Challenges in Crawling the Deep Web

Jianguo Lu[1], Yan Wang[2]

[1] School of Computer Science, University of Windsor, Canada
jlu@uwindsor.ca
[2] School of Information, Central University of Finance and Economics, China
dayanking@gmail.com

1 Introduction

The deep web is considered full of rich content that is much bigger than the surface web [1]. Nowadays almost every web site comes with a search box, and many of them such as twitter.com provide in addition a programmable web API. First of all it would be nice if those deep web documents were search engine visible. Not surprisingly general search engines such as Google [4] [9] and Bing [10] try to index some of these un-crawled territory. In addition, numerous applications want to tap into the rich deposit of data to build distributed search engines [11], data integration applications, vertical portals [12] etc. While the deep web data providers are happy to serve the data to ordinary users and even application programs, they may not want to be overloaded with automated crawlers whose target is to index or even worse to download the entire database to set up their own operation. The downloading problem is exacerbated when the data is served from the cloud by a web service that enables automated access to the data. In addition, the downloading may not be done once for all. Quite often a data source is downloaded or scanned many times to keep the freshness of the data [12]. Although data providers can thwart robot visiting by judging from its visiting pattern, the monitoring and detection of such downloading is by no means an easy task [13].

Thus an intriguing question that is of interest to both deep web data providers and crawlers is how difficult it is to harvest most of the data records inside a deep web data source by sending appropriate queries. In fact, the selection of the queries sits in the key role to answer the question. For a textual deep web data source, a primitive method of query selection for a crawler could be randomly selecting query from a dictionary. However this method is not efficient because a large number of rare queries may not match any document or there could be many overlapping returns which occupies too much network traffic. It is proved that, without an elaborate query selection method, only random queries (even no empty returns) will cause approximate 9 times repeated retrieval when 90% unique documents of the data source are covered. Thus, some well-crafted query selection methods are designed by researchers [4,16,17,31,18,32,33,34] to significantly reduce the redundant returns and save network transmission. On the other side, facing the crawlers with state-of-the-art query selection methods, the data providers further increase the difficulty of retrieving data by setting a return limit, i.e., only part of matched document by a query can be returned. It is shown that the return limit plus ranking support can extremely exacerbate the efficiency of crawlers in later section of this chapter. To our knowledge, so far, there is no satisfying query selection method to dealing with the problem.

In this chapter, our study is the query selection for textual data sources, ignoring much of the engineering challenges such as extracting meaningful data from HTML pages. We more focus on the analysis of the difficulty of crawling textual deep web data sources from the viewpoint of query selection, i.e., whether it is possible to reach the bottom of the deep web, if not, how deep we can reach in a deep web data source. Meanwhile, the related work are introduced, i.e., how existing research work addressed this difficulty.

The rest of the chapter is arranged as follows: in section 2, a graph model is presented to describe the crawling process for textual deep web data sources; in section 3, with the graph model, the query selection problem is introduced and formally defined; in section 4, categorizing deep web data sources into four models according to the difficulty of crawling; in section 5 and 6, the hardness analysis of crawling and related work are demonstrated according to the category; in section 7, more discussions and some conclusions are presented.

2 Textual deep web crawling

The crawling of the surface web has been well studied since the advent of the Web, and has become a mature technique used in industry. The deep web crawling, given its similar goal and similar crawling process, seems to be a trivial problem which can be solved by borrowing what we have learnt in the surface web crawling. Why is the deep web crawling an issue in the first place?

The deep web can be modelled as a bipartite graph $G = (D, Q, E)$, where nodes are divided into two separate sets D and Q, i.e., the set of documents D and the set of queries Q where $|D| = m$ and $|Q| = n$. Every edge in E links a query and a document. There is an edge between a query and a document if the query occurs in the document.

The deep web crawling problem is to find the queries so that they can cover all the documents. If we regard queries as URLs in surface web pages, the deep web crawling process is very much the same as the surface web crawling: starting from some seed URLs (terms), we obtain the web pages. From those web pages, more URLs(terms) are collected and subsequently issued to the web server again. This process repeats until all the pages are traversed. Figure 1 gives an example of a deep web that is represented as a bigraph, where $D = \{d_1, d_2, \ldots, d_9\}$ and $Q = \{q_1, q_2, \ldots, q_7\}$. The brute-force crawling process can be illustrated as follows. Suppose that the seed query is q_1. After q_1 is sent, document d_2 is retrieved and more queries q_3 and q_6 are found from the document. We can put these newly found queries in a queue and send these queries in sequel. The process is repeated until all the documents are found.

The algorithm works fine for the surface web but not the deep web due to the following key difference: in the surface web crawling each URL will return

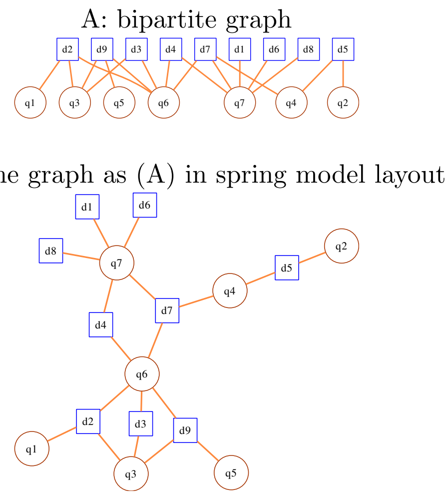

Fig. 1: Deep web as a bipartite graph

only one page while in deep web crawling each term will return multiple pages. A surface web page has an unique identifier, i.e., the URL, while a deep web document has many terms to access it. Because of the multiplicity of the returns it is inevitable that many of the returns are redundant, and there is no way to exclude these duplicates locally before retrieving them from the server. Such redundancy is costly because it occupies network communications.

Essentially the brute force method needs to send out all the queries that can be found. Suppose the graph is connected and we can find all the queries, the average cost per document retrieved is

$$\langle d \rangle = \frac{v}{m}.$$

When the queries are single terms, the cost is the average length of the documents. More formally, we can define the volume of a graph as follows.

Definition 1 *(Volume) Suppose there are m number of documents and n number of queries. Let d_i denote the degree of the document node i , for $i \in \{1, 2, \ldots, m\}$, and f_j the degree (or document frequency) of the query node j for $j \in \{1, 2, \ldots, n\}$. The volume v of the documents D is*

$$v = \sum_{i=1}^{m} d_i = \sum_{j=1}^{n} f_j.$$

The mean degree of D is

$$\langle d \rangle = v/m.$$

If the brute force crawling policy is used, all the queries will be issued and the total cost is the volume of the graph. The redundancy is huge in general due to the fact that documents can have many queries, thus they can be retrieved many times. If a query can consist of multiple terms, the number of queries in a document is significantly large than the number of terms inside a document. The brute force algorithm will induce roughly $\langle d \rangle$ overlapping rate, i.e., on average each document is retrieved $\langle d \rangle$ number of times, where $\langle d \rangle$ is the average degree of the documents. This redundancy is the source of the problems in deep web crawling.

3 Query selection problem

Since sending all the queries is not an option, we need to select a subset of the queries $Q_s \subseteq Q$. When queries in Q_s are sent and the matched documents are retrieved, a subgraph, denoted by (D_S, Q_S, E_S), of (D, Q, E) is formed, where $D_S \subseteq D$ is the set of the retrieved documents, $E_S \subseteq E$ is the set of edges that connects the queries and documents. Let $n_s = |Q_s|$, $m_s = |D_s|$, $v_s = |E_s|$. The crawling cost is

$$\langle d_S \rangle = \frac{v_S}{m_S} \tag{1}$$

The *query selection problem* is to find a Q_s such that $\langle d_S \rangle$ is minimal while satisfying the constraint $m_s = m$, i.e., all the documents are covered by the selected queries.

The performance of a crawler, for both the surface web and the deep web, is normally measured as the coverage or weighted coverage over the cost to reach that coverage [29] [16] [8]. In the deep web, the cost can be measured in terms of a few factors, such as the number of queries sent (i.e., n) [30], the accumulative total number of documents retrieved (i.e., v), or some combination of both based on some weighting scheme[16]. Most of the research on the deep web crawling focusing on minimizing the number of queries to be sent, in the belief that the network traffic of sending the queries constitutes the major cost of the web crawling.

In reality it is the *query sessions*, not the number of queries, that determines the crawling cost. Many data sources paginate the results and return only a small number of matches, say p of them, for each query. If documents beyond the first p results are required, the same query needs to be sent again. For instance, in Bing web service, if a query q has 200 matches, they will be paginated into 20 sections, each query session with keyword q can only obtain one of the sections. In order to obtain all the 200 matches, for the same query 20 query sessions are required. So the number of query sessions grows linearly with the number of documents retrieved.

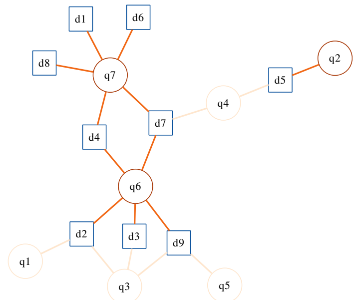

Fig. 2: A solution of the deep web crawling. The darker nodes and edges are in the solution graph. The lighter ones are for illustration purpose and they are not part of the graph.

Example 1 (subgraph) *Figure 2 shows a solution* $Q_s = \{q_2, q_6, q_7\}$ *and the corresponding subgraph.*

$$\langle d_S \rangle = \frac{v_s}{m_s} = \frac{1+5+5}{9} = \frac{11}{9}.$$

The other solution $Q'_s = \{q_2, q_3, q_5, q_7\}$ *has a lower average degree:*

$$\langle d_S \rangle = \frac{v_s}{m_s} == \frac{1+3+1+5}{9} = \frac{10}{9}.$$

Although Q'_s has more queries, it is considered a better one because of its low redundancy. Based on this observation it is more accurate to measure the crawling cost in terms of v, the total number of documents retrieved, instead of the queries sent. In order to discuss the cost independent of the data source size or *return limit* (i.e., only part of matched documents can be returned), we measure the cost by the ratio between the accumulative total number of documents (v) that have been crawled and the distinct documents among them (m), i.e.,

$$\langle d_S \rangle = \frac{v_S}{m_S} \tag{2}$$

Intuitively, we want to measure that for each new document retrieved, on average how many redundant URLs have to be downloaded over the internet.

4 Models of textual deep web data sources

"The science and practice of deep web crawling is in its infancy" [29]. Crawling a deep web data source is a challenging task, even for textual content with one input field. There is not a panacea that works for all kinds of data sources– different kind of data sources behaves differently and needs its own way of crawling. According to the crawling difficulty, here we categorize deep web data sources into four different models by means of the concepts of *the probability of document being captured* and *document ranking*

First of all, the probability of a document being captured is defined as follows:

Definition 2 *(Probability of document being captured) Given a bipartite graph $G = (D, Q, E)$ derived from a data source containing m documents, let p_i ($1 \leq i \leq m$) denote the probability of the document node i being captured by random edges from E, then we have*

$$p_i = \frac{d_i}{v},$$

$$\sum_{i=1}^{m} p_i = 1.$$

Since each edge can be considered as the action of issuing the corresponding query, the concept of the probability of document being captured is used to measure how easy a document can be returned by queries in Q.

In many data sources, for a query, all matched document will be returned according to a ranking, in which each document has a rank number. A matched document with a smaller rank number will be returned earlier than the one with a bigger rank number. The ranking is called document ranking and data sources with document ranking are called *ranked data sources*.

With the two concepts, we can classify data sources into several models by the crawling difficulty as follows:

Model M_0 Every document has the same probability of being captured (e.g., $\forall i, j$, $p_i = p_j$), and all the matched documents are returned;
Model M_h Heterogeneous data source where documents have varying capturing probabilities (e.g., $\exists p_i \neq p_j$, $i \neq j$);
Model M_r Ranked data source that has a return limit;
Model M_{hr} Heterogeneous and ranked data sources.

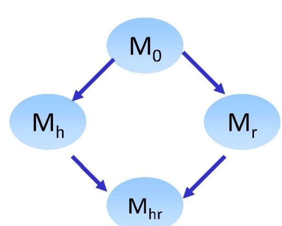

Fig. 3: Models of data sources.

Our category is based on such an intuition i.e., the difficulty of crawling a data source is not much if its document can be easily returned by random queries from

Q. The relationship between these models can be depicted by Figure 3, where $M_x \to M_y$ indicates that M_y is harder to crawl than M_x. To be more precise, in order to reach the same coverage by using random queries, M_y will require higher cost than M_x. The figure shows that models M_h and M_r are harder to crawl then M_0, while M_h and M_r are two orthogonal dimensions that add the complexity to the problem. Model M_{hr} inherits the complexity from both M_h and M_r. Therefore it is the most challenging case in our classification.

Obviously, heterogeneous data or ranking matched documents plus return limit can significantly increase the difficulty to retrieving documents from a target data source. For heterogeneous data, from the definition of p_i, we can see that long documents (that contain more terms) has higher probability being captured than the short documents. It means that long documents are easier to be retrieved repeatedly and it will cause higher cost. For the ranks of matched documents plus the return limit, they could make only documents with small rank numbers returned, which could lead to a very low coverage and a very high cost. In the later section, we will discuss how these factors have an effect on crawling in details.

Although the assumptions of M_0 can hardly occur in real applications, it is the starting point for understanding other models as illustrated in Figure 3. In addition, the result of M_0 also serves as a lower bound for crawling cost when random queries are used. Model M_h is rather common in data sources, because there are unranked data sources, and more importantly, because many data sources are not very large. When a data source is of moderate size, many queries match less than k documents. Therefore the data source will return most of the matches even though the matches are ranked.

5 Query selection on Model M_0 and M_h

In the Web, most of deep web data sources belong to M_h models, which has no ranking on matched documents and the return limit (in fact, maybe there is a document ranking but the data source still belongs to M_h model if all matched documents are returned). This is also the reason that many research work [6,30,4,16,17,31,18,32,33,34] about query selection focuses on data sources in M_h. In this section, we first discuss the difficulty of crawling M_h data sources from analysing the case of M_0, then some previous related work are introduced.

5.1 Hardness analysis

There is a diminishing return when crawling continues. Typically the relationship between the harvest and the cost can be depicted in a shape like Figure 4, albeit the exact diminishing speed is hardly studied in theory. On the side of empirical study, different diminishing speeds are reported in various experiments [16]. We find that the diminishing speed is in fact solely dependent on the stage of the crawling. More precisely, we give the following theorem:

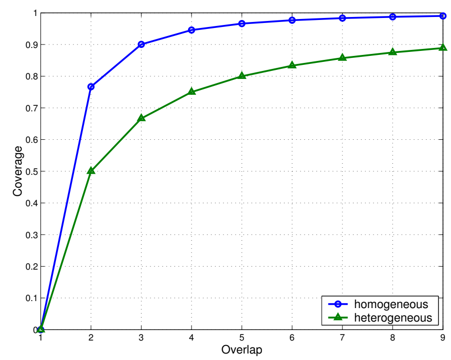

Fig. 4: Coverage P as a function of overlapping, i.e., the average degree $\langle d \rangle$ with respect to the query set. Two curves are drawn from Equations 7 and 8, depicting the homogeneous documents M_0 and heterogeneous documents M_h respectively.

Theorem 1 *Let P denote the percentage of the data that has been harvested, and $\langle d \rangle$ the average degree of the collected documents with respect to the random queries. Then*

$$\langle d \rangle \approx \frac{-ln(1-P)}{P}. \tag{3}$$

when $f_j << m$ and the documents are homogeneous, i.e., they are of the same probability of being matched. The equation becomes exact when $m \to +\infty$.

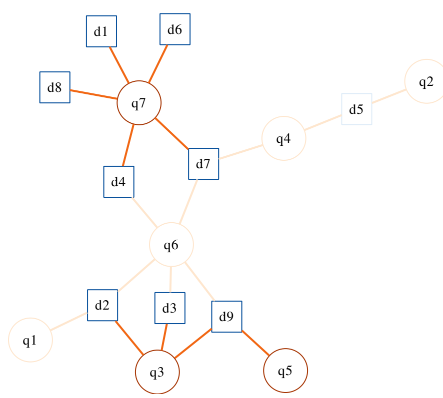

Fig. 5: Random graph when queries are selected randomly.

Equation 3 reveals that the percentage of the documents retrieved is solely dependent on the average degree of the graph, or the overlapping rate. We use Figure 5 to explain the relationship between $\langle d \rangle$ and P. The figure is created when three queries are randomly selected. In this graph the darker edges and nodes are the ones selected. They constitute the graph under construction while the lighter edges and nodes are drawn so that we can view what is the original graph. In this graph 8 documents are covered, among them d_9 is covered twice. Hence the mean degree $d = 9/8$. Since the original graph has 9 document nodes, the percentage of the documents being covered is $P = 8/9$. Note that this small example is for illustration purpose only–the data is too small to coincide with Equation 3.

Derivation Let us first consider the most simple scenario when every document has the same probability of being connected to an edge. At the outset, there are m document nodes and zero edge. When query j is issued, or equivalently, f_j number of edges are added randomly to the graph, the probability of a document i not being covered by any edge is $(1 - 1/m)^{f_j}$. Without loss of generality, let us assume that the return of a query is small relative to the corpus size, i.e., $f_j \ll N$. In this case the sampling with replacement can be approximated by the sampling with replacement. After n number of queries are fired, v number of random edges are added to the graph. The probability of document i is still isolated is:

$$S = \left(1 - \frac{1}{m}\right)^v \approx e^{-\frac{v}{m}} \tag{4}$$

Note that Equation 4 becomes exact when $m \to +\infty$. S can be also interpreted as the fraction of the nodes that are isolated in the graph. Let u denote the number of documents that are covered by some queries. u/m is the fraction of the documents that are already captured, i.e., $P = u/m = 1 - S$. Rearrange Equation 4 we obtain:

$$ln(1 - P) = -\frac{v}{m} = -\frac{v}{u}\frac{u}{m} = -\langle d \rangle P$$

Hence

$$\langle d \rangle = -\frac{ln(1 - P)}{P}$$

Theorem 1 is useful to estimate the cost of downloading. More often we need to use the inverse of the equation to estimate the fraction of the data harvested based on the cost. Unfortunately Equation 3 does not have a simple analytical solution for P. Although Lambert-W function can be used to approximate the solution when P is large, for smaller P there is a big discrepancy even when the expansion contains hundreds of terms. Hence there is a need to find an approximation for P.

In real applications queries have varying number of matches. Given a sequence of queries $(q_1, q_2, ..., q_k)$, whose document frequencies are $(m_1, m_2, \ldots, m_k)$. For query q_j, a document i that is not matched by the query is $1 - m_j/N$. For now we are assuming that every document has equal probability of being matched. Let P(i) denote the probability of a document i that is captured after these k

number of queries, u the number of documents retrieved, P the fraction of the documents retrieved. $u = PN$. Then

$$
\begin{aligned}
P = & \sum_{i=1}^{N} P_i/N = P_i \\
= & 1 - \prod_{j=1}^{k}(1 - \frac{m_j}{N}) \\
\approx & 1 - e^{\frac{\sum_{j=1}^{k} m_j}{N}} \\
= & 1 - e^{\frac{\sum_{j=1}^{k} m_j}{u} \frac{u}{N}} \\
= & 1 - e^{OR \times P}
\end{aligned} \tag{5}
$$

Although it is derived from a simplified document graph where only dis legomena (terms that occur only twice) are used, the result remains the same for other queries as long as the query cardinality is much smaller than N. In the derivation above, 2 is the cardinality of the query and it can be replaced by any other number, and result remains true. Hence we have the following

Apply Taylor expansion we have

$$\langle d \rangle = -\frac{ln(1 - P)}{P} = 1 + P/2 + P^2/3 + ... \tag{6}$$

A more accurate approximation of Equation 3 is obtained empirically by running regression on the data generated from Equation 3:

$$P \approx 1 - \langle d \rangle^{-2.1} \tag{7}$$

Implications An observation we can make based on Equation 6 is that when P is very small, $\sum_{i=2}^{\infty} P^i/(i+1)$ is neglectable compared with $P/2$. Hence P increases almost linearly with $\langle d \rangle$. With the increase of P, the cost increases at a faster speed. If we are harvesting as much data as possible from many data sources, instead of exhaustively siphoning all the data records from one single data source, Equation 3 gives a guideline as for when it is the good time to jump to another data source for a fixed crawling resource.

Since $\langle d \rangle$ can be calculated easily from the crawling history, Equation 7 is particularly useful to estimate how much data have been downloaded and when the crawling process shall stop. Another surprising observation we can make is that large queries induce the same overlapping rate as small queries, since the document frequencies f_j of the queries do not occur in the equation. While it is true that large queries will save the number of queries, it will not save the duplicates retrieved.

From Figure 5 it can be seen that $\langle d \rangle$ is rather small to capture a high percentage of the documents. As a rule of thumb, in order to harvest 50 % of the documents, the overlapping rate is only about 1.3. If on average each page is accessed three times, you can deduce that those accessed pages constitute 90% of the total population. This result is rather disturbing for data providers– it seems that people can download most of the data with ease.

Fortunately, the assumption for Theorem 1, i.e., all the documents can be obtained randomly with uniform distribution, does not hold in general. In real deep web data sources, web pages are of different sizes, causing them of different probabilities of being captured by queries. One opinion regarding the size of documents on the web is that it follows a power law with exponent around two, with the minimal value around 1k. For such data, it is shown that [35]

$$P = 1 - \langle d \rangle^{-1} \tag{8}$$

The comparison between Equations 8 and 7 is illustrated in Figure 4. Roughly speaking, to obtain 90% of the data the average degree is around 9 in the case of heterogeneous data, and 3 for homogeneous data. What impacting downloading much more is the return limit as we will explain in Section 6.

5.2 Existing methods

So far, most of research work about query selection in deep web crawling concentrate on data sources in M_h [6,30,4,16,17,31,18,33,34] since not only the data sources in M_h model are ubiquitous in the Web but also the cost of random queries can be significantly improved. In general, these research work can be divided into two kinds. One is incremental method and the other is sampling-based method.

Ntoulas et al. [16] proposed an incremental adaptive method for crawling textual data sources. An incremental method selects queries from the downloaded documents and the number of documents increases as more queries are sent. Their method first model the query selection problem as set-covering problem [36],which is NP hard problem. In such model, the data source is represented by a binary document-term matrix, each column and each row stand for a term and a document respectively. Each entry is "1" iff the corresponding document contains the corresponding term, otherwise, it is zero. Finally, the query selection problem is to select a set of columns that can cover all rows with minimal cardinalities.

The method in [16] iteratively selects the query returning most new documents per unit cost and it is represented by the formula $\frac{N_{new}(q_j)}{Cost(q_j)}$. For each query, its cost consists of the costs for sending q_j, retrieving the hyperlinks of the matched documents, and downloading them.

Since there is no prior knowledge of all the actual document frequencies of the queries in the original data source DB, this method requires the estimations of the actual document frequencies based on the documents already downloaded. With the estimated document frequencies of all queries in the downloaded documents, the number of matched new documents of each query will be calculated.

Madhavan et al. [4] developed another incremental method for textual data sources. Because the system is an industry product, it needs to consider how to select seed queries. Since they need to process difference languages, their approach does not select queries from a dictionary. Instead, Their system detects the feature of the HTML query form and selects the seed queries from it. After that, the iterative probing and the query selection approach are similar to those, that proposed in [16].

Their query selection policy is based on TF-IDF that is the popular measure in the information retrieval, which measures the importance of a word. Madhavan et al.'s method adds the top 25 words of every web page sorted by their TF-IDF values into the query pool. From the query pool, they remove the following two kinds of terms: 1) eliminate the high frequency terms, such as the terms that have appeared in many web pages (e.g., $> 80\%$), since these terms could be from menus or advertisements; 2) delete the terms which occur only in one page, since many of these terms are meaningless words that are not from the contents of the web pages, such as nonsensical or idiosyncratic words that could not be indexed by the search engine. The remaining words are issued to the target deep web data source as queries and a new set of web pages are downloaded. Then this is repeated again in the new iteration.

Currently, the authors of [18,34] presented another two incremental methods. Different from the previous incremental methods, they first modelled the query selection problem as reinforcement learning problem. In this model, a crawler and a target data source are considered as an agent and the environment respectively. Then its selection strategy will be dynamically adjusted by learning previous querying results and takes account of at most two-step long reward.

For sampling-based methods, Barbosa et al. [30] first proposed one that siphons the deep web by selecting queries with highest frequencies from the sample document collection. Unlike the incremental method, a sampling-based crawling method selects queries from a fixed or near fixed sample which is usually derived from the first batch of downloaded documents.

This method selects the highest frequency queries from the term list, which are expected to lead a high coverage. It is composed of two phases: phase 1 selects a set of terms from the HTML search form and randomly issues them to the target deep web data source until at least a non-empty result page is returned. By extracting high frequency terms from the result pages, their algorithm creates a term list. Then it iteratively updates the frequencies of the term and adds new high frequency terms into the list by randomly issuing the term in the list until the number of submissions reaches the threshold. In phase 2, the method uses a greedy strategy to construct a Boolean query to reach the highest coverage, it iteratively selects the term with the highest frequency from the term list, and adds it to a disjunctive query if it leads to an increase in coverage. For example, if 10 terms ($q_1, \ldots, q_{10}$) are selected, the final issued disjunctive query is $q_1 \vee \ldots \vee q_{10}$.

In [17,33,31], the authors also presented sampling-based query selection methods. In [17], the authors also modelled the query selection as set covering problem as [16]. The difference from [16] is that all queries are directly generated from a document sample of the target data source once for all. A greedy algorithm is used to select queries, which only need to run on a small document sample and a small set of terms that occur the documents of the sample. In [31], the authors followed the query selection framework in [17], their main contribution is to make use of the distributions of document sizes and term frequencies to optimize the quality of queries from the sample and thus improve the mapping results in the target data source.

6 Query selection on Model M_r and M_{hr}

In the late decade, more and more data sources rank the matched pages and return only top k documents for queries, which certainly belongs to M_{hr} since they contains since they contain various lengths of documents. Thus, an effective and efficient query selection method for crawling them is what the industry needs. However, to our knowledge, there is no satisfying query selection method to dealing with the data sources in M_{hr} due to the difficulty. As the previous section, we give the hardness analysis first and then some related work are demonstrated.

6.1 Hardness analysis

Many data sources rank matched documents and return only top k documents for queries. When the data source is large, most of the queries will match more than k documents, yet the returns are still limited by k. This will make the query selection almost an impossible task—large queries whose cardinality are larger than k can only trawl the top layer of data, while small queries (queries whose cardinalities are smaller than k) are hard to obtain. More importantly, using small queries may cause the document-query graph disconnected.

Large queries A naive way to solve the query selection problem could be using large queries such as popular words or disjunctions of multiple words. In literature large queries are often preferred. For instance, [30] selects large queries directly, [16] uses set covering method that will lead to the selection of large queries due to their cost model, and [4] uses tf-idf to select queries, where term frequency plays a major role in query selection.

This line of work is based on the assumption that all the matched documents are returned to the user. In reality, most data providers rank the matched documents and only return the top k documents, where k typically ranges between 10 to 1000. For instance Google only returns at most 1000 documents, even if the query matches with much more pages. We shall not confuse *return limit* with paginated *query sessions*. In the case of Google search, a query can match with one million documents, among them only top 1000 can be returned. This 1000 is the return limit. They can be returned but these 1000 documents are not returned all at once within one query session. Instead, the first *query session* will return 10 matches.

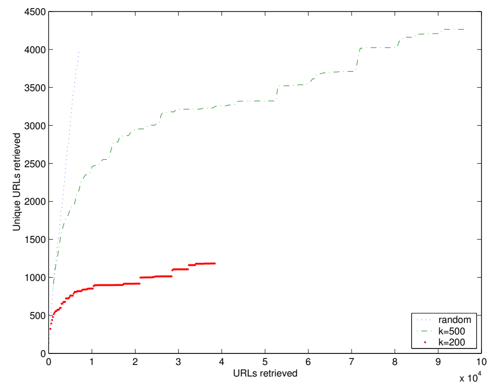

Fig. 6: In ranked data sources, large queries can only retrieve a portion of the data.

Note that with this limitation, only top k/f_{min} percent of the documents can be harvested [38], where f_{min} is the cardinality of the smallest query. That is to say, the larger the query, the smaller percentage of the documents you can harvest. More formally, we have

Theorem 2 *Given a statically ranked data source whose return limit is k. If every query returns more than f_{min} number of documents, then the maximal percentage of the documents can be retrieved is*

$$\frac{k}{f_{min}} \tag{9}$$

The static ranking can be the page ranking, or rankings based on the time stamp, file size, or alpha-numeric ordering. Although many data sources, especially advanced search engines, use a combination of static ranking (such as page ranking) and relevance ranking, many simple deep web sites use static ranking only. Therefore it is worthwhile to study the crawling problem for such ranking policy.

This counter-intuitive phenomenon can be illustrated using Figure 6, which shows the growth of the coverage over the number of documents retrieved for the classical 20 news groups corpus (NG20 hereafter)[3] that contains 19997 documents. The queries are all the 190 popular words from Webster dictionary with document frequency ranging between 1000 and 5000. More popular words are also tried and they result in even lower coverage. Two return limits, 200 and 500, are experimented. In the case of $k = 200$, in total 38,000 document are retrieved, among them only 1183 are distinct ones. In the case of $k = 500$, overall 95,000 documents are retrieved and among them 4265 are unique.

This experiment shows that even for such a small corpus, the harvest stagnates after a small portion of the documents are retrieved. Notice that in both

[3] Available at http://qwone.com/ jason/20Newsgroups/

cases $\langle d \rangle$, the ratio between the total retrieval and the unique documents are very large, indicating that most queries in later stage return only redundant documents. For comparison purpose, we also plotted the coverage for queries randomly selected from Webster, where k=200. Surprisingly, random queries perform much better than popular words due to the large amount of small queries whose document frequencies are within the value of k.

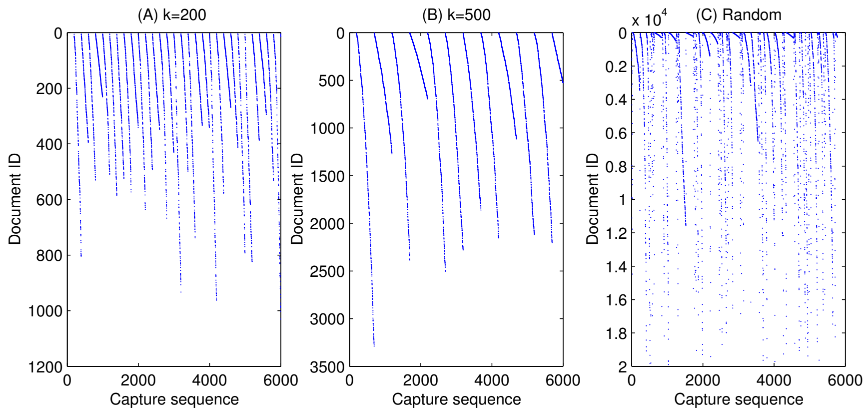

Fig. 7: Large queries can not reach the bottom of the deep web. In total 6000 documents are captured, including duplicates. Panel A: k=200, $df = 1000 \sim 5000$. Panel B: k=500, $df = 1000 \sim 5000$. Panel C: k=200, df is unrestricted.

Table 1 and Figure 7 show that the range of documents retrieved are inversely proportional to the size of the query (f_j). For instance, in the case of $k = 500$, for the first query *understand*, $f_1 = 1002$, and it covers the documents in the range of 1 and 3300. For the second query *said* which can match 2295 documents (more than twice of the first query), the range of covered documents is between 1 and 1300 (less than half of the first query).

Small queries Since large queries are not effective in crawling ranked data sources, small queries (queries whose cardinalities are smaller than k), seem to be the obvious choice. One may think that by using small queries we can harvest most of the data with ease. However, it is extremely challenging to learn the small queries and their document frequencies. What is worse, the documents may not be covered or connected even if we know aprior all the small queries. By Erdos-Renyi random graph model, a random graph is almost surely connected if the average degree $d > ln(m)$ [39]. In the same vein we have

Theorem 3 *When $\langle d \rangle = log(m)$, the probability of capturing all the documents is close to 1 when $m \to +\infty$.*

Table 1: Queries, their document frequencies(f_j), and new returns when k=200 and k=500

query	matches(f_j)	new (k=200)	new(k=500)
understand	1002	200	500
said	2295	123	353
until	1159	70	259
time	4221	46	135
free	1170	40	196
year	1681	38	157
government	1439	20	99
place	1359	18	91
right	2950	1	29
else	1679	16	84
once	1278	3	56
he	4040	1	9
available	1326	20	153
few	2006	4	19
bit	1289	53	129
system	2118	2	37
lot	1688	21	44
...			

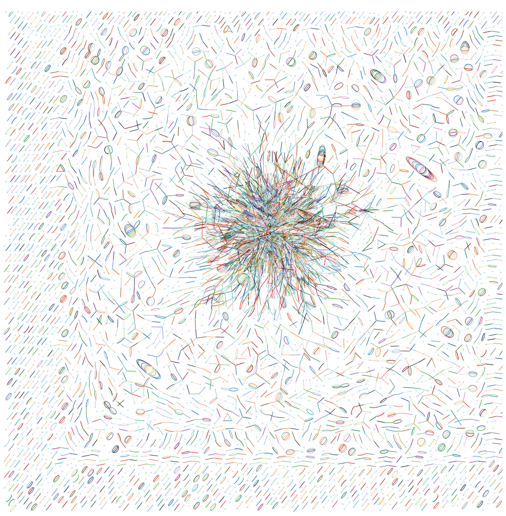

Fig. 8: Document graph constructed from all the dis legomena (terms whose df=2) in corpus NG20. It shows there is a large component and many small isolated data islands.

When small queries only are used, the average degree may not be larger than $log(m)$, hence the crawling task is doomed to fail. Figure 8 shows such a graph constructed from NG20 corpus when *all* the dis legomena are used. There are about half of the documents not covered by any query and not drawn in the graph. For covered documents we can see that they are scattered in many small and isolated data islands. It is challenging to learn these queries and the frequencies of these queries.

With the increase of document frequency, the large component becomes bigger and the islands are absorbed by the large component gradually. Only when very large queries are used can all the documents be retrieved. Since the graph is not connected, we can not reach most of the documents even using the brute force crawling method. If the seed query hits an island, the algorithm stops at the island by retrieving only a few documents. If the seed query hits the large component, we can download all the documents in the large component, leaving all the islands alone.

One misconception is that it is easy to learn most of the (small) queries from a sample set of documents, since there are large number of small queries by Zipf's law. Roughly speaking, supposing the vocabulary size is V, there are about $V/2$ number of hapax legomena and $V/6$ number of dis legomena. Given the sheer size of small queries, one may conclude incorrectly that these queries are good enough to harvest all the documents. However, this is not the case in both theory and practice. According to Heaps' law, the number of terms grows with the size of corpus, although with sub-linear speed. Hence it is impossible to learn the entire vocabulary of a corpus by taking a sample of it. The empirical experiments say that if the sample size is 1%, the fraction of the vocabulary can be learnt is around $10\%(=\sqrt{0.01})$ according to Heaps' law.

6.2 Related work

From the above analysis, we can recognize the difficulty of crawling textual data sources in M_r or M_{hr}. This is why there are no solid research work to address the problem. Thus, we just introduce some close related work here.

In [16], although the authors mentioned that the ranking and the return limit could have a severe effect up on their method, they did not provide a practical method to dealing with them. The authors only presented an ideal ratio-based query selection algorithm to avoid their effect, which assumes a random ranking for matched documents i.e., the matched documents returns randomly.

In [32], Valkanas et al. first propose a *breadth crawling* method for a certain deep web data source, i.e., only retrieving the *top-k* results for all potential queries instead of download all document inside it. They extended the selection strategy in [16] into

$$Efficiency(t) = \frac{(1-\omega) * P_{new}(t) + \omega * RankingGain(t)}{Cost(t)}$$

by inserting the ranking gain value of query, where ω is a weight and P_{new} is the fraction of estimated new documents that query t is expected to retrieve (over over all downloaded documents without the return limit). For each selection iteration, the term with maximum $Gain(t)$ will be sent as the next query. The purpose of this method is to build a small copy of the original data source based on all *top-k* documents by approximating the ranking function employed by it. Intuitively, for a query t, its efficiency of crawling can be accurately calculated if its P_{new} and ranking can be estimated well. The ranking gain $RankingGain(t)$ for each query is used to measure how similar an approximated ranking would be to the actual one, and bigger ranking gain means less accuracy ($RankingGain(t) = 0$ if the approximated ranking is complete same as the actual).

In [40], although the authors discussed the ranking and the return limit, their contribution is not to crawling deep web data sources in M_r or M_{hr}. Instead, they tried to build a TFIDF-based ranking function that ranks matched documents by queries for data sources without ranking support. There are similar literature [41,42].

7 Discussions and Conclusions

In deep web crawling a query returns multiple documents that result in duplicates. Reducing this redundancy is a unique problem in deep web crawling, and the source of the challenges in deep web crawling. The major cost is the network traffic which could be measured by the number of queries for small data sources. For large data sources such as online social networks where most of the queries match a large number of documents, the matched documents are often paginated and returned by multiple query sessions. Therefore the query cost is the number of query sessions, which is proportional to the accumulated number of documents retrieved, or the overlapping rate.

In Model M_0 the crawling cost (the overlapping rate) is a function of the percentage of the data retrieved, independent on the types of queries (e.g., large or small queries) used. For homogeneous data sources (Model M_0) where each document has equal probability of being captured by queries, it is rather easy to harvest most of the data. For heterogeneous data sources (Model M_h), the cost to downloading the data is higher, but still affordable.

In ranked data sources (Model M_r or M_{hr}) only the top k documents are returned, causing that the documents ranked low may never be returned if large queries are used. Therefore to the contrary of common practice small queries, or the queries whose document frequencies are commensurate to k, should be preferred according to our analytical analysis and empirical experiments. Although large queries are effective in the beginning of the crawling process, they have zero contribution to reach high coverage. For instance, if the document frequency of a query is $2k$, then it can only retrieve the top half of the data source. Therefore it has zero contribution to the retrieval of the remaining half data. To retrieve the other half, smaller queries have to be used.

Ranked data sources requires small queries that matches less than k number of documents. Selecting queries of appropriate size is a daunting task because of the well known over-estimation and large variance problem. For instance, if the sample size is $1/k$ of the total database, every query that occurs once in the sample (hapax legomenon) is estimated to have k matches. But in reality its document frequencies can be anywhere between 1 and k. When the total data base is very large, any feasible sample will be much smaller than $1/k$ of the original size. In that case even the terms only occurring once in the sample may have document frequencies much larger than k. That means even the hapax legomena in the sample may return only duplicate documents.

When learning the queries from sample is not feasible, the other alternative is to crawl the documents and collect the new terms in newly retrieved documents. Regardless of the high cost of this approach, it will work only when the graph is connected. Our empirical experiments show that when only small queries are used, the graph is not connected and there are many small isolated islands albeit one large component does exist. This result is supported by the random graph theory.

There are a few implications from our results. For unranked data sources, or small ranked data sources, it is rather easy to download most of the data. For large ranked data sources with a small return limit, it is impossible to download most of the data even with unlimited computing resources. Data providers can design the data source carefully by setting the return limit based on the data source size so that the downloading is infeasible.

References

1. M.K.Bergman: The deepweb: Surfacing hidden value. The Journal of Electronic Publishing **7**(1) (2001)
2. D.Shestakov, S.S.Bhowmick, E.P.Lim: Deque: querying the deep web. Journal of Data Knowledge Engineering **52**(3) (2005) 273–311
3. B.He, M.Patel, Z.Zhang, K.C.Chang: Accessing the deep web: A survey. Communications of the ACM **50**(5) (May 2007) 94–101
4. J.Madhavan, D.Ko, L.Kot, V.Ganapathy, A.Rasmussen, A.Halevy: Google's deepweb crawl. In: Proc. of VLDB. (2008) 1241–1252
5. Ipeirotis, P., Gravano, L., Sahami, M.: Probe, count, and classify: categorizing hidden web databases. In: SIGMOD. (2001) 67–68
6. S.Raghavan, H.G.Molina: Crawling the hidden web. In: Proc. of the 27th international Conference on Very Large Data Bases (VLDB). (2001) 129–138
7. Alvarez, M., Raposo.Raposo, J., pan, A., Cacheda, F., Bellas, O., Carneiro, V.: Crawling the content hidden behind web forms. In: ICCSA. (2007) 322–333
8. S.W.Liddle, D.W.Embley, D.T.Scott, S.H.Yau: Extracting data behind web forms. In: Proc. of Advanced Conceptual Modeling Techniques. (2003) 402–413
9. Madhavan, J., Afanasiev, L., Antova, L., Halevy, A.: Harnessing the deep web: Present and future. In: Proc. of CIDR. (2009)
10. P.Wu, J.R.Wen, H.Liu, W.Y.Ma: Query selection techniques for efficient crawling of structured web sources. In: Proc. of ICDE. (2006) 47–56
11. Ipeirotis, P., Gravano, L.: Distributed search over the hidden web: Hierarchical database sampling and selection. In: VLDB. (2002)
12. Yang, M., H. Wang, L.L., Wang, M.: Optimizing content freshness of relations extracted from the web using keyword search. In: SIGMOD. (2010) 819–830

13. Dasgupta, A., Zhang, N., Das, G., Chaudhuri, S.: Privacy preservation of aggregates in hidden databases: why and how? In: SIGMOD. (2009) 153–164
14. L.Barbosa, J.Freire: An adaptive crawler for locating hidden-web entry points. In: Proc. of WWW. (2007) 441–450
15. R.Khare, Y.An, I.Song: Understanding deep web search interfaces: A survey. ACM SIGMOD Record **39**(1) (March 2010) 33–40
16. A.Ntoulas, P.Zerfos, J.Cho: Downloading textual hidden web content through keyword queries. In: Proc. of the Joint Conference on Digital Libraries (JCDL). (2005) 100–109
17. J.Lu, Y.Wang, J.liang, J.Chen, J.Liu: An approach to deep web crawling by sampling. In: Proc. of Web Intelligence. (2008) 718–724
18. L.Jiang, Z.Wu, Q.Feng, J.Liu, Q.Zheng: Efficient deep web crawling using reinforcement learning. In: Proc. of PAKDD. (2010) 428–439
19. Lawrence, S., Giles, C.: Searching the world wide web. Science **280**(5360) (1998) 98–100
20. K.Bharat, A.Border: A technique for measuring the relative size and overlap of public web search engines. In: WWW. (1998) 379–388
21. Bar-Yossef, Z., Gurevich, M.: Random sampling from a search engine's index. Journal of the ACM **55**(5) (2008) 1–74
22. Thomas, P., Hawking, D.: Evaluationg sampling methods for uncooperative collections. In: SIGIR. (2007) 503–510
23. Broder, A., Fontoura, M., Josifovski, V., Kumar, R., Motwani, R., Nabar, S., Panigrahy, R., Tomkins, A., Xu, Y.: Estimating corpus size via queries. In: Proc. of CIKM. (2006) 594–630
24. Zhang, M., Zhang, N., Das, G.: Mining a search engine's corpus:efficient yet unbiased sampling and aggregate estimation. In: Proc. of SIGMOD. (2011) 793–804
25. Dasgupta, A., Jin, X., Jewell, B., Zhang, N.: Unbiased estimation of size and other aggregates over hidden web databases. In: SIGMOD. (2010) 855–866
26. Dasgupta, A., Das, G., Mannila, H.: A random walk approach to sampling hidden databases. In: SIGMOD. (2007) 629–640
27. Henzinger, M., Heydon, A., Mitzenmacher, M., Najork, M.: On near-uniform url sampling. Computer Networks **33**(1-6) (2000) 295–308
28. Zhou, J., Li, Y., Adhikari, V., Zhang, Z.: Counting youtube videos via random prefix sampling. In: SIGCOMM, ACM (2011) 371–380
29. C.Olston, M.Najork: Web crawling. Informatoin Retrieval **4**(3) (2010) 175–245
30. L.Barbosa, J.Freire: Siphoning hidden-web data through keyword-based interfaces. In: Proc. of SBBD. (2004)
31. Y.Wang, J.Lu, J.Chen: Crawling deep web using a new set covering algorithm. In: Proc. of ADMA. (2009) 326–337
32. Valkanas, G., Ntoulas, A., Gunopulos, D.: Rank-aware crawling of hidden web sites. In: Proc. of In WebDB. (2011)
33. Wang, Y., Lu, J., Liang, J., Chen, J., Liu, J.: Selecting queries from sample to crawl deep web data sources. Web Intelligence and Agent Systems **10**(1) (2012) 75–88
34. Zheng, Q., Wu, Z., Cheng, X., Jiang, L., Liu, J.: Learning to crawl deep web. Information Systems (2013)
35. J.Lu, D.Li: Estimating deep web data source size by capture-recapture method. Informatoin Retrieval **13**(1) (2010) 70–95
36. T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein.: Introduction to Algorithms. Second edition edn. MIT Press and McGraw-Hill (2001)

37. G.K.Zipf: Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology. Addison-Wesley Press (1949)
38. J.Lu: Ranking bias in deep web size estimation using capture recapture method. Journal of Data and Knowledge Engineering **69**(8) (2010) 866–879
39. Erdos, P., Renyi, A.: On the evolutionof random graphs. Publ. Math Inst Hung Acad Sci **5** (1960) 17–61
40. Hristidis, V., Hu, Y., Ipeirotis, P.G.: Relevance-based retrieval on hidden-web text databases without ranking support. IEEE Transactions on Knowledge and Data Engineering **23**(10) (2010) 1555–1568
41. Richardson, M.: Beyond pagerank: Machine learning for static ranking. In: WWW. (2006) 707–715
42. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. In: CNIS. (1998) 107–117