Crawling Deep Web Using a New Set Covering Algorithm

Yan Wang[1], Jianguo Lu[12], and Jessica Chen[1]

[1] School of Computer Science, University of Windsor
Windsor, Ont. Canada N9B 3P4
{wang16c,jlu,xjchen}@uwindsor.ca
[2] Key Lab of Novel Software Technology, Nanjing, China.

Abstract. Crawling the deep web often requires the selection of an appropriate set of queries so that they can cover most of the documents in the data source with low cost. This can be modeled as a *set covering* problem which has been extensively studied. The conventional set covering algorithms, however, do not work well when applied to deep web crawling due to various special features of this application domain. Typically, most set covering algorithms assume the uniform distribution of the elements being covered, while for deep web crawling, neither the sizes of documents nor the document frequencies of the queries is distributed uniformly. Instead, they follow the power law distribution. Hence, we have developed a new set covering algorithm that targets at web crawling. Compared to our previous deep web crawling method that uses a *straightforward* greedy set covering algorithm, it introduces *weights* into the greedy strategy. Our experiment carried out on a variety of corpora shows that this new method consistently outperforms its un-weighted version.

Keywords: deep web crawling, set covering problem, greedy algorithm.

1 Introduction

Deep web [1] is the web that is dynamically generated from data source such as databases or file system. Unlike surface web where data are available through URLs, data from a deep web are guarded by a search interface. The amount of data in deep web exceeds by far that of the surface web. This calls for deep web crawlers to excavate the data so that they can be reused, indexed, and searched upon in an integrated environment.

Crawling deep web [2–6] is the process of collecting hidden data by issuing queries through various search interfaces including HTML forms, web services and programmable web APIs. Crawling deep web data is important for several reasons, such as indexing deep web data sources, or backing up data.

Crawling the deep web often requires the selection of an appropriate set of queries so that they can cover most of the documents in the data source with low cost. Focusing on querying *textual data sources*, we provide a solution to this problem. Since it is not possible to select queries directly from the entire data source, we can make our selection from a *sample* of the database. It is shown that queries selected from a sample data source can perform on the total data source as well as on the sample one [7]. This leads to the following 4-step framework for crawling deep web [7]:

– Randomly selecting documents to build a sample database (*SampleDB*) from the original corpus (called *TotalDB*).
– Creating a set of queries called *query pool* (*QueryPool*) based on the *SampleDB*.
– Selecting a proper set of queries based on the *SampleDB* and the *QueryPool*.
– Mapping the selected queries into *TotalDB*.

An essential task in this framework is to select a proper set of queries based on the *SampleDB* and the *QueryPool* so that the cost of mapping the selected queries into *TotalDB* can be minimized. This can be modeled as a set covering problem which has been extensively studied. The conventional set covering algorithms, however, do not work well when applied to deep web crawling due to various special features of this application domain. Typically, most set covering algorithms assume the uniform distribution of the elements being covered, while for deep web crawling, neither the sizes of documents nor the document frequencies of the queries is distributed uniformly. Instead, they follow the power law distribution. In this regard, we have developed a new set covering algorithm that targets at web crawling. Compared to our previous deep web crawling method that uses a *straightforward* greedy set covering algorithm, it introduces *weights* into the greedy strategy. Our experiment carried out on a variety of corpora shows that this new method consistently outperforms its un-weighted version.

2 Problem formalization

We have shown in [7] the criteria to select $SampleDB$ and $QueryPool$. Our task here is to select from $QueryPool$ an appropriate set of queries so that they can cover *all* the documents in $SampleDB$. In terms of efficiency, we would like to keep the *overlap* minimal, where the overlap refers to the number of documents covered by more than one queries.

This can be modeled as a Set Covering Problem (SCP) [8] as below:

Definition 1. *Let $A = (a_{ij})$ be a 0-1 $m \times n$ matrix, and $c = (c_j)$ be an n-dimensional integer vector. Let $M = \{1, ..., m\}$ and $N = \{1, ..., n\}$. The value c_j ($j \in N$, $c_j > 0$) represents the cost of column j. We say that a column j ($j \in N$) covers a row i ($i \in M$) if a_{ij} =1. SCP calls for a minimum-cost subset S ($S \subseteq N$) of columns, such that each row is covered by at least one column.*

Example 1 *Table1 gives a matrix A, where each column represents a query in $QueryPool = \{q_1, q_2, \ldots, q_5\}$, and each row represents a document of $SampleDB$ $= \{d_1, \ldots, d_9\}$. $c_j = \sum a_{ij}$ is the document frequency (df) of the term. One solution of the problem is $Q = \{q_3, q_4, q_5\}$, which can be obtained by the greedy algorithm [7].*

Table 1. Matrix A: the input matrix for set covering algorithms

doc number	q_1	q_2	q_3	q_4	q_5
d_1	0	0	1	0	0
d_2	0	0	1	1	0
d_3	1	0	1	0	1
d_4	0	0	1	0	1
d_5	1	0	0	0	1
d_6	1	1	0	1	0
d_7	0	0	0	1	0
d_8	1	1	0	0	1
d_9	0	0	1	1	1

3 Weighted greedy algorithm

Constructing the set of queries in a step-by-step manner, simple greedy algorithm of the set covering problem selects the most cost effective query in each step. Let Q be a set of queries already selected. According to simple greedy algorithm, we select the next query q to cover as many as possible new documents (i.e. documents not covered by any query in Q) per *unit cost*. Here, the cost is the document frequency df, and *unit cost* can be represented by $1/df$. In other words, we select q to maximize the value of new/df where new is the number of documents covered by q but not by any query in Q.

As an improvement of simple greedy algorithm, we introduce the weight of queries into the greedy strategy.

If a document can only be matched by one query, apparently that query must be included into Q. In general, when selecting a query, we should pay more attention to cover small documents since usually they can be matched by only very few queries. We assign a weight to each document, where small documents have larger weights. With this intuition, we introduce the weight of a document:

Definition 2. *Let $D = \{d_1, ..., d_m\}$ be the SampleDB and $QP = \{q_1, ..., q_n\}$ be the QueryPool. We consider each document as a set of terms and use the notation $q_j \in d_i$ to indicate that a term $q_j (1 \leq j \leq n)$ occurs in the document $d_i (1 \leq i \leq m)$. The weight of a document with respect to QP and d_i, denoted by $dw_{d_i}^{QP}$ (or dw for short), is the inverse of the number of terms in QP that occurs in the document d_i, i.e.*

$$dw_{d_i}^{QP} = \frac{1}{|d_i \cap QP|}. \tag{1}$$

Definition 3. *The weight of a query q_j $(1 \leq j \leq n)$ in QP with respect to D, denoted by $qw_{q_j}^{QP}$ (or qw for short), is the sum of the document weights of all documents containing term q_j, i.e.,*

$$qw_{q_j}^{QP} = \sum_{q_j \in d_i, d_i \in D} dw_{d_i}^{QP}. \tag{2}$$

As for greedy strategy, we prefer queries q_j with larger number of qw. However, a larger number of qw should be obtained by fewer number of dw. In other words, we prefer queries with smaller df/qw. Our weighted greedy algorithm is based on choosing the next query with the smallest df/qw.

Example 2 *Based on the matrix in Table 1, the weights of the documents are shown in the top part of Table 2. The document frequencies, the weights of the queries, and their quotients are listed at the bottom of the table. For example, the weight of d_1 is one, the weight of d_2 is $1/2$, and the weight of q_2 is the sum of the weights of the documents that is covered by q_2, i.e., 0.66.*

Table 2. Matrix B: the initial weight table of the example corresponding to Matrix A

doc number	q_1	q_2	q_3	q_4	q_5
d_1	0	0	1	0	0
d_2	0	0	0.5	0.5	0
d_3	0.33	0	0.33	0	0.33
d_4	0	0	0.5	0	0.5
d_5	0.5	0	0	0	0.5
d_6	0.33	0.33	0	0.33	0
d_7	0	0	0	1	0
d_8	0.33	0.33	0	0	0.33
d_9	0	0	0.33	0.33	0.33
df	4	2	5	4	5
qw	1.49	0.66	2.66	2.16	1.99
df/qw	2.68	3.03	1.88	1.85	2.51

Table 3. Matrix B: The second-round weight table of the example

doc number	q_1	q_2	q_3	q_5
d_1	0	0	1	0
d_3	0.33	0	0.33	0.33
d_4	0	0	0.5	0.5
d_5	0.5	0	0	0.5
d_8	0.33	0.33	0	0.33
df	4	2	5	5
qw	1.16	0.33	1.83	1.66
df/qw	3.44	6.06	2.73	3.01

The more detailed weighted greedy algorithm is given in Algorithm 1.

Example 3 *Here we give an example to show how the weighted greedy method works. Table 1 is the matrix A of the example, Table 2 shows the initial values*

Algorithm 1: Weighted Greedy Algorithm.

```
Input: SampleDB, QueryPool QP, $m \times n$ Matrix $A$, where m=|SampleDB|
       and n=|QP|
Output: A set of queries Q
1. $Q = \phi$;
2. Let $B = (b_{ij})$ be a $m \times n$ matrix and $b_{ij} = a_{ij} \times dw_{d_i}^{QP}$ ;
3. Based on the matrix $B$, we calculate the query weight for each term and move
   the $q_j$ that minimizes $\frac{df_j}{qw_{q_j}^{QP}}$ into $Q$ ;
4. Check if the queries in $Q$ can cover all documents in SampleDB. If yes, the
   process ends;
5. Update matrix $B$ by removing the selected query and the documents that are
   covered by the query. Go to Step 3.
```

of weights and Table 4 shows the result from the weighted greedy method for the example.

For the weighted greedy algorithm, at the beginning, we calculate the value of df/qw for each query from Table 2 and find that q_4 has the minimal df/qw value (1.85) hence q_4 is selected as the first query. Then the column of q_4 and the covered rows, i.e., d_2, d_6, d_7 and d_9 are removed from the matrix B and the resulting matrix is shown in Table3. In the second round, q_3 becomes the second query because it has the minimal value for df/qw (2.73) and the matrix B is updated again. Finally, there are only two rows d_5 and d_8 left in the matrix B. q_1 is selected for its minimal df/qw value (4.82). After the third round, the selected queries can cover all documents and the weighted greedy algorithm terminates. For convenience to compare the two algorithms, we also give one solution for the example from the greedy algorithm as shown in Table 5. Greedy algorithm can produce several solutions depending on which query is selected in the first step of the algorithm. Initially all the queries has the same value for df/new, hence an arbitrary query can be selected. In our example, we select the query that is the same as the one of the weighted greedy algorithm. From Table 2, we can see that only q_3 and q_4 can cover d_1 and d_7 respectively. So q_3 and q_4 are required and they should be selected as early as possible. The useful information can be used by the weighted greedy method because such required query usually has a smaller df/qw value and it could be selected earlier.

Table 4. The result of the example by using the weighted greedy method

column	df	qw	df/qw	cost	unique rows
q_4	4	2.16	1.85	4	4
q_3	5	1.83	2.73	9	7
q_1	4	0.83	4.82	13	9

Table 5. The result of the example by using the greedy method

column	df	new rows	df/new	cost	unique rows
q_4	4	4	1	4	4
q_5	5	4	1.25	9	8
q_3	5	1	5	14	9

4 Experiment

We have run our experiments on the same data as that of [7] from four corpora: Reuters, Gov, Wikipedia and Newsgroup. These are standard test data used by many researchers in information retrieval. All *SampleDB* used have sample size 3000 and *relative size* [7] 20. We have used our search engine implemented in Lucene [9] to do all experiments in order to obtain the details of a data source such as its size. The details of the corpora are summarized in Table 6.

Table 6. Summary of test corpora

Name	Size in documents	Size in MB	Avg file size(KB)
Reuters	806,791	666	0.83
Wikipedia	1,369,701	1950	1.42
Gov	1,000,000	5420	5.42
Newsgroups	30,000	22	0.73

Table 7. The results based on 100 times running in *SampleDB* (G:greedy method; WG: weighted greedy method; Diff: difference)

	Reuters			Wiki			Gov			Newsgroup		
	G	WG	Diff	G	WG	Diff	G	WG	Diff	G	WG	Diff
MaxCost	14677	11421	0.22	11939	9468	0.20	18284	13853	0.24	13016	11067	0.14
MinCost	13613	11421	0.16	10514	9466	0.09	15945	13853	0.13	11955	11063	0.07
AveCost	14151	11421	0.19	11252	9467	0.15	16992	13853	0.18	12567	11065	0.11
SD	255.6	0	0.02	262.5	0.98	0.02	428.5	0	0.02	213	2	0.02

Since the algorithms are partly evaluated in terms of HR over OR, we give the definitions for HR and OR here.

Definition 4. *(Hit Rate, HR) Given a set of queries* Q=$\{q^1, q^2, ..., q^k\}$ *and a database DB. The hit rate of Q on DB, denoted by HR(Q,DB), is defined as the ratio between the number of unique data items collected by sending the queries* *in Q to DB and the size of the database DB, i.e., hit rate at the k-th step is:*

$$HR(Q,DB) = \frac{|\bigcup_{p=1}^{k} S(q^p, DB)|}{|DB|} \tag{3}$$

Definition 5. *(Overlapping Rate, OR) Given a set of queries Q=$\{q^1,\ldots,q^k\}$, the overlapping rate of Q on DB, denoted by OR(Q,DB), is defined as the ratio between the total number of collected data items and the number of unique data items retrieved by sending queries in Q to DB. i.e.,overlapping rate at the k-th step is:*

$$OR(Q,DB) = \frac{\sum_{p=1}^{k} |S(q^p, DB)|}{|\bigcup_{p=1}^{k} S(q^p, DB)|} \tag{4}$$

Table 7 shows that our weighted greedy method is much better than the greedy method. As the result of the greedy method depends on which query is slected in the first step of the algorithm, we have given some statistic results here, such as the standard deviation, maximum and minimal values. Our weighted greedy method also has such problem: initially there can be more than one queries having the same value of df/qw. However, seldom occurs in practice. From Table 7 we can see that (i) even the maximum cost of our weigted greedy method is better than the minimal cost of the greedy method; (ii) on average our method outperforms the normal greedy method by approximately 16%.

The difference between the results of the two methods can be explained as follows. The term with lower weight may cause less overlap if it is selected as a query because there will be fewer other terms contained by the same document which can be selected as queries in future. Based on this observation, we can see that *df/qw* of a term can somehow show an *overlapping possibility* of the term. At the same time, the value of *df/qw* of a term can also represent a *requirement degree*. For example, if the df of a term is 1 and the corresponding document also only contains the term, the value of *df/qw* of the term is 1 (the minimal value of *df/qw*). In this case, of course, the term is *required* and should be selected as early as possible.

Figure 1 and Figure 2 show the relationship between the df and the number of terms and the relationship between the document size and the number of documents in the *SampleDB*. From these two figures, we can see that both follow the power law distribution: most of the documents only contain few terms; most of the terms are contained by very few documents; but a few high df terms are in most of the documents. In such a situation, documents only containing one or two high df terms can have a smaller value of df/qw and thus are more desired. If such required high df terms can be selected earlier, the result should be better in *SampleDB*. Figure 3 shows that the weighted greedy method can select many high df terms earlier than the greedy method in *SampleDB*.

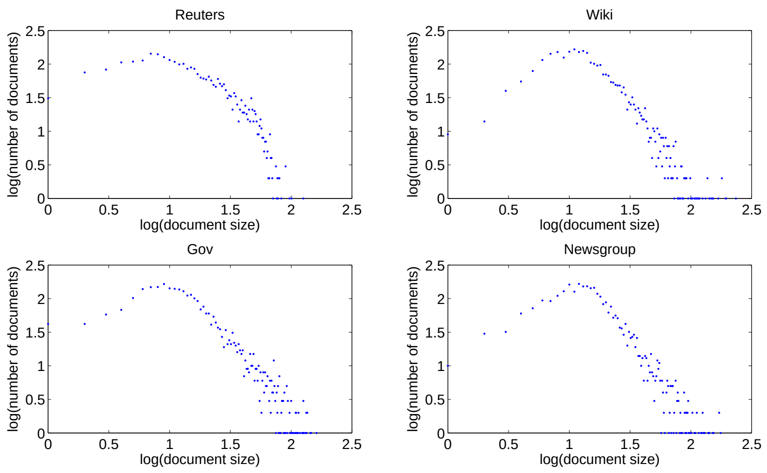

Fig. 1. the distribution of document size in *SampleDB*

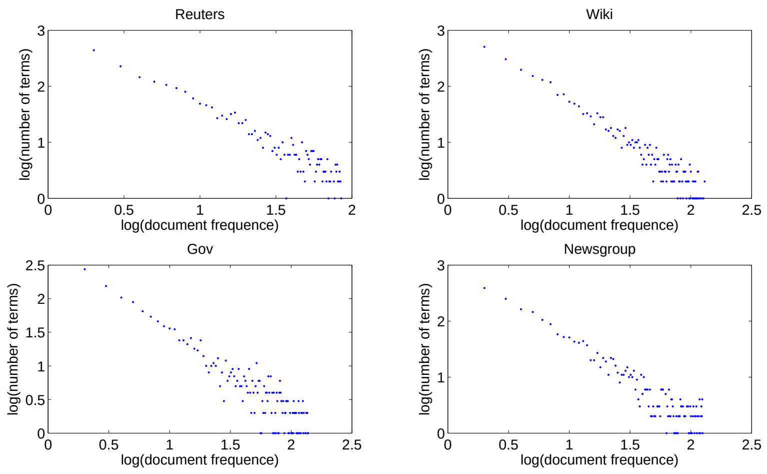

Fig. 2. the distribution of df in *SampleDB*

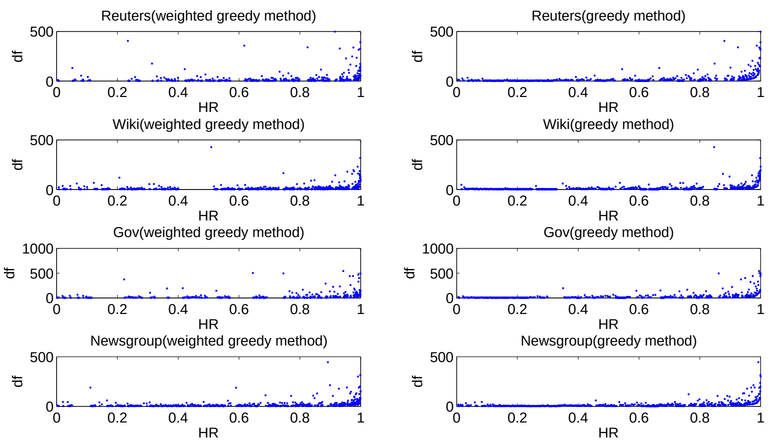

Fig. 3. the df changes in SampleDB

Figure 4 is derived from the experiments in *SampleDB*. In this figure, we can see that the values of *df/tw* of the terms selected by the weighted greedy method are much smaller than those selected by the greedy method especially when HR ranges from 0% to 60% in *SampleDB*. The smaller number of *df/tw* implies that the terms selected by the weighted greedy method introduce less overlapping. Furthermore, as each *SampleDB* is a representative of its corresponding *TotalDB*, a term having a small overlapping possibility and a high requirement degree in *SampleDB* will have these properties preserved in *TotalDB*.

Now we discuss the performance of the two methods in *TotalDB*. Figure 5 shows that the weighted greedy method has a better performance than the greedy method and the mapping coverages of the two methods are good enough. For example, on Wikipedia corpus, when HR is 90%, the weighted greedy method is 15.1% better than the greedy method. The size of Wikipedia corpus is 1.36 million of documents. Thus, we can save 0.15*0.9*1.3 million of documents which is a significant saving.

5 Related work

There are many research works on deep web crawling. Basically, two research challenges have been substantially addressed. One is learning and understanding the interface and the returning result so that query submission [10, 11] and data extraction [12] can be automated. The other is how to harvest hidden documents as many as possible with a low cost [2, 5, 6, 13]. Here we discuss several closely related works in the second area. The deep web is guarded by search interface,

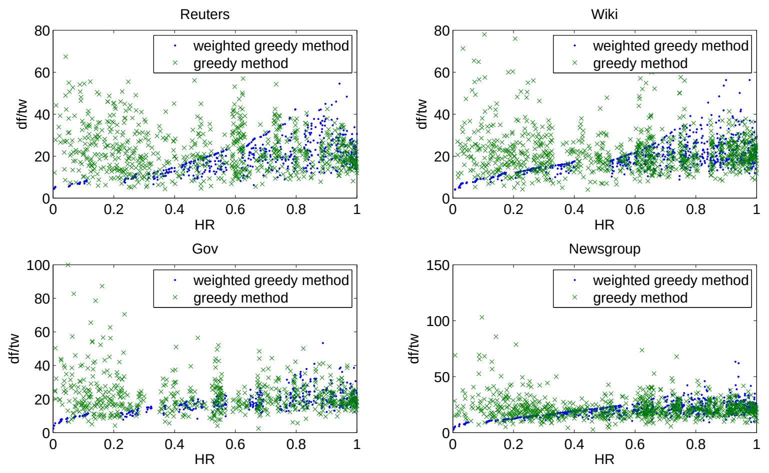

Fig. 4. Comparison on df/qw between the two methods

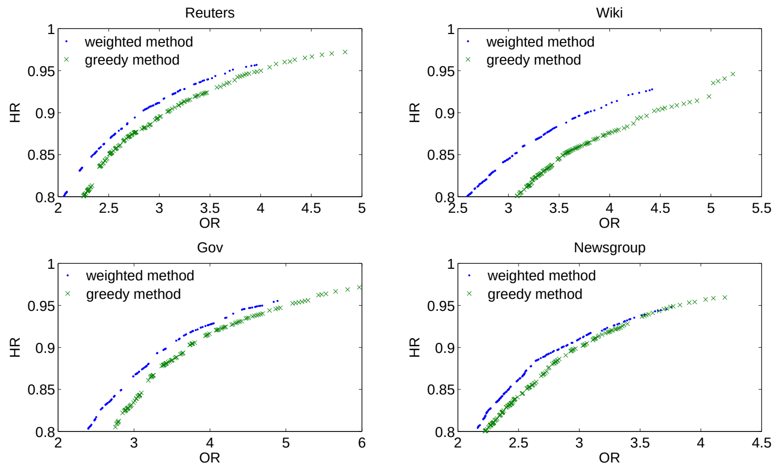

Fig. 5. Comparison on mapping results HR from 80%

hence difficult to access. A sampling-based method could be a good choice for deep web crawling.

Generally speaking, the sampling-based methods have two directions. One is to use Information Retrieval Technologies to crawl the deep web [14–16]. In [15], the general idea is that firstly some carefully designed queries are issued to the database and then some samples as returns from the database are obtained. Secondly those samples are analyzed and further classified by some classification algorithms to obtain some typical terms that can accurately describe the database. Finally, those typical terms are used as queries to harvest the database. The other one is to use some selection methods to create a sample that can be representative of the database and the principle of human language to crawl the deep web [2, 17, 5]. For example, the Zip's law can be used to evaluate the frequency of a term in the database based on a sample [5]. In [17], the authors present a new technique to automatically create a description (a sample) for the database. They argue that accurate description can be learned by sampling a text database with simple keyword-based queries. Actually our framework for crawling the deep web is based on Callan and Connells research work in [17].

For the first kind of methods, usually they are based on existing domain knowledge. The authors suppose that the database is so heterogeneous and it is hard to obtain a high HR with queries selected randomly from a dictionary hence the emphasis is to have a high HR by issuing queries. For the second kind of methods, the authors try to minimize the number of queries with a high HR. On the contrary, we argue that the bottleneck to deep web crawling is the number of documents crawled, not the queries issued. Therefore our algorithm tries to minimize the documents retrieved, not the queries sent. Another difference from the Ntoulas et al's method [5] is that they estimate the returns and overlaps of the i-th query based on the documents downloaded by the previous $i-1$ queries. This approach requires the downloading of almost all the documents, hence it is not efficient. Our approach only requires a small portion of the data source, and learn the appropriate queries from the sample database.

6 Conclusions

In an earlier paper [7], we proposed a deep web crawling method based on sampling. In that paper, we showed that it is effective to learn appropriate queries from a sample data source, and empirically identified the appropriate sizes of the sample and the query pool. This paper presents a better algorithm, the weighted greedy algorithm, to select the queries from a sample data source. The weighted greedy method has a much better result than the greedy method in *SampleDB* from the four corpus so that the result from the weighted greedy method can beat the result from the greedy method in *TotalDB*. In the *SampleDB*, the weighted greedy method can select the term as a query which has lower overlapping possibility and higher requirement degree as earlier as possible and such properties can be successfully described by the query weight.

Acknowledgments. This work is supported by Key Laboratory of Novel Software Technology and the Natural Sciences and Engineering Research Council of Canada.

References

1. M.K.Bergman: The deepweb: Surfacing hidden value. The Journal of Electronic Publishing **7**(1) (2001)
2. L.Barbosa, J.Freire: Siphoning hidden-web data through keyword-based interfaces. In: Proc. of SBBD. (2004)
3. C.H.Chang, M.Kayed, M.R.Girgis, K.F.Shaalan: A survey of web information extraction systems. IEEE Transactions on Knowledge and Data Engineering **18**(10) (Oct. 2006) 1411–1428
4. S.W.Liddle, D.W.Embley, D.T.Scott, S.H.Yau: Extracting data behind web forms. In: Proc. of Advanced Conceptual Modeling Techniques. (2002)
5. 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
6. 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
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. A.Caprara, P.Toth, M.Fishetti: Algorithms for the set covering problem. Annals of Operations Research **98** (2004) 353–371
9. E.Hatcher, O.Gospodnetic: Lucene in Action. Manning Publications (2004)
10. 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
11. M.L.Nelson, J.A.Smith, I.G.D.Campo: Efficient, automatic web resource harvesting. In: Proc. of RECOMB. (2006) 43–50
12. 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
13. P.G.Ipeirotis, P.Jain, L.Gravano: Towards a query optimizer for text-centric tasks. ACM Transactions on Database Systems **32** (2007)
14. L.Gravano, P.G.Ipeirotis, M.Sahami: Query- vs. crawling-based classification of searchable web databases. Bulletin of the IEEE Computer Society Technical Committee on Data Engineering **25**(1) (2002) 1–8
15. J.Caverlee, L.Liu, D.Buttler: Probe, cluster, and discover: focused extraction of qa-pagelets from the deep web. In: Proc. of the 28th international conference on Very Large Data Bases. (2004) 103–14
16. A.Ibrahim, S.A.Fahmi, S.I.Hashmi, H.Choi: Addressing effective hidden web search using iterative deepening search and graph theory. In: Proc. of IEEE 8th International Conference on Computer and Information Technology Workshops. (2008) 145–149
17. J.Callan, M.Connell: Query-based sampling of text databases. ACM Transactions on Information Systems (2001) 97–130