links

Radioisotope bone scans in the preoperative staging of hepatopancreatobiliary cancer.
Br J Surg. 2004 Nov 1 [Epub ahead of print]
PMID: 15521077 [PubMed - as supplied by publisher]

□ 2: Hoare J, Waddington S, Thomas HC, Coutelle C, McGarvey MJ. Related Articles, Links

Complement inhibition rescued mice allowing observation of transgene expression following intraportal delivery of baculovirus in mice.
J Gene Med. 2004 Nov 1 [Epub ahead of print]
PMID: 15521052 [PubMed - as supplied by publisher]

□ 3: Ohana P, Schachter P, Ayesh B, Mizrahi A, Birman T, Schneider T, Matouk I, Ayesh S, Kuppen PJ, de Groot N, Czerniak A, Hochberg A. Related Articles, Links

Regulatory sequences of H19 and IGF2 genes in DNA-based therapy of colorectal rat liver metastases.
J Gene Med. 2004 Nov 1 [Epub ahead of print]
PMID: 15521051 [PubMed - as supplied by publisher]

□ 4: Vaquero J, Blei AT. Related Articles, Links

Cooling the patient with acute liver failure.
Gastroenterology. 2004 Nov;127(5):1626-1629. No abstract available.
PMID: 15521028 [PubMed - as supplied by publisher]

□ 5: Reinehr R, Becker S, Wettstein M, Haussinger D. Related Articles, Links

Involvement of the Src family kinase yes in bile salt-induced apoptosis.
Gastroenterology. 2004 Nov;127(5):1540-1557.
PMID: 15521021 [PubMed - as supplied by publisher]

(a) List of matching pages for query “liver”.

□ 1: Br J Surg. 2004 Nov 1 [Epub ahead of print] Related Articles, Links

Radioisotope bone scans in the preoperative staging of hepatopancreatobiliary cancer.

Sheth H, Javed SS, Hilson AJ, Buscombe JR, Davidson BR.

University Department of Surgery, Royal Free Hospital, London, UK.

BACKGROUND:: The aim of the study was to determine the value of radioisotope bone scans in the preoperative staging of patients with hepatopancreatobiliary (HPB) cancer. METHODS:: Bone scanning was performed as part of a routine staging protocol in 402 consecutive patients with HPB cancer over a period of 5 years. Patients with positive bone scans underwent coned radiography, computed tomography with review on bone windows, or a bone biopsy. Bone scans were reviewed along with staging investigations, surgical and histological findings. Patients were followed for a minimum of 6 months. RESULTS:: There were 171 patients with colorectal liver metastases, 106 with suspected pancreatic cancer, 47 with hepatocellular cancer, 52 with gallbladder cancer or cholangiocarcinoma, and 26 with other types of HPB cancer. Bone scans were negative in 377 patients (93.8 per cent) and positive in 25 patients (6.2 per cent). Of the 25 positive scans, 16 were falsely positive as a result of degenerative bone disease. Of nine patients with a true-positive bone scan, four had locally irresectable disease and four distant metastases. In only one patient did the bone scan result alone influence the decision to resect the HPB cancer. Overall sensitivity was 100 per cent, specificity 95.9 per cent, positive predictive value 36.0 per cent and negative predictive value 100 per cent. CONCLUSION:: Bone scanning should not be included in the routine staging protocol for HPB cancer. Copyright (c) 2004 British Journal of Surgery Society Ltd. Published by John Wiley & Sons, Ltd.

PMID: 15521077 [PubMed - as supplied by publisher]

(b) The first matching page for “liver”.

Figure 3: Pages from the PubMed Web site.

```
ALGORITHM 2.1.   Crawling a Hidden Web site
Procedure
while ( there are available resources ) do
    // select a term to send to the site
    $q_i$ = SelectTerm()
    // send query and acquire result index page
    $R(q_i)$ = QueryWebSite( $q_i$ )
    // download the pages of interest
    Download( $R(q_i)$ )
done
```

Figure 4: Algorithm for crawling a Hidden Web site.

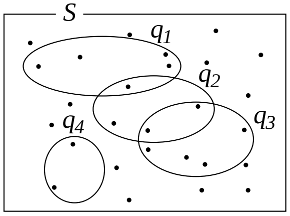


Figure 5: A set-formalization of the optimal query selection problem.

2.3 Problem formalization

Theoretically, the problem of query selection can be formalized as follows: We assume that the crawler downloads pages from a Web site that has a set of pages S (the rectangle in Figure 5). We represent each Web page in S as a point (dots in Figure 5). Every potential query q_i that we may issue can be viewed as a subset of S, containing all the points (pages) that are returned when we issue q_i to the site. Each subset is associated with a weight that represents the cost of issuing the query. Under this formalization, our goal is to find which subsets (queries) cover the maximum number of points (Web pages) with the minimum total weight (cost). This problem is equivalent to the *set-covering* problem in graph theory [16].

There are two main difficulties that we need to address in this formalization. First, in a practical situation, the crawler does not know which Web pages will be returned by which queries, so the subsets of S are not known in advance. Without knowing these subsets the crawler cannot decide which queries to pick to maximize the coverage. Second, the set-covering problem is known to be NP-Hard [16], so an efficient algorithm to solve this problem optimally in polynomial time has yet to be found.

In this paper, we will present an *approximation algorithm* that can find a near-optimal solution at a reasonable computational cost. Our algorithm leverages the observation that although we do not know which pages will be returned by each query q_i that we issue, we can *predict how many* pages will be returned. Based on this information our query selection algorithm can then select the “best” queries that cover the content of the Web site. We present our prediction method and our query selection algorithm in Section 3.

2.3.1 Performance Metric

Before we present our ideas for the query selection problem, we briefly discuss some of our notation and the cost/performance metrics.

Given a query q_i, we use $P(q_i)$ to denote the fraction of pages that we will get back if we issue query q_i to the site. For example, if a Web site has 10,000 pages in total, and if 3,000 pages are returned for the query $q_i =$ “*medicine*”, then $P(q_i) = 0.3$. We use $P(q_1 \wedge q_2)$ to represent the fraction of pages that are returned from both q_1 and q_2 (i.e., the intersection of $P(q_1)$ and $P(q_2)$). Similarly, we use $P(q_1 \vee q_2)$ to represent the fraction of pages that are returned from either q_1 or q_2 (i.e., the union of $P(q_1)$ and $P(q_2)$).

We also use *Cost*(q_i) to represent the cost of issuing the query q_i. Depending on the scenario, the cost can be measured either in time, network bandwidth, the number of interactions with the site, or it can be a function of all of these. As we will see later, our proposed algorithms are independent of the exact cost function.

In the most common case, the query cost consists of a number of factors, including the cost for submitting the query to the site, retrieving the result index page (Figure 3(a)) and downloading the actual pages (Figure 3(b)). We assume that submitting a query incurs a fixed cost of c_q. The cost for downloading the result index page is proportional to the number of matching documents to the query, while the cost c_d for downloading a matching document is also fixed. Then the overall cost of query q_i is

$$\textit{Cost}(q_i) = c_q + c_r P(q_i) + c_d P(q_i). \tag{1}$$

In certain cases, some of the documents from q_i may have already been downloaded from previous queries. In this case, the crawler may skip downloading these documents and the cost of q_i can be

$$\textit{Cost}(q_i) \ = c_q + c_r P(q_i) + c_d P_{new}(q_i). \tag{2}$$

Here, we use $P_{new}(q_i)$ to represent the fraction of the *new* documents from q_i that have *not* been retrieved from previous queries. Later in Section 3.1 we will study how we can estimate $P(q_i)$ and $P_{new}(q_i)$ to estimate the cost of q_i.

Since our algorithms are independent of the exact cost function, we will assume a generic cost function *Cost*(q_i) in this paper. When we need a concrete cost function, however, we will use Equation 2.

Given the notation, we can formalize the goal of a Hidden-Web crawler as follows:

PROBLEM 1. *Find the set of queries $q_1, \ldots, q_n$ that maximizes*

$$P(q_1 \vee \cdots \vee q_n)$$

under the constraint

$$\sum_{i=1}^{n} \mathit{Cost}(q_i) \leq t.$$

Here, t is the maximum download resource that the crawler has.

3. KEYWORD SELECTION

How should a crawler select the queries to issue? Given that the goal is to download the maximum number of unique documents from a textual database, we may consider one of the following options:

- *Random*: We select random keywords from, say, an English dictionary and issue them to the database. The hope is that a random query will return a reasonable number of matching documents.
- *Generic-frequency*: We analyze a generic document corpus collected elsewhere (say, from the Web) and obtain the generic frequency distribution of each keyword. Based on this generic distribution, we start with the most frequent keyword, issue it to the Hidden-Web database and retrieve the result. We then continue to the second-most frequent keyword and repeat this process until we exhaust all download resources. The hope is that the frequent keywords in a generic corpus will also be frequent in the Hidden-Web database, returning many matching documents.
- *Adaptive*: We analyze the documents returned from the previous queries issued to the Hidden-Web database and estimate which keyword is most likely to return the most documents. Based on this analysis, we issue the most "promising" query, and repeat the process.

Among these three general policies, we may consider the random policy as the *base comparison point* since it is expected to perform the worst. Between the generic-frequency and the adaptive policies, both policies may show similar performance if the crawled database has a *generic* document collection without a specialized topic. The adaptive policy, however, may perform significantly better than the generic-frequency policy if the database has a very specialized collection that is different from the generic corpus. We will experimentally compare these three policies in Section 4.

While the first two policies (random and generic-frequency policies) are easy to implement, we need to understand how we can analyze the downloaded pages to identify the most "promising" query in order to implement the adaptive policy. We address this issue in the rest of this section.

3.1 Estimating the number of matching pages

In order to identify the most promising query, we need to estimate how many new documents we will download if we issue the query q_i as the next query. That is, assuming that we have issued queries $q_1, \ldots, q_{i-1}$ we need to estimate $P(q_1 \vee \cdots \vee q_{i-1} \vee q_i)$, for every potential next query q_i and compare this value. In estimating this number, we note that we can rewrite $P(q_1 \vee \cdots \vee q_{i-1} \vee q_i)$ as:

$$\begin{aligned}
&P((q_1 \vee \cdots \vee q_{i-1}) \vee q_i) \\
&= P(q_1 \vee \cdots \vee q_{i-1}) + P(q_i) - P((q_1 \vee \cdots \vee q_{i-1}) \wedge q_i) \\
&= P(q_1 \vee \cdots \vee q_{i-1}) + P(q_i) \\
&\quad - P(q_1 \vee \cdots \vee q_{i-1}) P(q_i | q_1 \vee \cdots \vee q_{i-1}) \qquad (3)
\end{aligned}$$

In the above formula, note that we can precisely measure $P(q_1 \vee \cdots \vee q_{i-1})$ and $P(q_i \mid q_1 \vee \cdots \vee q_{i-1})$ by analyzing previously-downloaded pages: We know $P(q_1 \vee \cdots \vee q_{i-1})$, the union of all pages downloaded from $q_1, \ldots, q_{i-1}$, since we have already issued $q_1, \ldots, q_{i-1}$ and downloaded the matching pages.[4] We can also measure $P(q_i \mid q_1 \vee \cdots \vee q_{i-1})$, the probability that q_i appears in the pages from $q_1, \ldots, q_{i-1}$, by counting how many times q_i appears in the pages from $q_1, \ldots, q_{i-1}$. Therefore, we only need to estimate $P(q_i)$ to evaluate $P(q_1 \vee \cdots \vee q_i)$. We may consider a number of different ways to estimate $P(q_i)$, including the following:

1. *Independence estimator*: We assume that the appearance of the term q_i is independent of the terms $q_1, \ldots, q_{i-1}$. That is, we assume that $P(q_i) = P(q_i | q_1 \vee \cdots \vee q_{i-1})$.
2. *Zipf estimator*: In [19], Ipeirotis et al. proposed a method to estimate how many times a particular term occurs in the *entire* corpus based on a subset of documents from the corpus. Their method exploits the fact that the frequency of terms inside text collections follows a power law distribution [30, 25]. That is, if we rank all terms based on their occurrence frequency (with the most frequent term having a rank of 1, second most frequent a rank of 2 etc.), then the frequency f of a term inside the text collection is given by:

   $$f = \alpha(r + \beta)^{-\gamma} \qquad (4)$$

   where r is the rank of the term and α, β, and γ are constants that depend on the text collection.

   Their main idea is (1) to estimate the three parameters, α, β and γ, based on the subset of documents that we have downloaded from previous queries, and (2) use the estimated parameters to predict f given the ranking r of a term within the subset. For a more detailed description on how we can use this method to estimate $P(q_i)$, we refer the reader to the extended version of this paper [27].

After we estimate $P(q_i)$ and $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ values, we can calculate $P(q_1 \vee \cdots \vee q_i)$. In Section 3.3, we explain how we can efficiently compute $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ by maintaining a succinct summary table. In the next section, we first examine how we can use this value to decide which query we should issue next to the Hidden Web site.

3.2 Query selection algorithm

The goal of the Hidden-Web crawler is to download the maximum number of unique documents from a database using its limited download resources. Given this goal, the Hidden-Web crawler has to take two factors into account. (1) the number of new documents that can be obtained from the query q_i and (2) the cost of issuing the query q_i. For example, if two queries, q_i and q_j, incur the same cost, but q_i returns more new pages than q_j, q_i is more desirable than q_j. Similarly, if q_i and q_j return the same number of new documents, but q_i incurs less cost then q_j, q_i is more desirable. Based on this observation, the Hidden-Web crawler may use the following *efficiency* metric to quantify the desirability of the query q_i:

$$\mathit{Efficiency}(q_i) = \frac{P_{new}(q_i)}{\mathit{Cost}(q_i)}$$

Here, $P_{new}(q_i)$ represents the amount of *new* documents returned for q_i (the pages that have not been returned for previous queries). *Cost*(q_i) represents the cost of issuing the query q_i.

Intuitively, the efficiency of q_i measures how many new documents are retrieved per unit cost, and can be used as an indicator of

[4] For exact estimation, we need to know the total number of pages in the site. However, in order to compare only relative values among queries, this information is not actually needed.

```
ALGORITHM 3.1.   Greedy SelectTerm()
Parameters:
  $T$: The list of potential query keywords
Procedure
  (1)  Foreach $t_k$ in $T$ do
  (2)      Estimate Efficiency$(t_k) = \frac{P_{new}(t_k)}{Cost(t_k)}$
  (3)  done
  (4)  return $t_k$ with maximum Efficiency$(t_k)$
```

Figure 6: Algorithm for selecting the next query term.

how well our resources are spent when issuing q_i. Thus, the Hidden Web crawler can estimate the efficiency of every candidate q_i, and select the one with the highest value. By using its resources more efficiently, the crawler may eventually download the maximum number of unique documents. In Figure 6, we show the query selection function that uses the concept of efficiency. In principle, this algorithm takes a *greedy approach* and tries to maximize the "potential gain" in every step.

We can estimate the efficiency of every query using the estimation method described in Section 3.1. That is, the size of the new documents from the query q_i, $P_{new}(q_i)$, is

$$\begin{aligned} & P_{new}(q_i) \\ & = P(q_1 \vee \cdots \vee q_{i-1} \vee q_i) - P(q_1 \vee \cdots \vee q_{i-1}) \\ & = P(q_i) - P(q_1 \vee \cdots \vee q_{i-1}) P(q_i | q_1 \vee \cdots \vee q_{i-1}) \end{aligned}$$

from Equation 3, where $P(q_i)$ can be estimated using one of the methods described in section 3. We can also estimate *Cost*(q_i) similarly. For example, if *Cost*(q_i) is

$$Cost(q_i) \; = \; c_q + c_r P(q_i) + c_d P_{new}(q_i)$$

(Equation 2), we can estimate *Cost*(q_i) by estimating $P(q_i)$ and $P_{new}(q_i)$.

3.3 Efficient calculation of query statistics

In estimating the efficiency of queries, we found that we need to measure $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ for every potential query q_i. This calculation can be very time-consuming if we repeat it from scratch for every query q_i in every iteration of our algorithm. In this section, we explain how we can compute $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ efficiently by maintaining a small table that we call a *query statistics table*.

The main idea for the query statistics table is that $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ can be measured by counting how many times the keyword q_i appears within the documents downloaded from $q_1, \ldots, q_{i-1}$. We record these counts in a table, as shown in Figure 7(a). The left column of the table contains all potential query terms and the right column contains the number of previously-downloaded documents containing the respective term. For example, the table in Figure 7(a) shows that we have downloaded 50 documents so far, and the term *model* appears in 10 of these documents. Given this number, we can compute that $P(model | q_1 \vee \cdots \vee q_{i-1}) = \frac{10}{50} = 0.2$.

We note that the query statistics table needs to be updated whenever we issue a new query q_i and download more documents. This update can be done efficiently as we illustrate in the following example.

Term t_k	$N(t_k)$
model	10
computer	38
digital	50

Total pages: 50

(a) After $q_1, \ldots, q_{i-1}$

Term t_k	$N(t_k)$
model	12
computer	20
disk	18

New pages: 20

(b) New from $q_i = computer$

Term t_k	$N(t_k)$
model	10+12 = 22
computer	38+20 = 58
disk	0+18 = 18
digital	50+0 = 50

Total pages: $50 + 20 = 70$

(c) After $q_1, \ldots, q_i$

Figure 7: Updating the query statistics table.

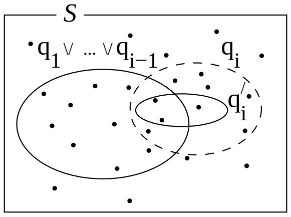


Figure 8: A Web site that does not return all the results.

EXAMPLE 1. *After examining the* query statistics table *of Figure 7(a), we have decided to use the term "computer" as our next query* q_i. *From the new query* q_i = "computer," *we downloaded* 20 *more new pages. Out of these,* 12 *contain the keyword "model" and* 18 *the keyword "disk." The table in Figure 7(b) shows the frequency of each term in the newly-downloaded pages.*

We can update the old table (Figure 7(a)) to include this new information by simply adding corresponding entries in Figures 7(a) and (b). The result is shown on Figure 7(c). For example, keyword "model" exists in $10 + 12 = 22$ *pages within the pages retrieved from* $q_1, \ldots, q_i$. *According to this new table,* $P(model | q_1 \vee \cdots \vee q_i)$ *is now* $\frac{22}{70} = 0.3$.

3.4 Crawling sites that limit the number of results

In certain cases, when a query matches a large number of pages, the Hidden Web site returns only a portion of those pages. For example, the Open Directory Project [2] allows the users to see only up to $10,000$ results after they issue a query. Obviously, this kind of limitation has an immediate effect on our Hidden Web crawler. First, since we can only retrieve up to a specific number of pages per query, our crawler will need to issue more queries (and potentially will use up more resources) in order to download all the pages. Second, the query selection method that we presented in Section 3.2 assumes that for every potential query q_i, we can find $P(q_i | q_1 \vee \cdots \vee q_{i-1})$. That is, for every query q_i we can find the fraction of documents in the *whole* text database that contains q_i with at least one of $q_1, \ldots, q_{i-1}$. However, if the text database returned only a portion of the results for any of the $q_1, \ldots, q_{i-1}$ then the value $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ is not accurate and may affect our decision for the next query q_i, and potentially the performance of our crawler. Since we cannot retrieve more results per query than the maximum number the Web site allows, our crawler has no other choice besides submitting more queries. However, there is a way to *estimate* the correct value for $P(q_i | q_1 \vee \cdots \vee q_{i-1})$ in the case where the Web site returns only a portion of the results.

Again, assume that the Hidden Web site we are currently crawling is represented as the rectangle on Figure 8 and its pages as points in the figure. Assume that we have already issued queries $q_1, \ldots, q_{i-1}$ which returned a number of results less than the maximum number than the site allows, and therefore we have downloaded all the pages for these queries (big circle in Figure 8). That is, at this point, our estimation for $P(q_i|q_1 \vee \cdots \vee q_{i-1})$ is accurate. Now assume that we submit query q_i to the Web site, but due to a limitation in the number of results that we get back, we retrieve the set q_i' (small circle in Figure 8) instead of the set q_i (dashed circle in Figure 8). Now we need to update our *query statistics table* so that it has accurate information for the next step. That is, although we got the set q_i' back, for every potential query q_{i+1} we need to find $P(q_{i+1}|q_1 \vee \cdots \vee q_i)$:

$$\begin{aligned} &P(q_{i+1}|q_1 \vee \cdots \vee q_i) \\ &= \frac{1}{P(q_1 \vee \cdots \vee q_i)} \cdot [P(q_{i+1} \wedge (q_1 \vee \cdots \vee q_{i-1}))+ \\ &\quad P(q_{i+1} \wedge q_i) - P(q_{i+1} \wedge q_i \wedge (q_1 \vee \cdots \vee q_{i-1}))] \end{aligned} \tag{5}$$

In the previous equation, we can find $P(q_1 \vee \cdots \vee q_i)$ by estimating $P(q_i)$ with the method shown in Section 3. Additionally, we can calculate $P(q_{i+1} \wedge (q_1 \vee \cdots \vee q_{i-1}))$ and $P(q_{i+1} \wedge q_i \wedge (q_1 \vee \cdots \vee q_{i-1}))$ by directly examining the documents that we have downloaded from queries $q_1, \ldots, q_{i-1}$. The term $P(q_{i+1} \wedge q_i)$ however is unknown and we need to estimate it. Assuming that q_i' is a random sample of q_i, then:

$$\frac{P(q_{i+1} \wedge q_i)}{P(q_{i+1} \wedge q_i')} = \frac{P(q_i)}{P(q_i')} \tag{6}$$

From Equation 6 we can calculate $P(q_{i+1} \wedge q_i)$ and after we replace this value to Equation 5 we can find $P(q_{i+1}|q_1 \vee \cdots \vee q_i)$.

4. EXPERIMENTAL EVALUATION

In this section we experimentally evaluate the performance of the various algorithms for Hidden Web crawling presented in this paper. Our goal is to validate our theoretical analysis through real-world experiments, by crawling popular Hidden Web sites of textual databases. Since the number of documents that are discovered and downloaded from a textual database depends on the selection of the words that will be issued as queries[5] to the search interface of each site, we compare the various selection policies that were described in section 3, namely the *random*, *generic-frequency*, and *adaptive* algorithms.

The *adaptive* algorithm learns new keywords and terms from the documents that it downloads, and its selection process is driven by a cost model as described in Section 3.2. To keep our experiment and its analysis simple at this point, we will assume that the cost for every query is constant. That is, our goal is to maximize the number of downloaded pages by issuing the least number of queries. Later, in Section 4.4 we will present a comparison of our policies based on a more elaborate cost model. In addition, we use the *independence estimator* (Section 3.1) to estimate $P(q_i)$ from downloaded pages. Although the *independence estimator* is a simple estimator, our experiments will show that it can work very well in practice.[6]

For the *generic-frequency* policy, we compute the frequency distribution of words that appear in a 5.5-million-Web-page corpus downloaded from 154 Web sites of various topics [26]. Keywords are selected based on their decreasing frequency with which they appear in this document set, with the most frequent one being selected first, followed by the second-most frequent keyword, etc.[7]

Regarding the *random* policy, we use the same set of words collected from the Web corpus, but in this case, instead of selecting keywords based on their relative frequency, we choose them randomly (uniform distribution). In order to further investigate how the quality of the potential query-term list affects the random-based algorithm, we construct two sets: one with the $16,000$ most frequent words of the term collection used in the generic-frequency policy (hereafter, the random policy with the set of 16,000 words will be referred to as *random-16K*), and another set with the 1 million most frequent words of the same collection as above (hereafter, referred to as *random-1M*). The former set has frequent words that appear in a large number of documents (at least $10,000$ in our collection), and therefore can be considered of "high-quality" terms. The latter set though contains a much larger collection of words, among which some might be bogus, and meaningless.

The experiments were conducted by employing each one of the aforementioned algorithms (adaptive, generic-frequency, random-16K, and random-1M) to crawl and download contents from three Hidden Web sites: The PubMed Medical Library,[8] Amazon,[9] and the Open Directory Project[2]. According to the information on *PubMed*'s Web site, its collection contains approximately 14 million abstracts of biomedical articles. We consider these abstracts as the "documents" in the site, and in each iteration of the adaptive policy, we use these abstracts as input to the algorithm. Thus our goal is to "discover" as many unique abstracts as possible by repeatedly querying the Web query interface provided by PubMed. The Hidden Web crawling on the PubMed Web site can be considered as topic-specific, due to the fact that all abstracts within PubMed are related to the fields of medicine and biology.

In the case of the *Amazon* Web site, we are interested in downloading all the hidden pages that contain information on books. The querying to Amazon is performed through the Software Developer's Kit that Amazon provides for interfacing to its Web site, and which returns results in XML form. The generic "keyword" field is used for searching, and as input to the adaptive policy we extract the product description and the text of customer reviews when present in the XML reply. Since Amazon does not provide any information on how many books it has in its catalogue, we use random sampling on the 10-digit ISBN number of the books to estimate the size of the collection. Out of the $10,000$ random ISBN numbers queried, 46 are found in the Amazon catalogue, therefore the size of its book collection is estimated to be $\frac{46}{10000} \cdot 10^{10} = 4.6$ million books. It's also worth noting here that Amazon poses an upper limit on the number of results (books in our case) returned by each query, which is set to $32,000$.

As for the third Hidden Web site, the *Open Directory Project* (hereafter also referred to as *dmoz*), the site maintains the links to 3.8 million sites together with a brief summary of each listed site. The links are searchable through a keyword-search interface. We consider each indexed link together with its brief summary as the document of the *dmoz* site, and we provide the short summaries to the adaptive algorithm to drive the selection of new keywords for querying. On the dmoz Web site, we perform two Hidden Web crawls: the first is on its generic collection of 3.8-million indexed

[5] Throughout our experiments, once an algorithm has submitted a query to a database, we exclude the query from subsequent submissions to the same database from the same algorithm.

[6] We defer the reporting of results based on the *Zipf estimation* to a future work.

[7] We did not manually exclude *stop words* (e.g., the, is, of, etc.) from the keyword list. As it turns out, all Web sites except PubMed return matching documents for the stop words, such as "the."

[8] PubMed Medical Library: `http://www.pubmed.org`

[9] Amazon Inc.: `http://www.amazon.com`

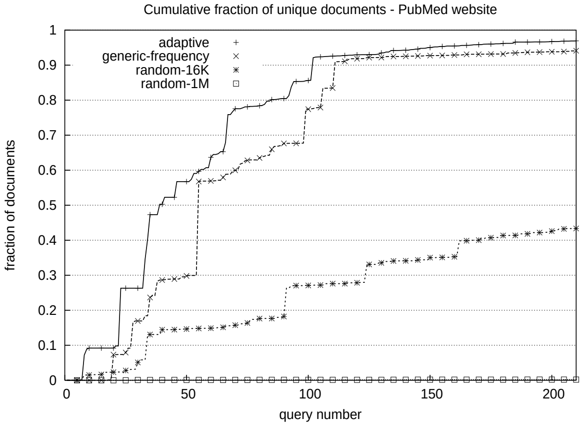


Figure 9: Coverage of policies for Pubmed

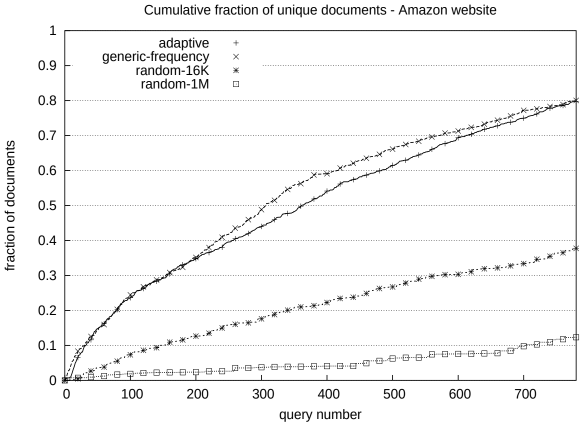


Figure 10: Coverage of policies for Amazon

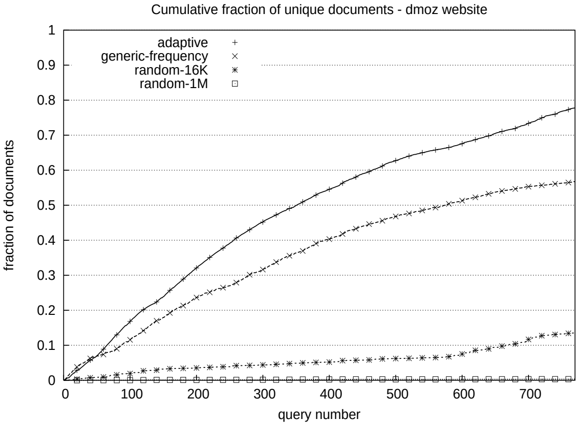


Figure 11: Coverage of policies for general dmoz

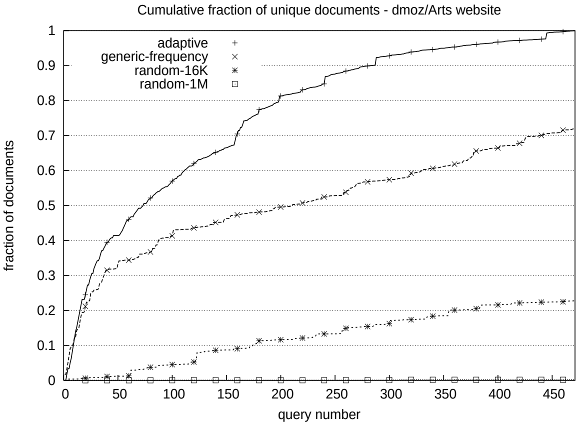


Figure 12: Coverage of policies for the Arts section of dmoz

sites, regardless of the category that they fall into. The other crawl is performed specifically on the Arts section of *dmoz* (`http://dmoz.org/Arts`), which comprises of approximately $429,000$ indexed sites that are relevant to Arts, making this crawl topic-specific, as in PubMed. Like Amazon, *dmoz* also enforces an upper limit on the number of returned results, which is $10,000$ links with their summaries.

4.1 Comparison of policies

The first question that we seek to answer is the evolution of the *coverage* metric as we submit queries to the sites. That is, what fraction of the collection of documents stored in the Hidden Web site can we download as we continuously query for new words selected using the policies described above? More formally, we are interested in the value of $P(q_1 \vee \cdots \vee q_{i-1} \vee q_i)$, after we submit $q_1, \ldots, q_i$ queries, and as i increases.

In Figures 9, 10, 11, and 12 we present the coverage metric for each policy, as a function of the query number, for the Web sites of PubMed, Amazon, general *dmoz* and the art-specific *dmoz*, respectively. On the y-axis the fraction of the total documents downloaded from the website is plotted, while the x-axis represents the query number. A first observation from these graphs is that in general, the generic-frequency and the adaptive policies perform much better than the random-based algorithms. In all of the figures, the graphs for the random-1M and the random-16K are significantly below those of other policies.

Between the generic-frequency and the adaptive policies, we can see that the latter outperforms the former when the site is topic specific. For example, for the PubMed site (Figure 9), the adaptive algorithm issues only 83 queries to download almost 80% of the documents stored in PubMed, but the generic-frequency algorithm requires 106 queries for the same coverage,. For the dmoz/Arts crawl (Figure 12), the difference is even more substantial: the adaptive policy is able to download 99.98% of the total sites indexed in the Directory by issuing 471 queries, while the frequency-based algorithm is much less effective using the same number of queries, and discovers only 72% of the total number of indexed sites. The adaptive algorithm, by examining the contents of the pages that it downloads at each iteration, is able to identify the topic of the site as expressed by the words that appear most frequently in the result-set. Consequently, it is able to select words for subsequent queries that are more relevant to the site, than those preferred by the generic-frequency policy, which are drawn from a large, generic collection. Table 1 shows a sample of 10 keywords out of 211 chosen and submitted to the PubMed Web site by the adaptive algorithm, but not by the other policies. For each keyword, we present the number of the iteration, along with the number of results that it returned. As one can see from the table, these keywords are highly relevant to the topics of medicine and biology of the Public Medical Library, and match against numerous articles stored in its Web site.

In both cases examined in Figures 9, and 12, the random-based policies perform much worse than the adaptive algorithm, and the generic-frequency. It is worthy noting however, that the random-based policy with the small, carefully selected set of $16,000$ "quality" words manages to download a considerable fraction of 42.5%

Iteration	*Keyword*	*Number of Results*
23	department	2,719,031
34	patients	1,934,428
53	clinical	1,198,322
67	treatment	4,034,565
69	medical	1,368,200
70	hospital	503,307
146	disease	1,520,908
172	protein	2,620,938

Table 1: Sample of keywords queried to PubMed exclusively by the adaptive policy

from the PubMed Web site after 200 queries, while the coverage for the Arts section of dmoz reaches 22.7%, after 471 queried keywords. On the other hand, the random-based approach that makes use of the vast collection of 1 million words, among which a large number is bogus keywords, fails to download even a mere 1% of the total collection, after submitting the same number of query words.

For the generic collections of Amazon and the *dmoz* sites, shown in Figures 10 and 11 respectively, we get mixed results: The generic-frequency policy shows slightly better performance than the adaptive policy for the Amazon site (Figure 10), and the adaptive method clearly outperforms the generic-frequency for the general *dmoz* site (Figure 11). A closer look at the log files of the two Hidden Web crawlers reveals the main reason: Amazon was functioning in a very flaky way when the adaptive crawler visited it, resulting in a large number of lost results. Thus, we suspect that the slightly poor performance of the adaptive policy is due to this experimental variance. We are currently running another experiment to verify whether this is indeed the case. Aside from this experimental variance, the Amazon result indicates that if the collection and the words that a Hidden Web site contains are generic enough, then the generic-frequency approach may be a good candidate algorithm for effective crawling.

As in the case of topic-specific Hidden Web sites, the random-based policies also exhibit poor performance compared to the other two algorithms when crawling generic sites: for the Amazon Web site, random-16K succeeds in downloading almost 36.7% after issuing 775 queries, alas for the generic collection of dmoz, the fraction of the collection of links downloaded is 13.5% after the 770th query. Finally, as expected, random-1M is even worse than random-16K, downloading only 14.5% of Amazon and 0.3% of the generic dmoz.

In summary, the adaptive algorithm performs remarkably well in all cases: it is able to discover and download most of the documents stored in Hidden Web sites by issuing the least number of queries. When the collection refers to a specific topic, it is able to identify the keywords most relevant to the topic of the site and consequently ask for terms that is most likely that will return a large number of results . On the other hand, the generic-frequency policy proves to be quite effective too, though less than the adaptive: it is able to retrieve relatively fast a large portion of the collection, and when the site is not topic-specific, its effectiveness can reach that of adaptive (e.g. Amazon). Finally, the random policy performs poorly in general, and should not be preferred.

4.2 Impact of the initial query

An interesting issue that deserves further examination is whether the initial choice of the keyword used as the first query issued by the adaptive algorithm affects its effectiveness in subsequent iterations. The choice of this keyword is not done by the selection of the adaptive algorithm itself and has to be manually set, since its query statistics tables have not been populated yet. Thus, the selection is generally arbitrary, so for purposes of fully automating the whole process, some additional investigation seems necessary.

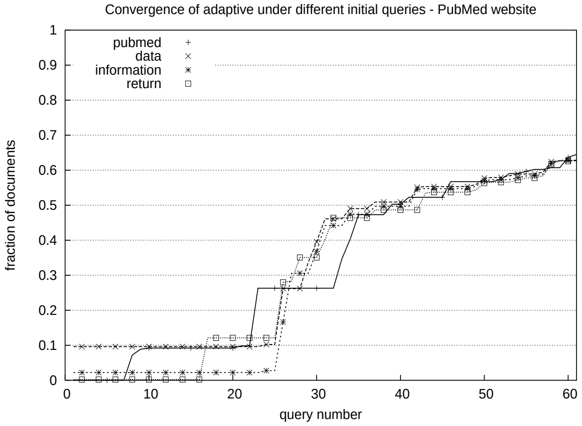


Figure 13: Convergence of the adaptive algorithm using different initial queries for crawling the PubMed Web site

For this reason, we initiated three adaptive Hidden Web crawlers targeting the PubMed Web site with different seed-words: the word “data”, which returns 1,344,999 results, the word “information” that reports 308,474 documents, and the word “return” that retrieves 29,707 pages, out of 14 million. These keywords represent varying degrees of term popularity in PubMed, with the first one being of high popularity, the second of medium, and the third of low. We also show results for the keyword “pubmed”, used in the experiments for coverage of Section 4.1, and which returns 695 articles. As we can see from Figure 13, after a small number of queries, all four crawlers roughly download the same fraction of the collection, regardless of their starting point: Their coverages are roughly equivalent from the 25th query. Eventually, all four crawlers use the same set of terms for their queries, regardless of the initial query. In the specific experiment, from the 36th query onward, all four crawlers use the same terms for their queries in each iteration, or the same terms are used off by one or two query numbers. Our result confirms the observation of [11] that the choice of the initial query has minimal effect on the final performance. We can explain this intuitively as follows: Our algorithm approximates the optimal set of queries to use for a particular Web site. Once the algorithm has issued a significant number of queries, it has an accurate estimation of the content of the Web site, regardless of the initial query. Since this estimation is similar for all runs of the algorithm, the crawlers will use roughly the same queries.

4.3 Impact of the limit in the number of results

While the Amazon and dmoz sites have the respective limit of 32,000 and 10,000 in their result sizes, these limits may be larger than those imposed by other Hidden Web sites. In order to investigate how a “tighter” limit in the result size affects the performance of our algorithms, we performed two additional crawls to the generic-dmoz site: we ran the generic-frequency and adaptive policies but we retrieved only up to the top 1,000 results for every query. In Figure 14 we plot the coverage for the two policies as a function of the number of queries. As one might expect, by comparing the new result in Figure 14 to that of Figure 11 where the result limit was 10,000, we conclude that the tighter limit requires a higher number of queries to achieve the same coverage. For example, when the result limit was 10,000, the adaptive pol-

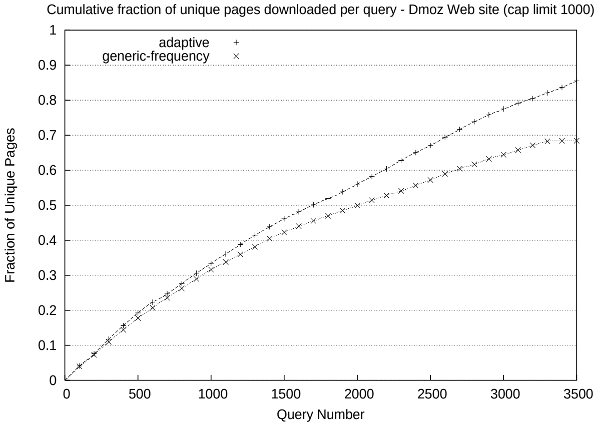


Figure 14: Coverage of general dmoz after limiting the number of results to 1,000

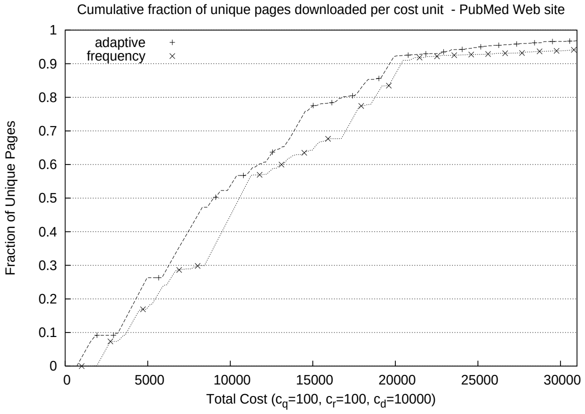


Figure 15: Coverage of PubMed after incorporating the document download cost

icy could download 70% of the site after issuing 630 queries, while it had to issue 2,600 queries to download 70% of the site when the limit was 1,000. On the other hand, our new result shows that even with a tight result limit, it is still possible to download most of a Hidden Web site after issuing a reasonable number of queries. The adaptive policy could download more than 85% of the site after issuing 3,500 queries when the limit was 1,000. Finally, our result shows that our adaptive policy consistently outperforms the generic-frequency policy regardless of the result limit. In both Figure 14 and Figure 11, our adaptive policy shows significantly larger coverage than the generic-frequency policy for the same number of queries.

4.4 Incorporating the document download cost

For brevity of presentation, the performance evaluation results provided so far assumed a simplified cost-model where every query involved a constant cost. In this section we present results regarding the performance of the adaptive and generic-frequency algorithms using Equation 2 to drive our query selection process. As we discussed in Section 2.3.1, this query cost model includes the cost for submitting the query to the site, retrieving the result index page, and also downloading the actual pages. For these costs, we examined the size of every result in the index page and the sizes of the documents, and we chose $c_q = 100$, $c_r = 100$, and $c_d = 10000$, as values for the parameters of Equation 2, and for the particular experiment that we ran on the PubMed website. The values that we selected imply that the cost for issuing one query and retrieving one result from the result index page are roughly the same, while the cost for downloading an actual page is 100 times larger. We believe that these values are reasonable for the PubMed Web site.

Figure 15 shows the coverage of the adaptive and generic-frequency algorithms as a function of the resource units used during the download process. The horizontal axis is the amount of resources used, and the vertical axis is the coverage. As it is evident from the graph, the adaptive policy makes more efficient use of the available resources, as it is able to download more articles than the generic-frequency, using the same amount of resource units. However, the difference in coverage is less dramatic in this case, compared to the graph of Figure 9. The smaller difference is due to the fact that under the current cost metric, the download cost of documents constitutes a significant portion of the cost. Therefore, when both policies downloaded the same number of documents, the saving of the adaptive policy is not as dramatic as before. That is, the savings in the query cost and the result index download cost is only a relatively small portion of the overall cost. Still, we observe noticeable savings from the adaptive policy. At the total cost of 8000, for example, the coverage of the adaptive policy is roughly 0.5 while the coverage of the frequency policy is only 0.3.

5. RELATED WORK

In a recent study, Raghavan and Garcia-Molina [29] present an architectural model for a Hidden Web crawler. The main focus of this work is to *learn* Hidden-Web query interfaces, not to generate queries automatically. The potential queries are either provided manually by users or collected from the query interfaces. In contrast, our main focus is to generate queries automatically without any human intervention.

The idea of automatically issuing queries to a database and examining the results has been previously used in different contexts. For example, in [10, 11], Callan and Connel try to acquire an *accurate language model* by collecting a uniform random sample from the database. In [22] Lawrence and Giles issue random queries to a number of Web Search Engines in order to estimate the *fraction* of the Web that has been indexed by each of them. In a similar fashion, Bharat and Broder [8] issue random queries to a set of Search Engines in order to estimate the *relative size and overlap* of their indexes. In [6], Barbosa and Freire experimentally evaluate methods for building multi-keyword queries that can return a large fraction of a document collection. Our work differs from the previous studies in two ways. First, it provides a theoretical framework for analyzing the process of generating queries for a database and examining the results, which can help us better understand the effectiveness of the methods presented in the previous work. Second, we apply our framework to the problem of Hidden Web crawling and demonstrate the efficiency of our algorithms.

Cope et al. [15] propose a method to automatically detect whether a particular Web page contains a search form. This work is complementary to ours; once we detect search interfaces on the Web using the method in [15], we may use our proposed algorithms to download pages automatically from those Web sites.

Reference [4] reports methods to estimate what fraction of a text database can be eventually acquired by issuing queries to the database. In [3] the authors study query-based techniques that can extract *relational data* from large text databases. Again, these works study orthogonal issues and are complementary to our work.

In order to make documents in multiple textual databases searchable at a central place, a number of "harvesting" approaches have been proposed (e.g., OAI [21], DP9 [24]). These approaches essentially assume cooperative document databases that willingly share some of their metadata and/or documents to help a third-party search engine to index the documents. Our approach assumes uncooperative databases that do not share their data publicly and whose documents are accessible only through search interfaces.

There exists a large body of work studying how to identify the most relevant database given a user query [20, 19, 14, 23, 18]. This body of work is often referred to as *meta-searching* or *database selection* problem over the Hidden Web. For example, [19] suggests the use of *focused probing* to classify databases into a topical category, so that given a query, a relevant database can be selected based on its topical category. Our vision is different from this body of work in that we intend to *download* and *index* the Hidden pages at a central location in advance, so that users can access all the information at their convenience from one single location.

6. CONCLUSION AND FUTURE WORK

Traditional crawlers normally follow links on the Web to discover and download pages. Therefore they cannot get to the Hidden Web pages which are only accessible through query interfaces. In this paper, we studied how we can build a Hidden Web crawler that can automatically query a Hidden Web site and download pages from it. We proposed three different query generation policies for the Hidden Web: a policy that picks queries at *random* from a list of keywords, a policy that picks queries based on their *frequency* in a generic text collection, and a policy which *adaptively* picks a good query based on the content of the pages downloaded from the Hidden Web site. Experimental evaluation on 4 real Hidden Web sites shows that our policies have a great potential. In particular, in certain cases the *adaptive* policy can download more than 90% of a Hidden Web site after issuing approximately 100 queries. Given these results, we believe that our work provides a potential mechanism to improve the search-engine coverage of the Web and the user experience of Web search.

6.1 Future Work

We briefly discuss some future-research avenues.

Multi-attribute Databases We are currently investigating how to extend our ideas to *structured multi-attribute* databases. While generating queries for multi-attribute databases is clearly a more difficult problem, we may exploit the following observation to address this problem: When a site supports multi-attribute queries, the site often returns pages that contain values for each of the query attributes. For example, when an online bookstore supports queries on `title`, `author` and `isbn`, the pages returned from a query typically contain the title, author and ISBN of corresponding books. Thus, if we can analyze the returned pages and extract the values for each field (e.g, `title = ‘Harry Potter’`, `author = ‘J.K. Rowling’`, etc), we can apply the same idea that we used for the textual database: estimate the frequency of each attribute value and pick the most promising one. The main challenge is to automatically segment the returned pages so that we can identify the sections of the pages that present the values corresponding to each attribute. Since many Web sites follow limited formatting styles in presenting multiple attributes — for example, most book titles are preceded by the label “Title:” — we believe we may learn page-segmentation rules automatically from a small set of training examples.

Other Practical Issues In addition to the automatic query generation problem, there are many practical issues to be addressed to build a fully automatic Hidden-Web crawler. For example, in this paper we assumed that the crawler already knows all query interfaces for Hidden-Web sites. But how can the crawler discover the query interfaces? The method proposed in [15] may be a good starting point. In addition, some Hidden-Web sites return their results in batches of, say, 20 pages, so the user has to click on a “next” button in order to see more results. In this case, a fully automatic Hidden-Web crawler should know that the first result index page contains only a partial result and “press” the next button automatically. Finally, some Hidden Web sites may contain an infinite number of Hidden Web pages which do not contribute much significant content (e.g. a calendar with links for every day). In this case the Hidden-Web crawler should be able to detect that the site does not have much more new content and stop downloading pages from the site. Page similarity detection algorithms may be useful for this purpose [9, 13].

7. REFERENCES

[1] Lexisnexis `http://www.lexisnexis.com`.

[2] The Open Directory Project, `http://www.dmoz.org`.

[3] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *ICDE*, 2003.

[4] E. Agichtein, P. Ipeirotis, and L. Gravano. Modeling query-based access to text databases. In *WebDB*, 2003.

[5] Article on New York Times. Old Search Engine, the Library, Tries to Fit Into a Google World. Available at: `http://www.nytimes.com/2004/06/21/technology/21LIBR.html`, June 2004.

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

[7] M. K. Bergman. The deep web: Surfacing hidden value,`http://www.press.umich.edu/jep/07-01/bergman.html`.

[8] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In *WWW*, 1998.

[9] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. Syntactic clustering of the web. In *WWW*, 1997.

[10] J. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *SIGMOD*, 1999.

[11] J. P. Callan and M. E. Connell. Query-based sampling of text databases. *Information Systems*, 19(2):97–130, 2001.

[12] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the web: Observations and implications. Technical report, UIUC.

[13] J. Cho, N. Shivakumar, and H. Garcia-Molina. Finding replicated web collections. In *SIGMOD*, 2000.

[14] W. Cohen and Y. Singer. Learning to query the web. In *AAAI Workshop on Internet-Based Information Systems*, 1996.

[15] J. Cope, N. Craswell, and D. Hawking. Automated discovery of search interfaces on the web. In *14th Australasian conference on Database technologies*, 2003.

[16] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms, 2nd Edition*. MIT Press/McGraw Hill, 2001.

[17] D. Florescu, A. Y. Levy, and A. O. Mendelzon. Database techniques for the world-wide web: A survey. *SIGMOD Record*, 27(3):59–74, 1998.

[18] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, 2003.

[19] P. Ipeirotis and L. Gravano. Distributed search over the hidden web: Hierarchical database sampling and selection. In *VLDB*, 2002.

[20] P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden web databases. In *SIGMOD*, 2001.

[21] C. Lagoze and H. V. Sompel. The Open Archives Initiative: Building a low-barrier interoperability framework In *JCDL*, 2001.

[22] S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360):98—100, 1998.

[23] V. Z. Liu, J. C. Richard C. Luo and, and W. W. Chu. Dpro: A probabilistic approach for hidden web database selection using dynamic probing. In *ICDE*, 2004.

[24] X. Liu, K. Maly, M. Zubair and M. L. Nelson. *DP9-An OAI Gateway Service for Web Crawlers*. In *JCDL*, 2002.

[25] B. B. Mandelbrot. *Fractal Geometry of Nature*. W. H. Freeman & Co.

[26] A. Ntoulas, J. Cho, and C. Olston. What's new on the web? the evolution of the web from a search engine perspective. In *WWW*, 2004.

[27] A. Ntoulas, P. Zerfos, and J. Cho. Downloading hidden web content. Technical report, UCLA, 2004.

[28] S. Olsen. Does search engine's power threaten web's independence? `http://news.com.com/2009-1023-963618.html`.

[29] S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, 2001.

[30] G. K. Zipf. *Human Behavior and the Principle of Least-Effort*. Addison-Wesley, Cambridge, MA, 1949.