TS-IDS Algorithm for Query Selection in the Deep Web Crawling

Conference Paper · September 2014

DOI: 10.1007/978-3-319-11116-2_17

3 authors, including:

Yan Wang
Central University of Finance and Economics
14 PUBLICATIONS **59** CITATIONS

SEE PROFILE

TS-IDS Algorithm For Query Selection in the Deep Web Crawling

Yan Wang[1], Jianguo Lu[23], and Jessica Chen[2]

[1] School of Information, Central University of Finance and Economics, China
dayanking@gmail.com
[2] School of Computer Science, University of Windsor, Canada
jlu@uwindsor.ca, xjche@uwindsor.ca
[3] Key Lab of Novel Software Technology, Nanjing, China.

Abstract. The deep web crawling is the process of collecting data items inside a data source hidden behind searchable interfaces. Since the only method to access the data is by sending queries, one of the research challenges is the selection of a set of queries such that they can retrieve most of the data with minimal network traffic. This is a set covering problem that is NP-hard. The large size of the problem, in terms of both large number of documents and terms involved, calls for new approximation algorithms for efficient deep web data crawling. Inspired by the TF-IDF weighting measure in information retrieval, this paper proposes the TS-IDS algorithm that assigns an importance value to each document proportional to term size (TS), and inversely proportional to document size (IDS). The algorithm is extensively tested on a variety of datasets, and compared with the traditional greedy algorithm and the more recent IDS algorithm. We demonstrate that TS-IDS outperforms the greedy algorithm and IDS algorithm up to 33% and 24%, respectively. Our work also makes a contribution to the classic set covering problem by leveraging the long-tail distributions of the terms and documents in natural languages. Since long-tail distribution is ubiquitous in real world, our approach can be applied in areas other than the deep web crawling.

Keywords: Deep web crawling, query selection, set covering problem, greedy algorithm, Zipf's law.

1 Introduction

The deep web [1] is the content that is dynamically generated from data sources such as databases or file systems. Unlike the surface web, where pages are collected by following the hyperlinks embedded inside collected pages, data from the deep web are guarded by search interfaces such as HTML forms, web services, or programmable web API, and can be retrieved by queries only. The deep web contains a much bigger amount of data than the surface web [2,3]. This calls for deep web crawlers to collect the data so that they can be used, indexed, and searched in an integrated environment. With the proliferation of publicly

available web services that provide programmable interfaces, where input and output data formats are explicitly specified, automated extraction of deep web data becomes more practical.

Deep web crawling is the process of collecting data from search interfaces by issuing queries. Sending queries and retrieving data are costly operations because they occupy network traffic. More importantly, many deep web data sources impose daily quota for the queries to be sent. In addition, most data sources paginate the matched results into many pages. All these restrictions call for the judicious selection of the queries.

The selection of queries can be modelled as a set covering problem. If we regard all the documents in a data source as the universe, each query is a subset of the documents it can match, the query selection problem is to find the subsets (the queries) to cover all the documents with minimal cost. Since the entire set of documents is not available, the queries have to be selected from a sample of partially downloaded documents [4,5,6,7]. In particular, [7,8] demonstrates that the queries selected from a sample set of documents can also work well for the entire data set. This paper will focus on the set covering algorithm on the sampled documents.

The set covering problem is NP-hard, and approximate algorithms have to be used. For large problems, the greedy algorithm is often recommended [9]. The greedy algorithm treats every document equally important. In deep web crawling, not every document is the same. A very large document containing virtually all possible terms is not an important document in the sense that it will be matched sooner or later by some terms. The query selection algorithm can almost neglect such documents since they can be covered by many terms. Therefore, the weight of a document is inversely proportional to the document size (IDS), or the distinct number of terms. In [10], we proposed and evaluated IDS approach.

This paper reveals that the document importance not only depends on the number of terms it contains, but also the sizes of these terms. The size of a term is the document it can cover, or its document frequency. A document that contains a small term can be covered with less redundancy, therefore they are of less importance in query selection. A document that is comprised of large terms only is costly to cover, since only large terms can be used. This kind of documents are more important, and the importance should be proportional to the minimal term size (TS) within the document.

Based on the above analysis, we propose the TS-IDS algorithm to select queries. It outperforms both greedy and IDS algorithms, and is extensively verified on a variety of datasets. We also exam the query selection process, and find that TS-IDS fundamentally differs from the other two approaches: both greedy and IDS methods prefer to use small terms (the terms with low document frequency) first, while TS-IDS tries to use frequent terms first even though it causes redundancy in the initial stage. In the final stage it uses the small terms to pick up remaining documents.

Our work also makes a contribution to the classic set covering problem by leveraging the distributions of the terms and documents in natural languages. Most of the set covering research assumes that the data are of normal or uniform distribution. For instance, the classic benchmark for set covering algorithms is the famous Beasley data, all are of normal distribution. However, most real-world data follows power law, including natural language texts [11]. We are the first to use data distribution to improve optimization algorithms as far as we are aware of.

In the following we will first give an overview of the related work on deep web crawling, focusing on the query selection task. Then we describe the problem formally in the context of set covering and bipartite graph. After explaining the motivations for the TS-IDS method, we present the detailed algorithm and its comparison with the greedy algorithm. Section 4 compares the three approaches, i.e., greedy, IDS, TS-IDS on four corpora.

2 Related work

In deep web crawling, the early work selects terms according to the frequencies of the terms [12], in the belief that high frequency terms will bring back more documents. Soon people realize that what is important is the number of new documents being retrieved, not the number of documents. If queries are not selected properly, most of the documents may be redundant. Therefore, query selection is modelled as a set covering [4] or dominating vertex [5] problem, so that the queries can return less redundancies. Since set covering problem or dominating vertex problem is NP-hard, the optimal solution can costly be found, especially because the problem size is very big that involves thousands or even more of documents and terms. Typically, a greedy method is employed to select the terms that maximize the new returns per cost unit. We realized that not every document is equal when selecting the queries to cover them. Large documents can be covered by many queries, no matter how the queries are selected. Therefore the importance of a document is inversely proportional to its size. We proposed IDS (inverse document size) algorithm in [10]. Our further exploration in this problem finds that the importance of the document depends not only on the number of the terms it contains, but also the sizes of those terms.

In addition to the optimization problem, query selection has also been modelled as reinforcement learning problem [13,14]. 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.

Query selection may have goals other than exhaustive exploration of the deep web data. For instance, in Google deep web crawling, the goal is to harvest some documents of a data source, preferably 'good' ones. Their focus is to look into many data sources, instead of exhausting one data source. In this case they use the traditional TF-IDF weighting to select the most relevant queries from the retrieved documents [6]. For another instance, data sources may be ranked and

only return the top-k matches per query. [15] studies the method to crawl the top ranked documents .

In addition to query selection, there are other deep web crawling challenges that are out of the scope of this paper. The challenges include locating the data sources [16,17], learning and understanding the interface and the returned results so that query submission and data extraction can be automated [18,17,19,20].

The set covering is an extensively studied problem [21,22,23]. The state of art commercial Cplex optimizer can find optimal solutions for small problems. When the matrix contains hundreds of elements, it often keeps on running for hours or days. The greedy algorithm is believed to be the better choice for large problems. Although many other heuristics are proposed, such as genetic algorithms[24], the improvements are very limited. The classic test cases are Beasley data [24]. Most of the data are synthesized from normal distribution, i.e., in the context of our document-term analogy, the document size and term size follow normal distributions. Some datasets even have uniform size for all the documents and terms. Such datasets can not reflect the almost universal power-law in real world, and prohibit the discovery of the algorithms such as TS-IDS. We also tested the TS-IDS algorithm on the Beasley datasets. The result is almost the same as the greedy algorithm. The reason is obvious–there is little variation of TS and DS, therefore it reduces to the greedy algorithm.

The inception of TS-IDS algorithm is inspired by the classic TF-IDF weighting. TS-IDS can be regarded as dual concept of TF-IDF. TF-IDF measures the importance of terms in a document in the presence of a collection documents, while TS-IDS measures the importance of a document covered by a term among a set of terms.

3 The TS-IDS algorithm

3.1 The Query Selection Problem

Given a set of documents $D = \{D_1, D_2, \ldots, D_m\}$ and a set of terms $T = \{T_1, T_2, \ldots, T_n\}$, each document contains a set of terms. In turn, each term covers a set of documents. Documents and terms form an undirected bipartite graph $G(D, T, E)$, where the nodes are D and T, E is the set of edges between T and D ($E \subseteq T \times D$). There is an edge between a document and a term iff the term occurs in the document. This graph can be represented by the document-term matrix $A = (a_{ij})$ where

$$a_{ij} = \begin{cases} 1, & \text{if } T_j \text{ occurs in } D_i; \\ 0, & otherwise. \end{cases} \tag{1}$$

Let d_i^D denote the degree of the document D_i (the size of document), and d_j^T the degree of term T_j (the size of term). Note that $d_i^D = \sum_{k=1}^{n} a_{ik}, d_j^T = \sum_{k=1}^{m} a_{kj}$. d_j^T is also called document frequency of the term in information retrieval. We call it term size to be consistent with document size. The query selection problem

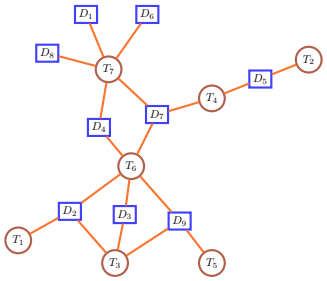

Fig. 1: A deep web data source is modelled as a bipartite graph.

can be modelled as the set covering problem [25]. In our context, it is a set covering problem where the cost for each term is the term size d_j^T:

Definition 1 *[Set Covering Problem]Given an $m \times n$ binary matrix $A = (a_{ij})$. The set covering problem is to find a binary n-vector $X = (x_1, \ldots, x_n)$ that satisfies the objective function*

$$Z = min \sum_{j=1}^{n} x_j d_j^T, \tag{2}$$

subject to

$$\begin{cases} \sum_{j=1}^{n} a_{ij} x_j \geq 1, & for\ all\ 1 \leq i \leq m, and \\ x_j \in \{0, 1\}, & 1 \leq j \leq n. \end{cases} \tag{3}$$

Here the first constraint requires that each document is covered at least once. The second constraint says that the solution is a binary vector consists of either one or zero, i.e., a term can be either selected or not selected. The solution cost $(\sum_{j=1}^{n} x_j d_j^T)$ is the total number of documents retrieved by the selected queries.

One may wonder that the cost of retrieving documents over a network depends on the total size of the documents rather than their number. In fact, for a crawler, when its goal is to download documents instead of URLs, it would be efficient to separate the URLs collection from the document downloading itself. More importantly, the cost for downloading documents is constant, for example, downloading all documents inside the target data source. Thus, the cost for downloading can be ignored. However, the cost for retrieving URLs is different from method to method. The cost of a query can be measured by the number of matched documents by it because of pagination function, i.e., web sites paginate the matched URLs into many *query sessions* (usually 10 URLs per session) not returning a long list. Each session needs one additional query (ordinarily it is a 'next' link). So the number of queries grows linearly with the number of matched documents. Hence, the cost for retrieving documents can be measured by the total number of documents retrieved by the selected queries $\sum_{j=1}^{n} x_j d_j^T$.

Fig. 2: The greedy algorithm.

$x_j = 0$, for $1 \leq j \leq n$;
$\mu_j = \sum_{i=1}^{m} a_{ij}$, for $1 \leq j \leq n$;
while *not all docs covered* **do**
  Find j that maximizes μ_j / d_j^T;
  $x_j = 1$;
  Remove column j;
  Remove all rows that contain T_j;
  $\mu_j = \sum_{i=1}^{m'} a_{ij}$ in the new matrix;
end

Fig. 3: The TS-IDS algorithm.

$x_j = 0$, for $1 \leq j \leq n$;
$\mu_j = \sum_{i=1}^{m} a_{ij} w_{ij}$, for $1 \leq j \leq n$;
while *not all docs covered* **do**
  Find j that maximizes u_j / d_j^T;
  $x_j = 1$;
  Remove column j;
  Remove all rows that contain T_j;
  $\mu_j = \sum_{i=1}^{m'} a_{ij} w_i$ in the new matrix.
end

Fig. 4: The greedy and the TS-IDS algorithms. The input is an $m \times n$ doc-term matrix. The output is an n-vector $X = (x_1, ..., x_n)$. $x_j = 1$ if term j is selected.

The set covering is an NP-hard problem. The greedy method described in Fig. 2 is proved to be an effective approximation algorithm [9]. It iteratively selects the best term that maximizes the new harvest per cost unit, as described in Fig. 2. Initially, all $x_j = 0$, meaning that no query is selected. Then it selects the next best query until all the documents are covered. The best term is the one that returns the most new documents per cost unit (μ_j / d_j^T). Note that m' is number of rows in the new matrix after the all covered documents are removed.

3.2 Motivation for the TS-IDS Algorithm

The greedy algorithm treats every document equally when it selects the best query using μ_j / d_j^T. Every document contributes unit one to μ_j, as long as it is a new document not being covered by other queries yet. However, not every document is of the same importance in the query selection process. This can be explained using the example in Fig. 1.

Document 7 and 6 have degrees 3 and 1, respectively, meaning that document 7 has more chances being captured than document 6. In this sense, D_6 is more important than D_7. If we include all the terms in the solution, D_7 is captured three times, while D_6 is captured only once. When we include a term, say T_4, in a solution, D_7 contributes only one third portion of the *new* document, because other terms could also be selected and D_7 will be covered again. Therefore, the importance of a document D_i is inversely proportional to document size d_i^D (IDS).

Furthermore, the document importance depends also on the term size (TS). Small (or rare) terms, whose degrees are small relative to other terms, are inherently better than large terms in achieving a good solution of set covering problem. Take the extreme case when all the terms have degree one. Every document will be covered only once without any redundancy. In general, when every term covers k documents, The greedy algorithm can approximate the optimal solution

within a factor of $\sum_{i=1}^{k} \frac{1}{i} \approx ln(k)$[26]. Small terms result in good solutions, while large terms prone to cause high cost. Documents containing only large terms are costly to cover, thus they are more important in the query selection process.

Looking back at our example again in Fig. 1. Both document 5 and 4 have degree two. Document 5 has a query whose degree is 1, while document 4 has two queries whose degrees are both 5. This means that document 5 can be covered by query 2 without any redundancy, while document 4 has to be covered by either T_6 or T_7, either one of them will most probably resulting in some duplicates. Therefore, we say that D_4 is more important than D_5 in the query selection process, even though their degrees are the same.

3.3 The TS-IDS Algorithm

For each document D_i, we define its document weight as follows:

Definition 2 (Document weight) *The weight of document D_i, denoted by w_i, is proportional to the minimal term size of the terms connected to D_i, and inversely proportional to its document size, i.e.,*

$$w_i = \frac{1}{d_i^D} \min_{T_j \in D_i} d_j^T. \tag{4}$$

With this definition, we give the TS-IDS as described in Fig. 3. Note that m' is the number of documents in the new matrix after covered documents are removed. The weighted new documents of a term T_j, denoted by μ_j, is the sum of the document weights containing term T_j, i.e., $\mu_j = \sum_{i=1}^{m} a_{ij} w_i$. Compared with the μ_j in the greedy algorithm, where $\mu_j = \sum_{i=1}^{m'} a_{ij}$, the difference in TS-IDS algorithm is the weight w_i for each document. Compared with the IDS algorithm where the weight is $1/d_i^D$, TS-IDS weights a documents not only by its length $(1/d_i^D)$, but also by the terms it contains. It gives a higher priority to short documents $(1/d_i^D)$ that contain popular terms only $(\min_{T_j \in D_i} d_j^T)$.

4 Experiments

4.1 Data

To demonstrate the performance of our TS-IDS algorithm, the experiment was carried out on four data collections that cover a variety of forms of web data, including regular web pages (Gov), wikipedia articles (Wikipedia), newswires (Reuters), and newsgroup posts (Newsgroup). 10,000 documents are selected uniformly at random from the original corpora. Table 1 summarizes the statistics of the four datasets, including the numbers of documents (m), the number of terms (n), and the degree properties of the documents and terms. Figures 5 plots in log-log scale the distributions of the document size and term size respectively. As expected, document size follows log-normal distribution, while term

Table 1: Statistics of the four datasets

Data	n	Document size			Term size		
		max	min	avg	max	min	avg
Reuters	56,187	833	8	106.7	3722	1	19.0
Wikipedia	224,725	2670	15	222.2	3041	1	9.89
Gov	164,889	3797	1	327.9	3028	1	19.8
Newsgroup	193,653	3836	1	245.6	3532	1	12.7

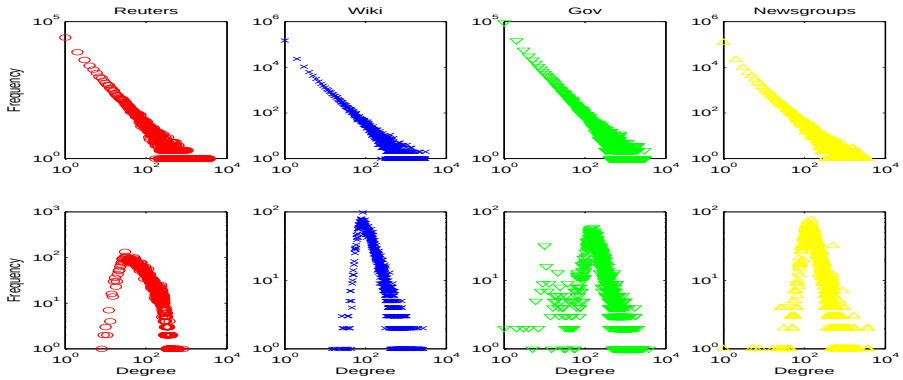

Fig. 5: Distributions of document sizes d_i^D and term sizes d_j^T of the four datasets. First row: term size distributions. Second row: document size distributions.

size follows power-law [27]. The highly skewed data distribution is the basis of the success of our algorithm. In traditional set covering studies, the benchmark test data, called Beasley data [28], are uniformly at random. For such data, term size (d_j^T) and document size (d_i^D) are mostly the same. We have also carried experiments on these data sets, and found that, as expected, TS-IDS and IDS algorithms perform very closely to the greedy algorithm. Due to space limitation, this paper focuses on the document-term matrix data only.

4.2 Results and Discussions

We run three algorithms, the Greedy (Fig. 2), the IDS algorithm in [10], and the TS-IDS algorithm (Fig. 3). Fig. 6 shows the comparison of these algorithms in terms of the solution quality, the average solution cost. The result is the average from running each algorithm 50 times for the same data. Each run may find a different solution because there may be a tie when selecting the best queries. When this happens, we select a random one from a set of equally good queries.

From Fig. 6, we can see that performance improvement differs from data to data. TS-IDS achieve better improvement for Reuters and Gov, but less for Wiki and Newsgroups, although all the four datasets have similar document size and term size distributions. For Reuters dataset, the TS-IDS outperforms the

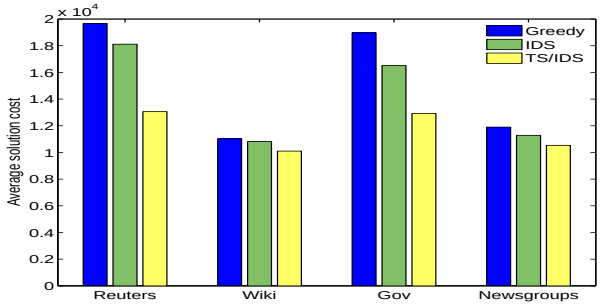

Fig. 6: Comparison between greedy, IDS, and TS-IDS algorithms on four datasets. The average solution costs are from 50 runs for each dataset.

IDS method around 24%, and outperforms the Greedy method around 33% in average. On the other hand, for Wiki and Newsgroup datasets, TS-IDS is better than IDS method and Greedy method around 6% and 10% respectively. This is because the solution costs for Wiki and Newsgroups are already close to one. Note that the best solution is one, whose redundancy is zero. Therefore there is little room for the TS-IDS algorithm to improve.

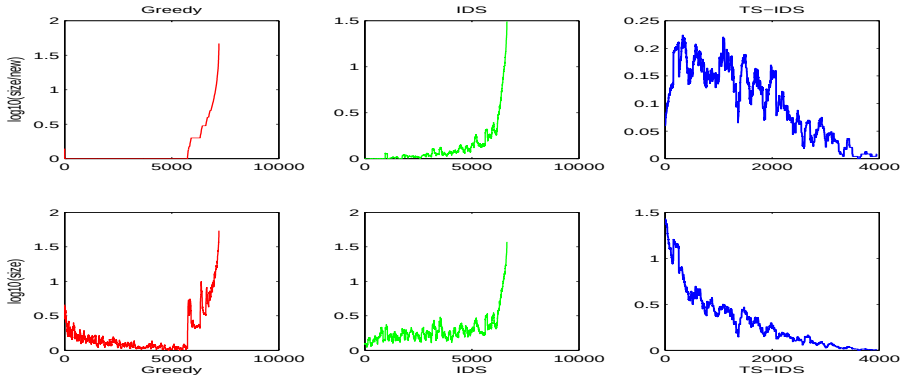

Fig. 7: Types of queries being selected in the process. First row: redundancy rate d_j^T/μ_j in the selection processes. Second row: corresponding query size d_j^T. Dataset is from Reuters corpus.

To gain the insight into these algorithms, we plot the *redundancy rate* (d_j^T/μ_j) for each query that is selected on Reuters data set in the first row of Figure 7. Note that the redundancy rate is the reciprocal of the new documents per returned document μ_j/d_j^T used in the greedy algorithm. Here the redundancy rate d_j^T/μ_j is used to analyse query selection process not μ_j/d_j^T since we want to

measure the quality of a query from the viewpoint of overlap. The x-axes are the number of queries selected. The y-axes are the redundancy rate of each query in log scale, i.e., the number of duplicates per new document in log10 scale. The redundancy rate is smoothed with moving window size 100. The second row shows the corresponding query size d_j^T, also in log10 scale and smoothed with the same window size. The data is obtained from the Reuters corpus. Other three datasets demonstrate similar patterns.

For the greedy algorithm, the first 5785 terms have redundancy rate one, meaning that all those terms have zero duplicates. After 5785 queries, the redundancy rate increases rapidly in exponential speed. For the last a few hundreds queries, the average blueredundancy rate is above 30. In the corresponding query size plot, we see that the first 5785 queries are mostly small ones. It starts with the average term size 3.2, then decreases because only smaller queries (i.e., small subsets) can be found that do not overlap with others already used. When overlapping occurs after first 5785 queries, the query size increases greatly, causing high redundancy rate. Because most of the small queries are already used in the first stage, it has to use large queries to cover the remaining documents.

The IDS algorithm improves the greedy algorithm by distinguishing documents according to their sizes. Long documents can be covered by many queries. Hence, in each iteration the IDS algorithm prefers the queries that cover smaller documents, even if the redundancy rate of the query is not the smallest. Thus, we can see that in the second column of Fig. 7 the redundancy rate is not always one for the first 5000 queries. However, the overall pattern is similar to that of the greedy algorithm: it tends to select small terms first, and it suffers the same problem of a surge in redundancy rate and query size when small and "good" queries are exhausted. A closer inspection on the dataset reveals that short documents normally contain popular words only. These documents can be picked up only by large queries, causing significant overlapping when most of the documents are already covered.

The TS-IDS algorithm solves this problem by giving higher priority to such documents. Since the document weight is proportional to term size, it starts with large queries to cover documents only containing high frequency terms, as we can see from column 3 of Fig. 7. Because of the large query, the redundancy rate is high at the beginning. The benefit of this approach is to save the small queries to fill in the small gaps in the final stage. These terms are not the best in terms of redundancy rate, but along the process of query selection, the redundancy rate decreases, and the overall performance is better. Surprisingly enough, the process is the inverse of the greedy algorithm: instead of selecting the best for the current stage, its current selection is in average worse than later selections.

The greedy algorithm not only has the highest redundancy rate here (1.94 compared with 1.81 for IDS and 1.31 for TS-IDS), but also uses more queries than other two methods. It selects 7256 queries, while IDS uses 6738 queries, and TS-IDS uses 4051 queries.

5 Conclusions

This paper presents the TS-IDS method to address the query selection problem in deep web crawling. It is extensively tested on textual deep web data sources whose document sizes and document frequencies follow the log-normal and power law distribution. By utilizing the distributions, TS-IDS method consistently outperforms the greedy and IDS algorithms. The success of the method is due to the highly skewed distributions of term size (Zipf's law) and document size (log-normal).

Without loss of generality, this paper discuss the set covering problem assuming each query is a single term. Our bipartite graph model can be extended to allow nodes representing multiple terms. Although this will greatly increase the graph size, such queries of multiple terms also follow power-law distribution. Therefore, our result can be extended to such queries as well.

In real deep web crawling scenario, usually it is impossible to directly apply a set covering algorithm to all the documents. Those documents are not known yet by the algorithm. Besides, a data source is usually so large that even approximate algorithms such as the ones discussed in this paper cannot handle it. The only option is to run the set covering algorithm on a sample subset of the data source. In [7], we showed that a solution that works well on a sample is also a good solution for the entire data source.

Our method is restricted to textual data sources that returns all the matched documents. Many data sources, especially large ones such Google, rank the matched documents and return only the top-k matches. This kind of data sources demand a different query selection strategy. One approach is to select and construct the low frequency queries, so that the number of matched documents is within the k limit.

6 Acknowledgements

This work is supported by NSERC, BSSF(No.14JGA001) and the 111 Project under Grant Number B14020.

References

1. M.K.Bergman: The deepweb: Surfacing hidden value. The Journal of Electronic Publishing **7**(1) (2001)
2. 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
3. Madhavan, J., Cohen, S., Dong, X., Halevy, A., Jeffery, S., Ko, D., Yu, C.: Web-scale data integration: You can afford to pay as you go. In: Proc. of CIDR. (2007) 342–350
4. 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

5. 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
6. J.Madhavan, D.Ko, L.Kot, V.Ganapathy, A.Rasmussen, A.Halevy: Google's deep-web crawl. In: Proc. of VLDB. (2008) 1241–1252
7. 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
8. 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
9. A.Caprara, P.Toth, M.Fishetti: Algorithms for the set covering problem. Annals of Operations Research **98** (2000) 353–371
10. Y.Wang, J.Lu, J.Chen: Crawling deep web using a new set covering algorithm. In: Proc. of ADMA. (2009) 326–337
11. Barabási, A., Albert, R.: Emergence of scaling in random networks. Science **286**(5439) (1999) 509–512
12. L.Barbosa, J.Freire: Siphoning hidden-web data through keyword-based interfaces. In: Proc. of SBBD. (2004)
13. Zheng, Q., Wu, Z., Cheng, X., Jiang, L., Liu, J.: Learning to crawl deep web. Information Systems (2013)
14. L.Jiang, Z.Wu, Q.Feng, J.Liu, Q.Zheng: Efficient deep web crawling using reinforcement learning. In: Proc. of PAKDD. (2010) 428–439
15. Valkanas, G., Ntoulas, A., Gunopulos, D.: Rank-aware crawling of hidden web sites. In: Proc. of In WebDB. (2011)
16. Sizov, S., Biwer, M., Graupmann, J., Siersdorfer, S., Theobald, M., Weikum, G., Zimmer, P.: The bingo! system for information portal generation and expert web search. In: Proc. of CIDR. (2003)
17. L.Barbosa, J.Freire: An adaptive crawler for locating hidden-web entry points. In: Proc. of WWW. (2007) 441–450
18. M.Alvarez, A.Pan, J.Raposo, F.Bellas, F.Cacheda: Extracting lists of data records from semi-structured web pages. Data Knowl Eng **64**(2) (2008) 491–509
19. C.A.Knoblock, K.Lerman, S.Minton, I.Muslea: Accurately and reliably extracting data from the web: a machine learning approach. IEEE Data Engineering Bulletin **23**(4) (2000) 33–41
20. J.Lu, D.Li: Estimating deep web data source size by capture-recapture method. Informatoin Retrieval **13**(1) (2010) 70–95
21. T.A.Feo, M.G.Resende: Greedy randomized adaptive search procedures. Journal of Global Optimization (1995) 109–133
22. L.W.Lorena, F.B.Lopes: A surrogate heuristic for set covering problems. European Journal of Operational Research **1994**(79) (1994) 138–150
23. Caprara, A., Fishetti, M., Toth, P.: A heuristic method for the set covering problem. Operations Research (1995)
24. J.E.Beasley, P.C.Chu: Theory and methodology. a genetic algorithm for the set covering problem. European Journal of Operational Research **94**(392-404) (1996)
25. T.H.Cormen, C.E.Leiserson, R.L.Rivest, C.Stein.: Introduction to Algorithms. Second edition edn. MIT Press and McGraw-Hill (2001)
26. Chvatal, V.: A greedy heuristic for the set-covering problem. Mathematics of operations research **4**(3) (1979) 233–235
27. G.K.Zipf: Human Behavior and the Principle of Least Effort: An Introduction to Human Ecology. Addison-Wesley Press (1949)
28. J.E.Beasley, K.Jornsten: Enhancing an algorithm for set covering problems. European Journal of Operational Research **58** (1992) 293–300