2012

Query Selection in Deep Web Crawling

Yan Wang

Query Selection in Deep Web Crawling

by

Yan Wang

A Dissertation
Submitted to the Faculty of Graduate Studies
through Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy at the
University of Windsor

Windsor, Ontario, Canada
2012

Query Selection in Deep Web Crawling

by

Yan Wang

APPROVED BY:

Dr. Chengzhong Xu, External Examiner
Wayne State University

Dr. Xiang Chen
Department of Electronic Engineering

Dr. Arunita Jaekel
Department of Computer Science

Dr. Dan Wu
Department of Computer Science

Dr. Jessica Chen, Co-supervisor
Department of Computer Science

Dr. Jianguo Lu, Co-supervisor
Department of Computer Science

Dr. Chunhong Chen, Chair of Defense
Department of Electronic Engineering

10 May 2012

Declaration of Previous Publication

This thesis includes three original papers that have been previously published in blind reviewed journal and conference proceedings, as follows:

Thesis Chapter	Publication title/full citation	Publication status
Part of Chapter 3	Yan Wang, Jianguo Lu and Jessica Chen. Crawling Deep Web Using a New Set Covering Algorithm. Proceedings of 5th International Conference On Advanced Data Mining and Applications (ADMA'09), page 326-337, Springer.	published
Major part of Chapter 4	Yan Wang, Jianguo Lu, Jie Liang, Jessica Chen, Jiming Liu. Selecting Queries from Sample to Crawl Deep Web Data Sources , Web Intelligence and Agent Systems, 2012.	published
	Jianguo Lu, Yan Wang, Jie Liang, Jiming Liu and Jessica Chen. An Approach to Deep Web Crawling by Sampling, Proceedings of the 2008 IEEE/WIC/ACM International Conference On Web Intelligence (WI'08). Page:718-724.	published

I certify that I have obtained a written permission from the copyright owners(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor. I declare that, to the best of my knowledge, my thesis does not infringe upon anyone's copyright or violate any proprietary rights and that any ideas, techniques, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission

from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University of Institution.

Abstract

In many web sites, users need to type in keywords in a search Form in order to access the pages. These pages, called the deep web, are often of high value but usually not crawled by conventional search engines. This calls for deep web crawlers to retrieve the data so that they can be used, indexed, and searched upon in an integrated environment. Unlike the surface web crawling where pages are collected by following the hyperlinks embedded inside the pages, there are no hyperlinks in the deep web pages. Therefore, the main challenge of a deep web crawler is the selection of promising queries to be issued.

This dissertation addresses the query selection problem in three parts: 1) *Query selection in an omniscient setting where the global data of the deep web are available.* In this case, query selection is mapped to the set-covering problem. A weighted greedy algorithm is presented to target the log-normally distributed data. 2) *Sampling-based query selection when global data are not available.* This thesis empirically shows that from a small sample of the documents we can learn the queries that can cover most of the documents with low cost. 3) *Query selection for ranked deep web data sources.* Most data sources rank the matched documents and return only the top k documents. This thesis shows that we need to use queries whose size is commensurate with k, and experiments with several query size estimation methods.

Keywords deep web, set covering, ranked data source, document frequency estimation

Dedication

To our heroes

This dates back to 1840 Opium War, in order to oppose internal and external enemies and fight for national independence and liberation, the freedom of happiness in China, the People's heroes sacrificed their lives in the past struggles.

Acknowledgements

I would like to acknowledge the important role of my PhD committee and the External Examiner, and thank them for their enlightening and encouraging comments and reviews.

I wish to express my gratitude to Dr.Jianguo Lu and Dr.Jessica Chen, my co-supervisors, for their valuable assistance and support during my thesis work, and for their persistent guidance through out my study during Ph.D. program. Especially for Dr. Chen, she has guided me for seven years since I was 24 years old. She gives me spirit and soul.

To my parents, I owe your everything and I can never fully pay you back whatever I say or do. I love you and I miss you even when you are next to me. To my elder brother and younger sister, I appreciate you for many years support and own you for taking my duty to parents without any complaint.

To my fiancee, Nan Li, many years lonely waiting takes away your passion, your beauty and your youth. But the one thing never changed is your will to helping me fulfil my dream. I stand speechless and humbled in front of you for your sacrifice and great love.

To all who help me get through the journey, without you I will never be who I am now.

Contents

Declaration of Previous Publication iii

Abstract v

Dedication vi

Acknowledgements vii

1 Introduction 1

2 Related work 6
- 2.1 Query selection problem . . . 6
  - 2.1.1 Unstructured data sources . . . 6
  - 2.1.2 Structured data source . . . 10
- 2.2 Automated form filling . . . 11
- 2.3 Automated searchable form discovery . . . 13
- 2.4 Sampling techniques . . . 15

3 Query selection using set covering algorithms 18
- 3.1 Introduction . . . 18
- 3.2 Set covering . . . 19
- 3.3 Greedy algorithm . . . 22
- 3.4 Introducing weight to greedy algorithm . . . 23
  - 3.4.1 Weighted algorithm . . . 25
  - 3.4.2 Redundancy removal . . . 26
- 3.5 Experiments and analysis . . . 27
  - 3.5.1 Experimental setup . . . 27
  - 3.5.2 Data . . . 28
  - 3.5.3 Results . . . 31
  - 3.5.4 Impact of data distribution . . . 41
- 3.6 Conclusion . . . 47

4 Sampling-based query selection method **48**
4.1 Introduction . . . 48
4.2 Problem formalization . . . 49
4.2.1 Hit rate and overlapping rate . . . 49
4.2.2 Relationship between HR and OR . . . 51
4.3 Our method . . . 52
4.3.1 Overview . . . 52
4.3.2 Creating the query pool . . . 54
4.3.3 Selecting queries from the query pool . . . 56
4.4 Experiment environment . . . 57
4.4.1 Hypothesis I . . . 58
4.4.2 Hypothesis II . . . 60
4.4.3 Hypothesis III . . . 60
4.4.4 Comparing queries on other data sources . . . 67
4.4.5 Comparing queries with Ntoulas' method . . . 68
4.4.6 Selecting queries directly from DB . . . 71
4.5 Conclusion . . . 71

5 Ranked data source **74**
5.1 Motivation . . . 74
5.2 Our crawling method . . . 78
5.3 Frequency estimation . . . 80
5.3.1 Introduction . . . 80
5.3.2 Evaluation of the estimation methods . . . 88
5.4 Crawling evaluation . . . 96
5.5 Conclusion . . . 100

6 Conclusions **102**

A Details of the comparison between greedy and weighted greedy algorithms **106**

B Detail of the example for Simple Good-Turing estimation **114**

C Maximum likelihood estimation **117**

Vita Auctoris **128**

List of Tables

2.1 The consistency between the conclusions of Callan's work [1] and our sampling-based query selection method 16

3.1 The doc-term Matrix A in Example 1 22
3.2 The initial weight table of Example 2 corresponding to Matrix A . . 26
3.3 The second-round weight table of Example 2 27
3.4 The properties of our data sources (avg(df): the average of document frequencies, max: the maximum document degree, min: the minimal document degree, avg(CV): the average of CVs, SD(CV): the standard deviation of CVs, number: the number of data sources.) 30
3.5 The properties of the Beasley data 31
3.6 The results with redundancy on our data sources. 32
3.7 The results without redundancy on our data sources. 35
3.8 The average results with redundancy for each set of the Beasley data. 39
3.9 The average result without redundancy for each set of Beasley data. 39

4.1 Summary of test corpora 57

5.1 The statistics of the samples for the experiments (m:the number of the documents in D, n: the number of the terms in D, r_1:$\frac{m}{|DB|}$, r_2: the ratio of the number of all the terms in D to the number of all the terms in DB). 90
5.2 The average of the parameter values for the Zipf-law-based estimation and SGT based on 10 3000-document samples. 90
5.3 The average errors for three estimators according to different corpus samples and sample df ranges. 91
5.4 The results of twenty randomly selected terms from a Newsgroup 3000-document sample (f: sample document frequency). 95
5.5 The number of the candidate terms of the three methods in each experiment. 97
5.6 Comparison of the three methods ($Imp = \frac{OR_{random}(OR_{popular}) - OR_{ours}}{OR_{random}(OR_{popular})}$). 99

A.1 Greedy vs weighted greedy. The results with redundancy based on 100 times running experiments on the Beasley data. 106

A.2 Greedy vs weighted greedy. The results without redundancy based on 100 times running experiments on Beasley data. 108

B.1 The full set of (f, n_f, Z_f, log10(f), log10(Z_f), f^*) tuples for the example of Section 2.4.2 114

List of Figures

1.1 The key components in this dissertation for unranked and ranked deep web data sources. 4

2.1 An example from Amazon.com website 12
2.2 The framework of form-focused crawling method 14

3.1 The illustration of the textual data source DB in Example 1. Each dot or circle represents one document or term, respectively. 21
3.2 The distribution of the document degrees using 'logarithmic binning' in log-log scale for our data sources. The bin size of the data increases in 2^i $(i = 1, 2, \ldots)$ 30
3.3 The distributions of element degrees of part of the Beasley data. The bin size is 5. 32
3.4 The results of all experiments on each corpus data with redundancy based on 100 runs. 'o': the greedy method, '.': the weighted greedy method. 33
3.5 The average document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on our data sources. The bin size is 5%. 35
3.6 The average values of new/df in the greedy and weighted greedy query selection processes on our corpus data sources. The bin size is 5%. 36
3.7 The average document frequencies of the selected documents in the greedy and weighted greedy query selection processes on our data sources. The bin size is 5%. 37
3.8 The results of all experiments on Beasley data with redundancy. G: the greedy method, W: the weighted greedy method. 40
3.9 The average document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on parts of the Beasley data. The bin size is 10%. The solid and dashed lines represent the results of the greedy and weighted greedy methods respectively. 42

3.10 The average values of *new*/*cost* for each column in the greedy and weighted greedy query selection processes on parts of the Beasley data. The bin size is 10%. In subgraph e.1, the histogram is replaced with the scatter plot because some values of *new*/*cost* are zero and no column is selected in the corresponding coverage ranges. 43
3.11 The average document frequencies of selected columns in the greedy and weighted greedy query selection processes on part of Beasley data. The bin size is 10%. In subgraph e.1, the histogram is replaced with the scatter plot and the reason is same as Figure 3.10. 44
3.12 the relationship between the CV of document (element) degree and the average of the improvement of the weighted greedy method on our and Beasley data with redundancy. 46

4.1 Our sampling-based query selection method. 52
4.2 Impact of sample size on HR projection. The queries are selected from D and cover above 99% of the documents in D. The HR in DB is obtained when those queries are sent to the original data source DB. $\mu = 20$. 59
4.3 Impact of sample size on OR projection. X axis is sample size, Y axis is HR. Sample size is from 100 to 4,000 documents and $\mu = 20$. 61
4.4 The impact of the sample size and the average document degree on HR. 63
4.5 The impact of the sample size and the average document degree on OR. 64
4.6 The impact of the value of the average document degree on OR improvement from the comparison between our and random methods for Reuters corpus. 66
4.7 Apply queries selected in one corpus to other corpora. Sample size is 3,000, $\mu = 20$. Each sub figure shows the querying results for the four data sources with the queries selected from one particular corpus. 68
4.8 Comparison with queries selected by using Ntoulas' method. For our method, the sample size is 3000, $\mu = 20$ and the range of the sample document frequencies of the queries are between 2 and 500. For Ntoulas' method, the stopping criterion is that no new document is returned after 10 consecutive queries sent. 70
4.9 Comparison with queries selected directly from DB. Each sub figure shows the querying results for queries selected from one particular data source. 72

5.1 Scatter plots for the query results from different types of queries. For each experiment, 100 queries are sent. X axis represents the document rank and the return limit $k = 20$. Sub figure (A) queries with $F = 40$; (B) queries with $40 \leq F \leq 80$; (C) queries with $F = 80$. The data source is the Reuters. 75
5.2 The scatter plot for the query result. 100 queries whose $F \leq 20$ are sent and the return limit $k = 20$. 77
5.3 Our query selection method for ranked deep web data sources . . . 79
5.4 The trend of sample document frequency in Example 4 on log-log scales. High sample document frequencies becomes horizontal along the line n_1. 84
5.5 The smoothing result of n_f shown in Figure 5.4 for our SGT example on log-log scales. $Z_f = \frac{2n_f}{f'' - f'}$ and the line of best fit $y = 3.12 - 1.707 \times log_{10}(x)$ is obtained by using the least-squares curve fitting algorithm provided in Matlab. 85
5.6 An artificial example to illustrate the basic idea of Ipeirotis' estimator. 87
5.7 The example of the Zipf-law-based estimation method. 89
5.8 The average errors of the three estimators for the terms with $1 \leq f \leq 10$ based on four different corpus samples. 93
5.9 The average of the document frequencies and the estimated document frequencies of the three estimators for the low frequency terms based on four 3000-document samples. The MLE: the average of estimated document frequencies from the MLE, the SGT: the average of estimated dfs from the SGT, Zipf: the average estimated dfs from the Zipf's-law-based estimation. 94
5.10 The results of the three methods on different corpora. The return limit k is 1000. Documents are statically ranked. MLE, SGT, Zipf: our methods with the MLE, the SGT and the Zipf-law-based estimation methods respectively. 97
5.11 The zoomed-in area of each subgraph in Figure 5.10. The range of HR is up to 10%. 98

A.1 The results of all the experiments on each corpus data based on 100 runs. '.': greedy with redundancy, '*': weighted greedy with redundancy, 'o': greedy without redundancy, '+': weighted greedy without redundancy. 111
A.2 Part of the results of all the experiments on the Beasley data based on 100 runs. '.': greedy with redundancy, '*': weighted greedy with redundancy, 'o': greedy without redundancy, '+': weighted greedy without redundancy. 112

A.3 The relationship between the CV of the document (element) degree and the average of the improvement of the weighted greedy method on our and the Beasley data without redundancy. 113

C.1 The probability mass functions of the document frequency of query q with $N = 1500$ and $p = 0.004$(A) and $p = 0.008$(B) 118
C.2 The likelihood function $L(p|N, f(q))$ with $N = 1500$ and $f(q) = 12$ is based on the binomial distribution model described in the text. . 119

Chapter 1

Introduction

The deep web [2] 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 [3], and can be retrieved by queries only. The deep web contains a much bigger amount of data than the surface web [4, 5]. 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. It has been studied from two perspectives. One is the study of the macroscopic views of the deep web, such as the number of the deep web data sources [6, 4, 7], the shape of such data sources (e.g., the attributes in the HTML form) [7], and the total number of pages in the deep web [2].

When crawling the deep web, which consists of tens of millions of HTML forms, usually the focus is on the coverage of those data sources rather than exhaustively crawling the content inside one specific data source [7]. That is, the breadth, rather than the depth, of the deep web is preferred when the computing resource of a crawler is limited. In this kind of breadth-oriented crawling, the challenges are locating the data sources [8, 9, 10, 11], learning and understanding the interface and the returned results so that query submission and data extraction can be au-

tomated [12, 11, 13, 14].

Another category of crawling is depth-oriented, focusing on one designated deep web data source, with the goal to garner most of the documents from the given data source [15, 16, 17, 7]. In this realm, the crucial problem is the *query selection problem*, that is, to cover most of the documents in a specific data source with minimal cost by submitting appropriate queries.

There are many existing works addressing the query selection problem [15, 16, 17, 18, 19, 20, 7] but still some issues remain as follows: 1) How to evaluate query selection algorithms; 2) How to optimize the query selection problem; 3) What the input of query selection algorithms at beginning is; 4) How to crawl the deep web data sources with return limit.

Based on the above issues, in this dissertation, we present a novel technique to addressing the query selection problem. It contains three parts:

- *Query selection using set covering algorithms*: first we map the query selection problem to the set covering problem. If we let the set of documents in a data source be the universe, each query represents the documents it matches, i.e., a subset of the universe. The query selection problem is thus cast as a set covering problem, and the goal is to cover all the documents with minimal sum of the cardinalities of the queries.

  Although set covering problem is well studied [21, 22, 23, 24] and numerous algorithms even commercial products such as CPLEX [25] are available, the greedy algorithm remains the convenient choice for large set covering problems. A conventional greedy algorithm assigns the same weight to each document, which may be good in other domains but not in our application. In deep web data sources, one empirical law on the document size is that its distribution is highly skewed, close to power law or log-normal distribution. Many documents are of small size, while the existence of very large documents cannot be neglected, so called long tail. Those large documents can be covered by many queries, therefore their importance or weight should be smaller. We assign the reciprocal of the document size as the weight of a document, and select the next best query accordingly.

  We conducted experiments on a variety of test data, including the Beasley data [26], the standard benchmarks for set covering problem, and four typical

deep web data sources. The results show that, with same time complexity, our weighted greedy algorithm outperforms the greedy algorithm in both Beasley and our data and, especially, for our data (heterogeneous data). Furthermore, we argue that data distribution has a great impact on the performances of the two greedy algorithms.

- *Sampling-based query selection*: before crawling the deep web, there is no input to the set covering algorithm, i.e., neither documents nor terms are available. To bootstrap the process, a sample of the data source is needed so that some good terms are selected and sent, and more documents are returned and added to the input of the set covering problem. This repetitive and incremental process is used in the methods proposed by [17, 7]. The disadvantage is the requirement of downloading and analyzing all the documents covered by current queries to select the next query to be issued, which is highly inefficient.

  We found that instead of incrementally running and selecting queries many times, selecting all queries on a fixed sample at once is more efficient and the result can be better as well. We first collect from the data source a set of documents as a sample that represents the original data source. From the sample data, we select a set of queries that cover most of the sample documents with a low cost. Then we use this set of queries to extract data from the original data source. Finally, we show that the queries working well on the sample will also induce satisfactory results on the original data source. More precisely, the vocabulary learnt from the sample can cover most of the original data source; the cost of our method in the sample can be projected to the original data source; the size of the sample and the number of candidate queries for our method do not need to be very large.

- *Query selection for ranked data sources*: many large data sources rank the matched documents and return only the top k documents, where k may be rather small ranging from 10 to 1000. For such ranked data sources, popular queries will return only the documents in the top layers of the data. Documents ranked low may never be returned if only popular terms are used. In order to crawl such data sources, we need to select the queries of appropriate size, in particular the small queries.

  The crucial challenge is to obtain the small queries and learn the frequencies

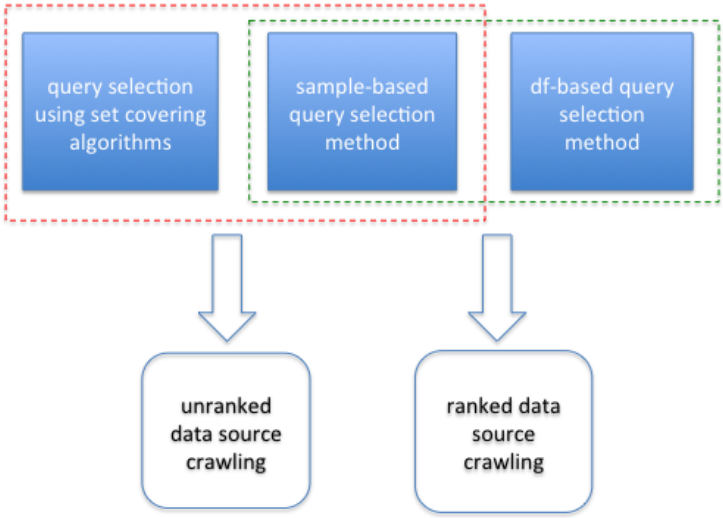

Figure 1.1: The key components in this dissertation for unranked and ranked deep web data sources.

of the small queries from a sample. In particular, the conventional Maximum Likelihood Estimation [27] overestimates the rare queries. We use Simple Good-Turing estimation [28] to correct the bias. Another estimation method is based on Zipf's law [29]. After a few probing queries issued for their frequencies, the method tries to propagate the known frequencies to nearby terms with similar frequencies.

The experimental result shows that crawling ranked data sources with small queries is better than using random queries from a middle-sized Webster English dictionary, and much better than using popular terms as queries whose frequencies are more than $2k$.

Figure 1.1 shows the key components in our query selection technique for unranked and ranked deep web data sources.

In this dissertation, we focus on textual data sources, i.e., the data sources that

contain plain text documents only. These kinds of data sources usually provide a single keyword-based query interface instead of multiple attributes, as studied by Wu et al. [20]. Madhavan et al.'s study [7] shows that the vast majority of the HTML forms found by Google deep web crawler contain only one search attribute, thus we focus on such a search interface.

The rest of this dissertation is organized as follows: Chapter 2 introduces the related work; in Chapter 3, we convert the query selection problem into set covering problem and propose a weighted greedy algorithm to addressing it; Chapter 4 details our sampling-based crawling method; Chapter 5 shows our novel df-based crawling method for crawling ranked deep web data sources; and finally Chapter 6 concludes this dissertation with important remarks.

Chapter 2

Related work

In this chapter, we present a detailed introduction to the related work of our research.

2.1 Query selection problem

The query selection problem is to select a set of queries to efficiently harvest a specific deep web data source. There are many existing works addressing this problem [7, 15, 16, 17, 18, 19, 20].

A primitive solution could be randomly selecting some words from a dictionary. However, this solution is not efficient because a large number of rare queries may not match any page, or there could be many overlapping returns. Recently, instead of selecting queries from a dictionary, several algorithms [17, 7, 15, 20, 16] have been developed to select queries from the downloaded documents, which were retrieved by previous queries submitted to the target deep web data source.

Generally speaking, query selection methods can be categorized as: the methods for unstructured data sources and the methods for structured data source.

2.1.1 Unstructured data sources

Here unstructured data sources refer to textual data sources that contain plain text documents only. This kind of data sources usually provide a single keyword-based query interface.

Ntoulas et al. [17] proposed an incremental adaptive method for unstructured

data sources. An incremental method selects queries from the downloaded documents and the number of documents increases as more queries are sent. Their method selects the query returning most new documents per unit cost iteratively 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.

They proposed two approaches to estimate. The first method, called the *independence* estimator, assumes that the probability of a term in the downloaded documents is equal to that in DB. Based on the document frequency of a term in the downloaded documents $N(q_j|subset\ collection)$, the method can estimate how many documents containing the term in the original data source $\widehat{N}(q_j)$. Then we can estimate the number of new returned documents by the equation $\widehat{N}_{new}(q_j) = \widehat{N}(q_j) - N(q_j|subset\ collection)$. The second estimation method is the Zipf-law-based estimator provided in [29] (the detail of this method will be introduced in later section), it estimates the actual document frequency of terms inside the document collection by following Zipf's law [30].

They compared their adaptive method with two other query selection methods: the random method (queries are randomly selected from a dictionary), and the generic-frequency method (queries are selected from a 5.5-million-web-page corpus based on their decreasing frequencies). The experimental result shows that the adaptive method performs remarkably well in all cases. However, their method selects queries from an incremental document collection and it means that the method needs to analyze each document once it is downloaded and to estimate the document frequency for every term inside it at each round. In order to estimate document frequency efficiently, the solution from [17] computes the document frequency of each term by updating the query statistics table after more documents are downloaded. But maintaining this table is still difficult.

Madhavan et al. [7] developed another incremental method for unstructured 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 [17].

Their query selection policy is based on TF-IDF that is the popular measure in the information retrieval. TF-IDF measures the importance of a word by the formula below.

$$tfidf(w,p) = tf(w,p) \times idf(w) \tag{2.1}$$

$tf(w,p)$ is the *term frequency* of the word w in any page p, and measures the importance of the word w in any page p.

$$tf(w,p) = n_{w,p}/N_p, \tag{2.2}$$

where $n_{w,p}$ represents the number of times the word w occurs in any page p and N_p is the total number of the terms in any page p.

$idf(w)$ (*inverse document frequency*) measures the importance of the word w among all the web pages, and is calculated by $\log(\frac{D}{d_w})$ where D is the total number of web pages and d_w is the number of web pages where the term w appears.

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:

- 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;
- 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.

Additionally, their approach emphasizes the breadth-oriented crawling that is quite different to prior researches. They observed the statistical data on Google.com and found that the results returned to users were more dependent on the number of the deep web data sources. They analyzed 10 millions of deep web data sources. They discovered that the top 10,000 deep web data sources accounted for 50% of the deep web results, while even the top 100,000 deep web data sources only accounted for 85%. This observation prompted them to their focus on crawling as many deep web data sources as possible, rather than surfacing on a specific deep web data source.

In [16], Barbosa et al. proposed a sampling-based method to siphon 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, ..., q_{10}) are selected, the final issued disjunctive query is $q_1 \vee \ldots \vee q_{10}$.

The method is easy to implement but its shortcoming is obvious. Most of websites limit the number of the subqueries of a Boolean query and the number of returned documents for a query. Thus using only one Boolean query to crawl a deep web data source could be hard.

2.1.2 Structured data source

A structured data source is a traditional relational database and all contents are stored as records in tables. Usually the deep web site with a structured data source provides a search interface with multiple attributes instead of a single keyword-based search bar.

Wu et al. [20] presented a graph-based crawling method to retrieve the data inside a specific deep web data source. In their method, a structured data source DB is viewed as a single relational table with n data records $\{t_1, \ldots, t_n\}$ over a set of m attributes $\{attr_1, \ldots, attr_m\}$. All distinct attribute values are contained by the Distinct Attribute Value set (DAV).

Based on a data source DB, an *attribute-value* undirected graph (AVG) can be constructed. Each vertex v_i represents a distinct attribute value $av_i \in DAV$ and each edge (v_i, v_j) stands for the coexist of the two attribute values av_i and av_j in a record t_k.

According to AVG, the process of crawling is transformed into a graph traversal in which the crawler starts with a set of seed vertices and at each iteration a previously seen vertex v is selected to visit, thus all directly-connected new vertices and the records containing them are discovered and stored for future visits.

Raghavan et al. [15] proposed a task-specific human-assisted crawling method for structured data sources. First, it needs a task-specific database D which contains enough terms for the crawler to formulate search queries relevant to a particular task. For example, to retrieve documents about semiconductor industry in a data source DB, D should contain the lists of semiconductor companies and product names that are of interest. For the query selection strategy, the authors assume that all attributes are independent from each other, and all records can be retrieved by using the Cartesian product of all possible attribute values from the task-specific database D.

Namely, for each attribute E_i shown in the multi-attribute search form, a matching function is used to assign a value shown in Equation 2.3.

$$Match((\{E_1, \ldots, E_m\}, S, M), D) = [E_1 = v_1, \ldots, E_m = v_m] \tag{2.3}$$

where $\{E_1, \ldots, E_m\}$ is a set of m form attributes, S is the submission information associated with the form (e.g., submission URL) and M is the meta infor-

mation about the form (e.g., the URL of the form page). A value assignment $[E_1 = v_1, \ldots, E_m = v_m]$ associates value v_i with form attribute E_i and the crawler uses each possible value assignment to 'fill-out' and submit the completed form. This process is repeated until the set of value assignments is exhausted.

In Raghavan et al.'s method, the product of all possible attribute values could be a large number and sometimes it means a high communication cost and an empty return result, which could significantly reduce the effectiveness and efficiency of the crawler.

2.2 Automated form filling

Before the query generation process, the problem of how to automatically interact with the search interface has to be addressed firstly.

For the deep web crawling, the automated form filling is simpler than other techniques such as the virtual data integration [31, 32, 33] and the web information extraction [34, 35, 19, 36], which need to deal with semantic inputs based on a given knowledge. Here the technique of form filling used by Google's deep web crawling [7] is presented as an example.

In Google's deep web crawling [7], they only focuses on two kinds of inputs in a searchable form. One is the selection input that offers users some imposed items to choose, such as the select menu, the radio button and the check box; the other one is the free input, e.g., the text box. A selection input often provides a default item, if it is not selected as a certain value, the effect of the input for search results can be ignored. Among the selection inputs, there exists a special ordering input which arranges the returned results according to an order. A crawler has to distinguish the ordering input from other kinds of selection inputs because such the ordering input causes a much overlapping retrieval. The free input could be anywhere, usually called a search bar. It accepts any keyword (or Boolean query) and returns documents containing the keyword. Figure 2.1 shows an example of the two kinds of inputs from the website of Amazon.com. In the figure, two selection inputs (drop-down menus) are on the left and right sides respectively. The right one is an ordering input, and a free input (text box) lies in the center.

According to the values of the free and selection inputs, the retrieved documents

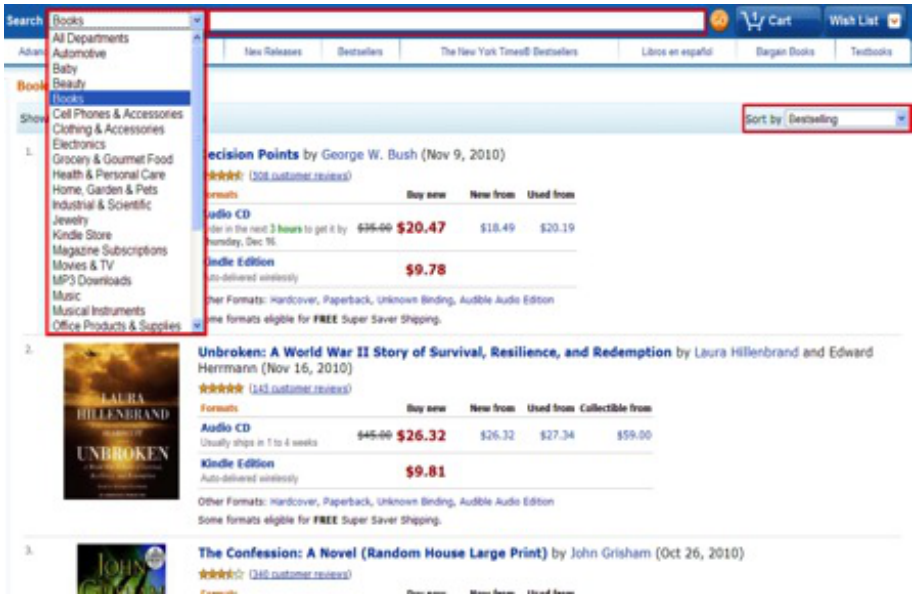

Figure 2.1: An example from Amazon.com website

can be described as following:

$$select\ K\ from\ DB\ where\ P\ ordered\ by\ O.$$

where $K = (keyword_1, ..., keyword_n)$, $P = (constrain_1 = c_1, ..., constrain_m = c_m)$, and $O = order_i$. Each keyword is from a free input, each constrain is set by the selection input one by one and the return order is decided by the value of the order selection input.

Recognizing free inputs and selection inputs, it can be done easily by analyzing the HTML code of the form pages. In [7], the free input of a search form is automatically filled out by selecting queries from the query list derived from the retrieved document collection. Each selection input and its value are decided by the *informativeness test* that is used to evaluate *query template*, i.e., the combinations of the selection inputs. The basic idea of the informativeness test is that all templates (the searchable form with different values for each selection input) are probed to check which can return sufficiently distinct documents. A template that returns enough distinct documents is deemed a good candidate for crawling.

2.3 Automated searchable form discovery

Before crawling the deep web data sources, first of all, we have to locate the entrances (searchable forms) of the targeted data sources. It is a challenging problem due to several reasons. Firstly, the contents in the Web keep changing at all time; it is continually updating the routine, adding new websites, and removing old ones. In addition, the entrances of these data sources are sparsely distributed on the Web. Thus, to acquire a few deep website locations, a surface web crawler needs to search tons of useless web pages and this could consume the resources of the crawler too much.

Note that a crawler for the surface web is quite different from the one for the deep web. The former usually starts at some seed web pages and follows all the outgoing hyperlinks of the visited web pages to traverse the linked pages, and this iterative process ends when all the reachable web pages are visited or the stopping criteria are satisfied. The latter works on the many searchable forms of the backend data sources by issuing queries and retrieving all the returned documents. The crawler for locating the entrances of deep web data sources is the one for the surface web.

At the beginning, an exhaustive crawler is used to crawl all the web pages it can reach to find the entrances (such as the crawlers Alta Vista, Scooter, and Inktomi). The whole process could take a few of weeks up to months. However, owing to the rapid increase of the Web, it becomes harder and harder to implement.

In common, users prefer a collection of searchable forms that can return high relevant documents. Hence, some researchers [10, 37] provided the concept of focused crawler that it is a topic-driven crawler. A focused crawler tries to retrieve a subset of web pages from the surface web that are highly relevant to a topic so that related searchable forms are likely contained in this collection.

In [10], the surface web is considered a directed graph G given a predefined hierarchy tree-structured classification C on topics, such as Yahoo!. Each node $c \in C$ refers to some web pages in G as examples denoted as $D(c)$. A user's interest can be characterized by a subset of topics $C^* \subset C$. Given any web page p, a method is used to calculate the relevant value $R_{C^*(p)}$ for p with respecting to C^*. Based on the relevant value of p , its neighbour web pages (directly linked by p) firstly estimates relevant values, and, after crawled, the estimated relevant values of the neighbour pages are updated to relevant values.

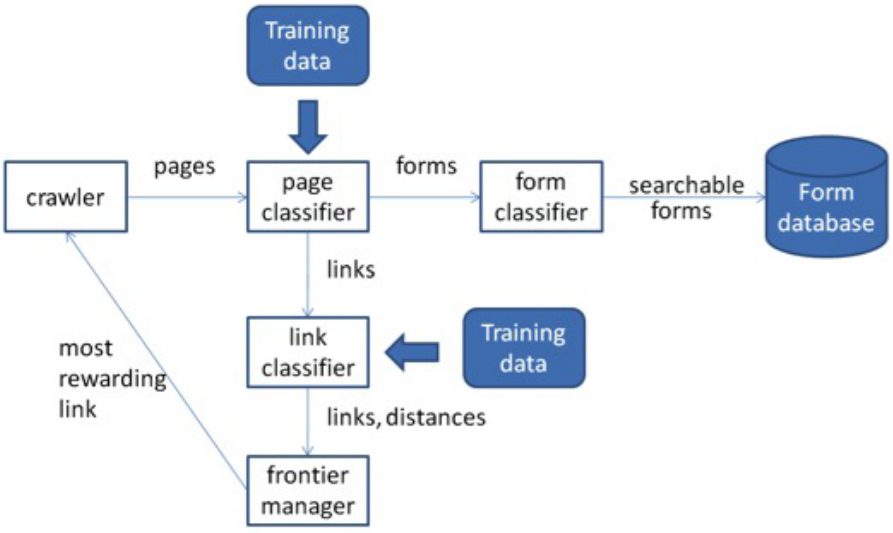

Figure 2.2: The framework of form-focused crawling method

The crawling process starts from example web pages $D(C^*)$. At each iteration, the crawler inspects the current set V of the visited web pages and then selects an unvisited web pages corresponding to the hyperlinks in one or more visited web pages by their estimated relevant values.

Focused crawlers can significantly reduce the number of useless web pages crawled but the ratio between the number of forms and the number of visited pages is still low. In a recent search work in [11], the authors provide a Form-Focused Crawling(FFC) method to seek the searchable forms based on topics. The difference is that the previous focused crawler focuses the search process based solely on the contents of the retrieved pages, and it is not enough. FFC combines the techniques of the previous focused crawlers with a link classifier that analyzes and prioritizes the links that will likely lead to searchable forms in one or more steps. Figure 2.2 shows the framework of the form-focused crawling method.

In this figure, there are four major parts in the framework: page classifier, link classifier, frontier manager, and form classifier. The page classifier is trained to identify the topic of the crawled web pages and it uses this strategy in [10]. The searchable form classifier is used to filter out non-searchable forms.

The link classifier and the frontier manager are highlight parts. The authors first make use of a backward search method to analyze and prioritize links which

are likely to lead to a searchable form in one or more steps. The backward search method starts with a given set of URLs of web pages that contain forms in a specified domain. Links to these pages are acquired by crawling backward from these pages based on the facilities from standard search engines, such as Google and Yahoo! [38]. The process of backward crawling is breadth-oriented, and all the retrieved documents in level $l+1$ are linked to at least one of the retrieved documents in level l. At each iteration, the best features of the links in the corresponding level are manually collected. Finally, those features are used to train the link classifier to estimate the steps from a given link to a searchable form.

In the frontier manager, there are multiple priority queues for the links that are not visited, and each queue corresponds to a certain estimated step given by the link classifier. It means that a link l is placed in the queue i if the link l has i estimated steps to the target searchable form. At each crawling step, the manager selects the link with the maximum reward value as the next crawling target, and the reward for a link is decided by the current status of the crawler and the priority of the link.

There are still a few of other methods used to locate searchable forms (e.g., IP sampling method). However, as far as we know, they do not address this problem well.

2.4 Sampling techniques

Query selection is based on various properties of the data source, such as the df of all the terms. However, such resource descriptions are not provided by most of deep web sites. They need to be leant by sample. To select appropriate queries for crawling, random sampling methods [1, 39, 40] become an important way to acquire the resource description of a target data source and such information usually is the basis of most of query selection methods.

In [1], the authors provided a query-based sampling method for textual deep web data sources and showed that their method can efficiently acquire the accurate term frequency list of the target data source based on a sufficiently unbiased sample.

Their query-based sampling algorithm is shown as follows:

1. Randomly select an initial query that can retrieve a nonempty set of documents;

Table 2.1: The consistency between the conclusions of Callan's work [1] and our sampling-based query selection method

No.	Callan's conclusion	Our conclusion
1.	sample can contain most of the common terms in the original DB	selected queries from sample can cover most of the original DB.
2.	the rank list of terms in the sample can represent the order of the corresponding terms in the original DB in a way	selected queries from the sample can cover most of the original DB with low cost.
3.	small sample can capture most of the vocabulary and the rank information of the original DB	to harvest most of the original DB, the sample size and the query pool size do not need to be very large.

2. Send the selected query to the target data source and retrieve the top k documents;

3. Update the resource description based on the characteristics of the retrieved documents (extract terms and their frequencies from the top k documents and add them to the learned resource description);

4. If the stopping criterion has not yet been reached, select a new query and go to Step 2

The above algorithm more looks more like a framework and some parameters need to be decided. What is the best value for N? How to select an initial query? How to select a term as query for further retrieval? What is the stopping criterion? All those questions are answered in [1].

The conclusions of the Callan's sampling method support our sampling-based query selection method in Chapter 4 and they are indirectly verified by our results. The corresponding relationship is shown in Table 2.1.

In [40], the authors presented a Boolean query-based sampling method for the sampling of uniform documents from a search engine's corpus. The algorithm formulates "random" queries by using disjunctive and conjunctive Boolean queries to pick uniformly chosen documents from returned results. The method needs the availability of a lexicon of terms that appear in the Web documents. Each term contained by the Boolean queries in the lexicon should be given an estimate of its

frequency on the Web. The lexicon is generated in a preprocess by crawling a large corpus of documents from the Web.

The method [40] is somehow the reverse process of the deep web crawling. For the crawling process, a random sample is needed first and then it begins based on the terms inside the sample, but this method first requires the crawling results that help to do the random sampling.

In [39], the authors proposed two elaborate random sampling methods for a search engine corpus. One is the lexicon-based method and the other is the random walk method. Both methods produce biased samples and each sample is given a weight to represent its bias. Finally, all samples in conjunction with the weights, applied using stochastic simulation methods, are considered a near-uniform sample. Compared to the methods in [1, 40], this method performs better but its complexity is much higher than the other two.

Chapter 3

Query selection using set covering algorithms

3.1 Introduction

In this and the next chapter, we discuss the query selection problem for deep web data sources without a return limit, i.e., all documents matched by a query should be returned. We focus on such data source first because

- there are many such data sources in the Web, especially the websites for public services, such as the PubMed website [41] or the website of United States Patent and Trademark Office (USPTO) [42];
- the strategies of the query selection for data sources without a return limit can provide an insight to the one with a return limit (we will discuss the latter in Chapter 5).

Let the set of documents in a data source be the universe. If one term is contained by one document, we say that the term *covers* this document. The query selection problem is to find a set of terms as queries which can jointly cover the universe with a minimal cost. Thus, it is cast as a set covering problem (SCP).

SCP is an NP-hard problem [43] and has been extensively studied by many researchers [21, 22, 23, 24] in fields such as scheduling problem, routing problem, flexible manufacturing and so on.

Many algorithms have been developed for set covering problem. Some of them, such as [44, 45, 46], can provide better solutions in general but require more resources for the execution. For example, since the Optimization Toolbox for binary integer programming problems provided by Matlab can only work within 1G memory limit [47], it is easy to be out of memory for one thousand by thousand input matrix.

Based on the above consideration, the greedy method is a better choice because it usually leads to one of the most practical and efficient set covering algorithms. But we found that, so far, most of the research work on greedy methods have been carried out on the normally distributed data and the corresponding results are acceptable compared with the optimal solutions (or the best known solutions). In deep web crawling, the *degrees of the documents* are not distributed normally. Instead, they follows a lognormal distribution. For data with a lognormal distribution, the results of the greedy method could be improved.

We have developed a weighted greedy set covering algorithm. Unlike the greedy method, it introduces weights to the greedy strategy. We differentiate among documents according to the dispersion of document degree caused by the lognormal distribution. A document with a smaller document degree is given a higher document weight. A document with a higher weight should usually be retrieved earlier since it will lower the total cost in the future. This is combined with the existing greedy strategy. Our experiment carried out on a variety of corpora shows that the new method outperforms the greedy method when applied on data with a lognormal distribution.

After analyzing the greedy and our weighted greedy methods on data with various distributions, we further argue that the data distribution plays a great role in the performances of the two greedy algorithms.

3.2 Set covering

The query selection problem is to find a set of terms as queries which can jointly cover the universe with a minimal cost. The query cost is defined by a number of factors, including the cost for submitting the query, retrieving the URLs of matched documents from resulting pages, and downloading actual documents. There is no need to consider separately the first two factors when measuring the cost. For

a given query, there could be many matched documents and they are returned in the resulting pages instead of in a long list. Thus the number of queries is proportional to the total number of retrieved documents because the same query needs to be sent out repeatedly to retrieve the subsequent pages. For example, if there are 1,000 matches, the data source may return one page that consists of only 10 documents. If you want the next 10 documents, a second query needs to be sent. Hence, to retrieve all the 1,000 matches, altogether 100 queries with the same term are required.

The cost of downloading actual documents should be separated from the cost in the query selection problem. Since no one will repeatedly download redundant documents, the cost of downloading all the documents of a data source is a constant. Thus it does not help to find out a set of appropriate queries to submit by measuring their downloading cost.

In this setting, we argue that the cost of retrieving the URLs of matched documents is the cost to consider for the query selection problem, and it can be represented by the total sum of the document frequencies of the selected queries.

Given a set of documents $D = \{d_1, ..., d_m\}$ and a set of terms $QP = \{q_1, ..., q_n\}$, their relationship can be represented by the well known document-term Matrix $A = (a_{ij})$ where $a_{ij} = 1$ if the document d_i contains the term q_j; otherwise $a_{ij} = 0$.

The query selection problem can be modeled as set covering problem defined in [43].

Definition 1 ***(SCP)*** *Given an $m \times n$ binary matrix $A = (a_{ij})$, let $C = (c_1, \ldots, c_n)$ be a non-negative n-vector and each $c_j = \sum_{i=1}^{m} a_{ij}$ represents the cost of the column j. SCP calls for a binary n-vector $X = (x_1, \ldots, x_n)$ that satisfies the objective function*

$$Z = min \sum_{j=1}^{n} c_j x_j. \tag{3.1}$$

Subject to

$$\sum_{j=1}^{n} a_{ij} x_j \geq 1, \qquad (1 \leq i \leq m). \tag{3.2}$$

$$x_j \in \{0, 1\}, \qquad (1 \leq j \leq n). \tag{3.3}$$

In the matrix representation of the data source DB, each column represents a

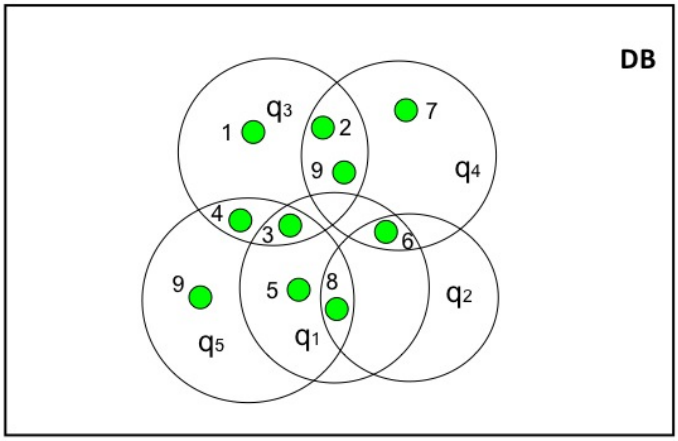

Figure 3.1: The illustration of the textual data source DB in Example 1. Each dot or circle represents one document or term, respectively.

term in the query pool QP and each row represents a document of DB. $c_j = \sum_{i=1}^{m} a_{ij}$ is the document frequency (*df* for short) of the term q_j, which is equal to the number of documents containing the term. We should be aware that the terms in the QP of a data source are usually parts of all the terms inside the data source; otherwise, there could be more than millions of terms in the QP and it would be out of the capability of most set covering algorithms.

Example 1 *Given a deep web textual data source* $DB = \{d_1, ..., d_9\}$ *shown in Figure 3.1, there are 5 terms* ($QueryPool = \{q_1, ..., q_5\}$) *and each is contained in at least one of the 9 documents. For example,* d_1 *contains* q_3 *only, and* d_2 *contains* $q3$ *and* q_4*. The doc-term matrix representation of the data source is shown in Table 3.1. The optimal solution of SCP here is* $\{q_4, q_3, q_1\}$ (*with corresponding* $X = (1, 0, 1, 1, 0)$) *and its cost is 13 (4+5+4).*

Table 3.1: The doc-term Matrix A in Example 1

	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
df	4	2	5	4	5

3.3 Greedy algorithm

The basic idea of the greedy algorithm [48] is to construct a solution in a step-by-step manner and to approximate a global optimal solution by choosing a *locally optimal* solution on each step. For a set covering problem, the solution is constructed step-by-step: on each step, one column is selected as a part of solution until the requirement is reached.

There are various ways to select a column.

- *Minimize cost*: we can select the next column q_u which has the lowest cost on this step, namely, $c_u = min(c_j)$ where $1 \leq j \leq n$ and $x_j = 0$. The lowest total cost $\sum_{j=1}^{n} c_j x_j$ is approximated by having the smallest cost on each step;
- *Maximize coverage*: another popular way is to select the next column q_u which can cover the largest number of rows that are not yet covered by the previously selected columns, namely, setting $x_u = 1$ to maximize $\sum_{i=1}^{m}((1-y_i) \times a_{iu})$ where $y_i \in \{0, 1\}$ and $y_i = 1$ if d_i have been covered, otherwise, $y_i = 0$. Such a local optimization aims at reaching the expected coverage in fewer steps so that the number of selected columns can be kept small. In this way, the total cost $\sum_{j=1}^{n} c_j x_j$ can get close to the smallest. This approach is especially suitable when all columns in Matrix A have the same cost, and the total cost is purely determined by the number of selected columns.

Of course, we can also combine the above two approaches: the next column q_u

is selected taking into account both its cost (c_u) and the number of new rows that can be covered ($\sum_{i=1}^{m}((1-y_i)\times a_{iu})$). One of the possible combinations is described as Algorithm 1 based on Definition 1. In the following, we use *new/cost* to denote the value for local query selection. In this algorithm, $y_i = 1$ indicates that the i-th

Algorithm 1: Greedy algorithm.

Input: an $m \times n$ matrix $A = (a_{ij})$
Output: a solution n-vector $X = (x_1, \ldots, x_n)$
Process:

1 $c_j = \sum_{i=1}^{m} a_{ij}$; $x_j = 0 (1 \le j \le n)$; $y_i = 0 (1 \le i \le m)$;
2 while($\sum_{i=1}^{m} y_i < m$){
3 find a column q_u which maximizes $\sum_{i=1}^{m}((1-y_i)\times a_{iu})/c_u$;
4 $x_u = 1$; $y_i = 1$ if $\sum_{j=1}^{n} a_{ij}x_j \ge 1$;
5 }

document is already covered.

Ostensibly, the greedy strategy is faultless but there are still some problems we need to discuss and such problems may lead to an insight to the potential shortage.

3.4 Introducing weight to greedy algorithm

With the step-by-step manner, different rows are covered in different steps. Is there any difference to covering a row earlier or later? In the above greedy strategy, all newly covered rows are always considered as having a unit cost and there is no difference whether they are covered earlier or later.

In Table 3.1, the rows d_1 and d_7 are only covered by the columns q_3 and q_4 respectively. Such documents should be covered as early as possible. To easily compare, let's say that q_4 is set to the initially selected column and then the unique solution from the greedy algorithm is $\{q_4, q_5, q_3\}$ with the cost 14. Actually, the optimal solution is $\{q_4, q_3, q_1\}$ with the cost 13, and this optimal solution can be reached if q_3 or q_4 is in the initially selected column. The greedy method failed to

find an optimal solution in this case is not to consider covering d_1 and d_7 by using q_3 and q_4 as early as possible.

Now that we know there is a difference in covering certain rows earlier or later, the second question is how to measure such difference to cover a row earlier or later. For each row i, we argue that if the number of the columns covering row i is bigger, it is better to be covered later. Here are two reasons:

- When row i is covered at high coverage (in later steps) and most of the rows are already covered, more columns covering row i mean that there could be more possibilities to select a small-cost column which covers few new rows (of course, at high coverage, no column can take many new rows);
- When row i is covered at low coverage (in earlier steps) and most of the rows are not covered yet, more columns covering row i mean that there are more possibilities to cause overlapped coverage, i.e., row i will be covered many times.

For each document, we call the number of the terms in the query pool QP covering it the *degree of the document*. Based on the above intuition, a higher degree of a document means that the document needs to be covered later. It is defined as follows:

Definition 2 ***(document degree)*** *The degree of a document (element) d_i in DB with respect to QP, denoted by $deg(i)$, is the number of different terms in QP that occur in the document d_i, i.e.,*

$$deg(i) = \sum_{j=1}^{n} a_{ij}. \tag{3.4}$$

Then we define $dw(i)$ as *document weight* by using the document degree as follows:

Definition 3 ***(document weight)*** *The weight of a document d_i in DB with respect to QP, denoted by $dw(i)$ (or dw for short), is the inverse of its document degree, i.e.,*

$$dw(i) = \frac{1}{deg(i)}. \tag{3.5}$$

Intuitively, the fewer the terms (columns) are contained in document d_i (row i), the larger the weight that is given to it.

After all, covering rows is implemented by selecting columns one by one. At each step, the column covering the rows with larger weights should be selected earlier. Based on the definition of the document weight, for each term (column) q_j at each step, we sum up all weights of uncovered documents (rows) covered by it and obtain the *query weight qw* for the term (column) q_j. The terms (columns) with larger weights should be selected as early as possible.

Definition 4 ***(query weight)*** *The weight of a query q_j $(1 \leq j \leq n)$ in QP with respect to DB, denoted by $qw(j)$ (or qw for short), is the sum of the document weights of all the uncovered documents containing the term q_j, i.e.,*

$$qw(j) = \sum_{i=1}^{m} (a_{ij} \times dw(i) \times (1 - y_i)), \tag{3.6}$$

where (a_{ij}) is the corresponding Matrix A and $y_i = 1$ if d_i has been covered, otherwise, $y_i = 0$.

3.4.1 Weighted algorithm

We consider *qw(j)* a better measurement than *new(j)* as it combines both the information about how many new documents can be obtained by selecting query q_j and the information about how soon the newly obtained documents should be considered to be covered. Consequently, we use *qw/cost* to replace *new/cost* for query selection. We call the corresponding algorithm *weighted greedy algorithm*.

Based on the definitions of the document weight and the query weight, we present our weighted greedy algorithm shown in Algorithm 2.

Example 2 *Based on the matrix in Table 3.1, the initial weights of the documents and the queries are shown in Table 3.2. In the first round of Algorithm 2, row q_4 has the maximum value of qw/cost (0.54). It is selected as the first query, hence the corresponding x_4 is set to 1. For convenient to explanation, the column q_4 and the covered rows, i.e., d_2, d_6, d_7 and d_9 are removed from the matrix A, and the resulting corresponding weighted matrix is shown in Table 3.3. In the second and*

Algorithm 2: Weighted greedy algorithm.

Input: an $m \times n$ matrix $A = (a_{ij})$
Output: a solution n-vector $X = (x_1, ..., x_n)$
Process:

1 $c_j = \sum_{i=1}^{m} a_{ij}$; $x_j = 0 (1 \leq j \leq n)$; $y_i = 0 (1 \leq i \leq m)$;
2 while$(\sum_{i=1}^{m} y_i < m)\{$
3 find a column q_u that maximizes $qw(u)/c_u$;
4 $x_u = 1$; $y_i = 1$ if $\sum_{j=1}^{n} a_{ij} x_j \geq 1$;
5 $\}$

Table 3.2: The initial weight table of Example 2 corresponding to Matrix A

	t_1	t_2	t_3	t_4	t_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
$cost(df)$	4	2	5	4	5
qw	1.49	0.66	2.66	2.16	1.99
$qw/cost$	0.37	0.33	0.53	0.54	0.39

third rounds, q_3 and q_1 are selected respectively and the solution of the weighted greedy is $X = (1, 0, 1, 1, 0)$, and its cost is 13(4+5+4).

3.4.2 Redundancy removal

A solution generated from the above two methods could usually contain redundancy, i.e., although some columns from the solution are removed, it can still cover all the rows of matrix A. Here, if a solution does not contain any redundancy, it is a *prime* solution. To extract a prime solution from the solution X, we use the standard procedure described in [49] and it is shown in Algorithm 3. In Algorithm 3, each selected column ($x_j = 1$) of the solution X is considered to be in order, and if

Table 3.3: The second-round weight table of Example 2

	t_1	t_2	t_3	t_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
$cost(df)$	4	2	5	5
qw	1.16	0.33	1.83	1.66
$qw/cost$	0.29	0.165	0.36	0.33

X without it is still a solution, the corresponding x_j is set to 0. When all the selected columns of X have been considered, the derived solution X becomes a prime solution.

Algorithm 3: Redundancy removal algorithm.

Input: an $m \times n$ matrix $A = (a_{ij})$, a solution n-vector $X = (x_1, ..., x_n)$
Output: a prime solution $X = (x_1, ..., x_n)$
Process:
1 foreach $x_u = 1$ $(1 \leq u \leq n)$
2 set $x_u = 0$ if $\forall i$, $(\sum_{j=1}^{u-1} a_{ij}x_j + \sum_{j=u+1}^{n} a_{ij}x_j) \geq 1$;

3.5 Experiments and analysis

3.5.1 Experimental setup

The purpose of the experiment is to test whether the weighted greedy is better than the greedy algorithm in terms of the solution results. Note that the performances of the two algorithms are similar since they are both greedy algorithms.

On the same input matrix, the greedy algorithm may produce different solutions because in each step, especially in the initial stage, there are ties to select from. In our implementation, we randomly selected one of the candidates when ties occurred. Therefore, the solution values of the same input fluctuate with each run.

We run each algorithm on each input data 100 times and record the statistics of the 100 results, such as the maximum cost (max), the minimal cost (min), the

average cost (avg) and the standard deviation of the costs (SD). More specifically, given the results $C_1, C_2, \ldots, C_{100}$, the average and standard deviation of the results are defined as follows:

$$SD = \sqrt{\frac{\sum_{i=1}^{n}(C_i - avg)^2}{n}},$$

where $avg = \frac{\sum_{i=1}^{n} c_j}{n}$ and $n = 100$.

The improvement of the weighted greedy algorithm can be calculated by using the formula

$$IMP = \frac{C_g - C_w}{C_g}.$$

where C_g and C_w are the results from the greedy and weighted greedy algorithms with the same characteristics, such as the average, maximum, and minimal results. For example, for calculating the average improvement, C_g and C_w should be the average costs from the greedy and weighted greedy algorithms.

3.5.2 Data

The experiment was carried out on two different sets of data. One is derived from the deep web crawling problem, and the other is from the Beasley data that are the standard benchmarks for set covering problems [26].

For the data in deep web crawling problem, we experimented with four data collections that cover a variety of forms of web data, including

- regular web pages (web pages under .gov domain),
- web articles (articles in wikipedia.com),
- newspaper articles (from Reuters),
- newsgroup posts in newsgroups.

We experimented with these data in order to show that our algorithm performs consistently across domains, and is independent of the forms of the web documents.

Although each data collection is huge, in the size of millions or more, we report only the results on subsets of the documents that contain ten thousand documents. We have experimented with some larger data size and observed similar results. For

practical consideration, we restrict the row size to ten thousand so that we can run numerous tests.

The columns or the queries are not so obvious to select. First we filter out very popular queries and queries that occur in only one document in order to avoid trivial solutions. For instance, if all the queries occur only once in all documents, it is trivial to find an optimal solution if there is one. We select the remaining terms randomly until $\mu = 20$, where μ is the average document degree defined in equation 3.7. The reason to set μ as 20 is that according to random graph theory [50], all the nodes are most surely connected when the average degree is $\ln(m)$ where m is the number of documents.

$$\mu = \frac{\sum_{i=1}^{m} deg(i)}{m}. \tag{3.7}$$

The distribution of the document degrees of the four data sources is shown in Figure 3.2 by using histogram on log-log scales. For each histogram, the x axis is for the document degree and the y axis is for the number of documents with a certain document degree. Each circle represents the number of documents whose degrees are in a specified interval (bin) at x axis. Since the distribution is very skewed, we use log-log format for x and y axes. To avoid the noise on the tail of each histogram, we use logarithmic binning to vary the bin width and it increases in 2^i $(i = 1, 2, 3, \ldots)$. From the figure, we can see that the distribution of the document degrees for our data follows the *lognormal distribution* [51].

The details of the corresponding matrices of the four data sources are summarised in Table 3.4 where the *Coefficient of Variation*(CV) is used to measure the dispersion of document degree (deg), which is the ratio of the standard deviation to the mean. Given an $m \times n$ matrix A, we apply the definition of CV in the following way to measure the dispersion of the document degrees:

$$CV = \frac{1}{\mu}\sqrt{\frac{\sum_{i=1}^{m}(deg(i) - \mu)^2}{m}}, \tag{3.8}$$

Here the two algorithms were tested on 50 standard test problems as well, which are called the Beasley data (J. E. Beasley is the first to produce those data). They are from the OR-Library and available electronically [26]. All the problems in the OR-library are used as standard data by researchers in Operations Research domain.

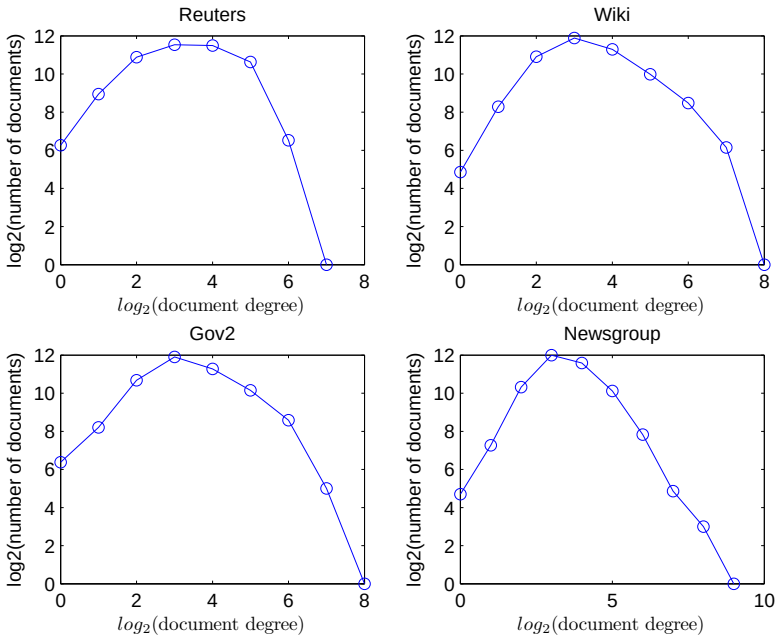

Figure 3.2: The distribution of the document degrees using 'logarithmic binning' in log-log scale for our data sources. The bin size of the data increases in 2^i ($i = 1, 2, \ldots$)

Table 3.4: The properties of our data sources (avg(df): the average of document frequencies, max: the maximum document degree, min: the minimal document degree, avg(CV): the average of CVs, SD(CV): the standard deviation of CVs, number: the number of data sources.)

Data source	Row(m)	Column(n)	avg(df)	document degree			
				max	min	μ	CV
Reuters	9,990	5707	32.1	136	1	19.3	0.77
Wikipedia	9,989	7213	26.74	239	1	19.3	1.09
Gov2	9,885	4969	38.77	176	1	19.5	0.99
Newsgroup	9,988	5954	32.1	331	1	19.2	0.97

Table 3.5: The properties of the Beasley data

Data sets	Row(m)	Column(n)	avg(df)	document degree					
				max	min	μ	CV		
							avg(CV)	SD(CV)	number
4	200	1000	3.97	36	8	19.8	0.214	0.02	10
5	200	2000	3.97	60	21	39.7	0.155	0.01	10
6	200	1000	9.89	71	29	49.5	0.135	0.008	5
a	300	3000	6.02	81	38	60.0	0.118	0.001	5
b	300	3000	15.0	192	114	149.6	0.079	0.0003	5
c	400	4000	7.99	105	56	79.9	0.107	0.0005	5
d	400	4000	20.02	244	159	200.2	0.068	0.0002	5
e	50	500	9.97	124	82	99.7	0.089	0.004	5

Data sets 4-6 and A-E are from [49] and [44] separately, and the details of those data sets is shown in Table 3.5.

In each problem of Beasley data, every column covers at least one row and every row is covered by at least two columns. Note that the cost of each column of a matrix is also randomly generated, representing generally defined cost. To use them for our set covering problem, we replaced the original column costs by using the cardinalities of the columns.

Figure 3.3 shows the distributions of the element (document) degrees of parts of 50 matrices. Their distributions nearly follow the *normal distribution* [52]. For each histogram in the figure, x and y axes are for the element degree and the number of elements respectively, and the bin size is 5.

3.5.3 Results

To clearly explain the results of the two methods, we separate the results obtained before removing redundancy from those without redundancy based on our corpus and the Beasley data.

For our data, Table 3.6 and Figure 3.4 show the results of the experiments with redundancy. According to the table and the figure, our weighted greedy method is much better than the greedy method applied to the four data sources:

- Even the maximum cost calculated by our weighted greedy method is better than the minimal cost calculated by the greedy method;
- On average, our method outperforms the greedy method by approximately 15%.

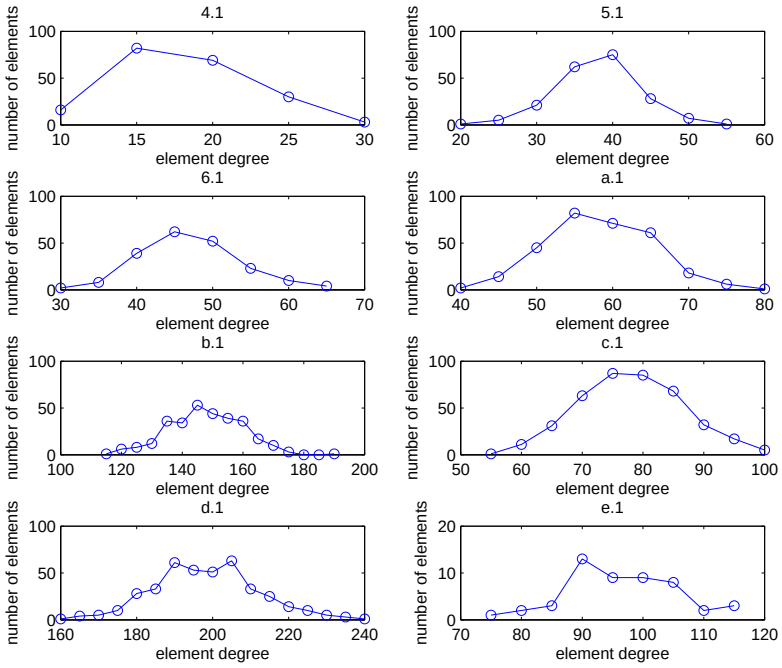

Figure 3.3: The distributions of element degrees of part of the Beasley data. The bin size is 5.

Table 3.6: The results with redundancy on our data sources.

Data source	Greedy Method				Weighted Greedy				IMP(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVE
Reuters	57907	55117	56716	508	47673	47664	47668	5	17.6	13.5	15.9
Wiki	69551	65293	67490	808	57001	56996	56998	2	18.0	12.7	15.5
Gov2	77302	71924	73444	844	62691	62673	62681	8	18.9	12.8	14.6
Newsgroup	68546	63331	66114	1089	56911	56319	56675	287	17.8	11.0	14.2

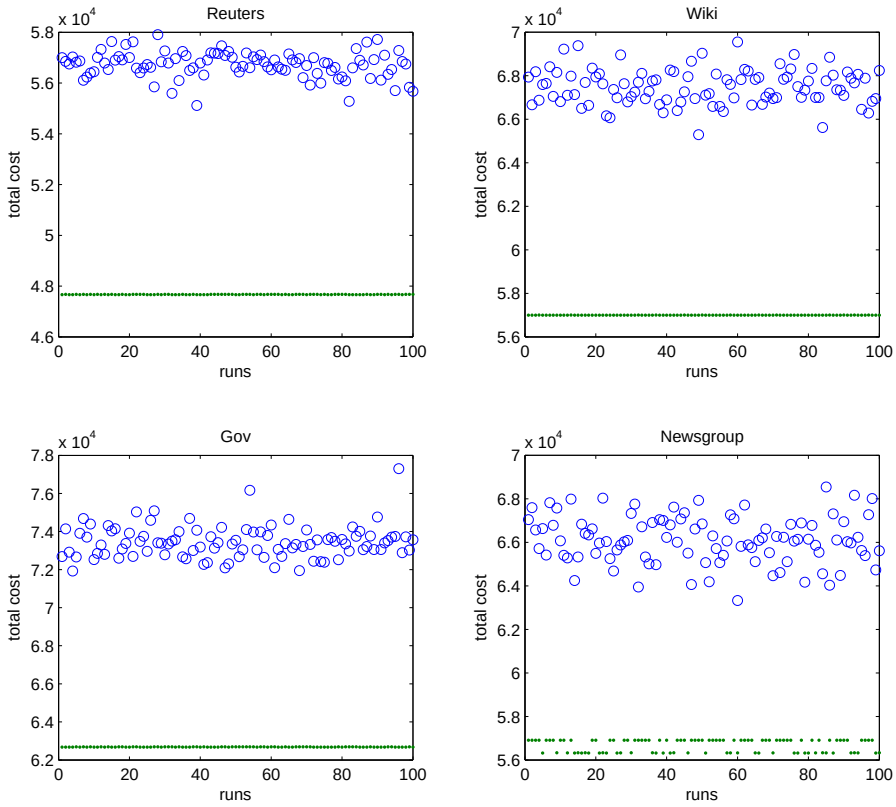

Figure 3.4: The results of all experiments on each corpus data with redundancy based on 100 runs. 'o': the greedy method, '.': the weighted greedy method.

The purpose of introducing *weight* into the greedy algorithm is to cover documents with low document degrees in earlier steps during the query selection. This is confirmed in our experiment. Figure 3.5 shows the average document degrees of the newly covered documents against the coverage in the greedy and weighted greedy query selection processes. From Figure 3.5, we find that,

- for the greedy algorithm, the documents with high document degrees are covered at low coverage (in earlier steps), and the documents with low document degrees are covered at high and moderate coverage (in later steps);
- compared to that of the greedy algorithm, its shape is relatively even. The documents with high document degrees are covered at moderate and high coverage, and the documents with low document degrees are covered at very high and very low coverage.

As we mentioned, if the documents with high degrees are covered in earlier steps, it will be covered repeatedly later and the values of *new/cost* of the rest of the columns to be selected will be reduced even more, which will lead to a higher total cost. This is demonstrated in Figure 3.6:

- except for the very beginning, the values of *new/cost* of the selected columns from the weighted greedy algorithm are better than the ones from the greedy algorithm;
- the values of *new/cost* of the selected columns from the greedy algorithm decrease faster than the ones from the weighted greedy algorithm;

Figure 3.7 shows the average document frequencies of the selected documents in the greedy and weighted greedy query selection processes on our data. From df angle, in our data, there is not much difference between the two algorithms.

For the four corpus data sources, Table 3.7 and Figure A.1 show the results of the two methods without redundancy. From Table 3.7, we find that

- the maximum and minimal costs of our weighted greedy method are better than the corresponding ones from the greedy method;
- on average, our method outperforms the greedy method by approximately 5%.

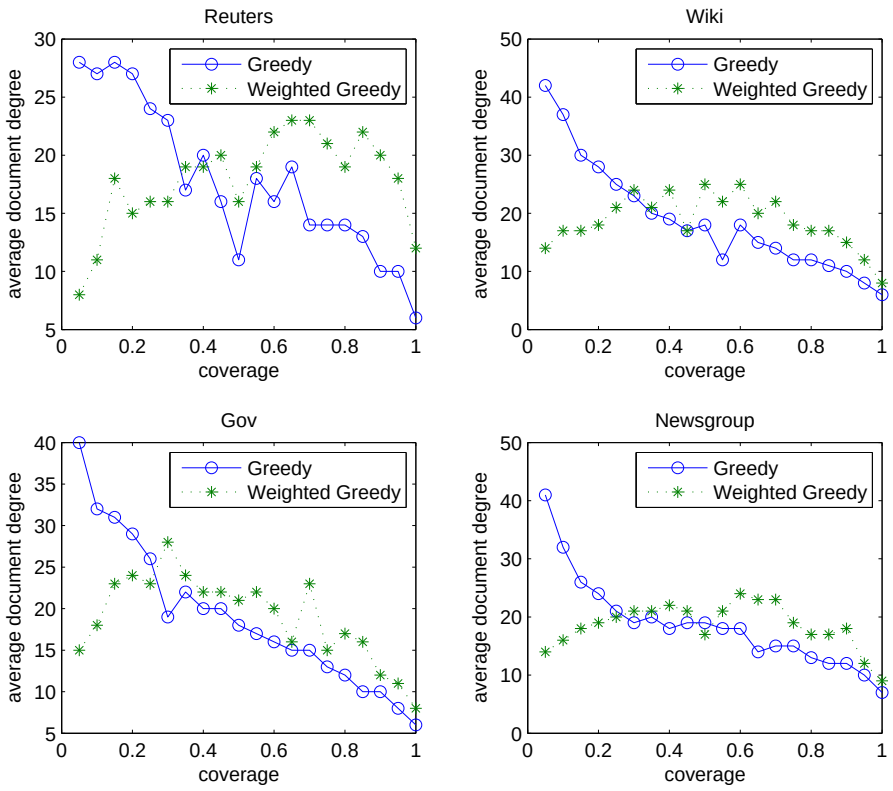

Figure 3.5: The average document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on our data sources. The bin size is 5%.

Table 3.7: The results without redundancy on our data sources.

Data source	Greedy Method				Weighted Greedy				IMP(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVE
Reuters	41407	38821	40173	518	39122	38354	38795	175	5.52	1.20	3.43
Wiki	49464	46422	47857	666	45744	44980	45344	183	7.52	3.11	5.25
Gov2	55845	52815	54028	600	52531	51378	52014	248	5.93	2.72	3.73
Newsgroup	47793	44397	45761	704	44611	42414	43578	566	6.66	4.47	4.77

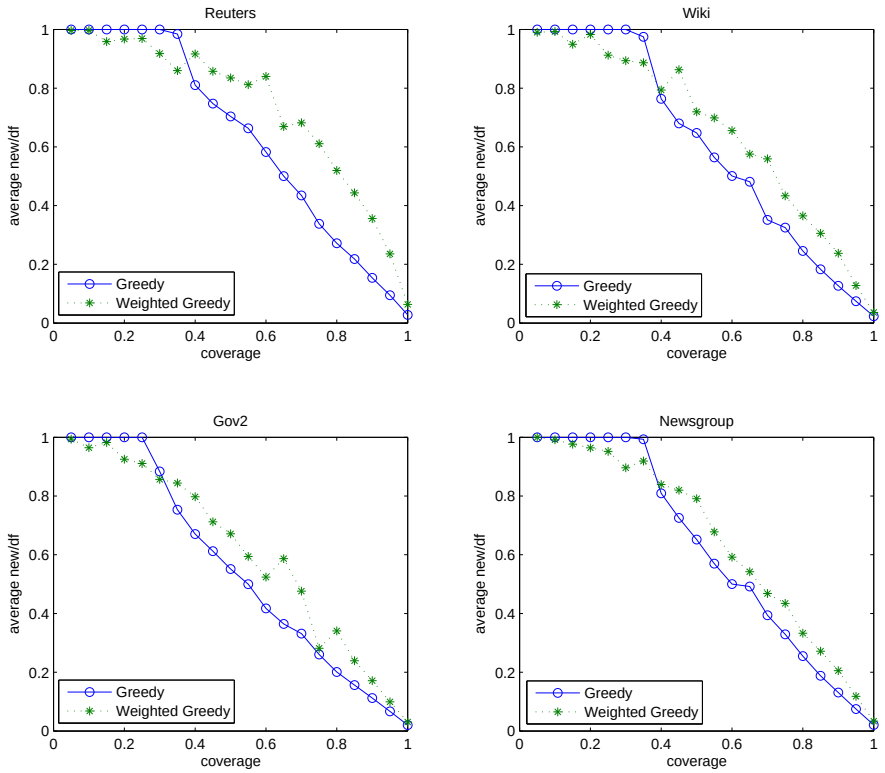

Figure 3.6: The average values of new/df in the greedy and weighted greedy query selection processes on our corpus data sources. The bin size is 5%.

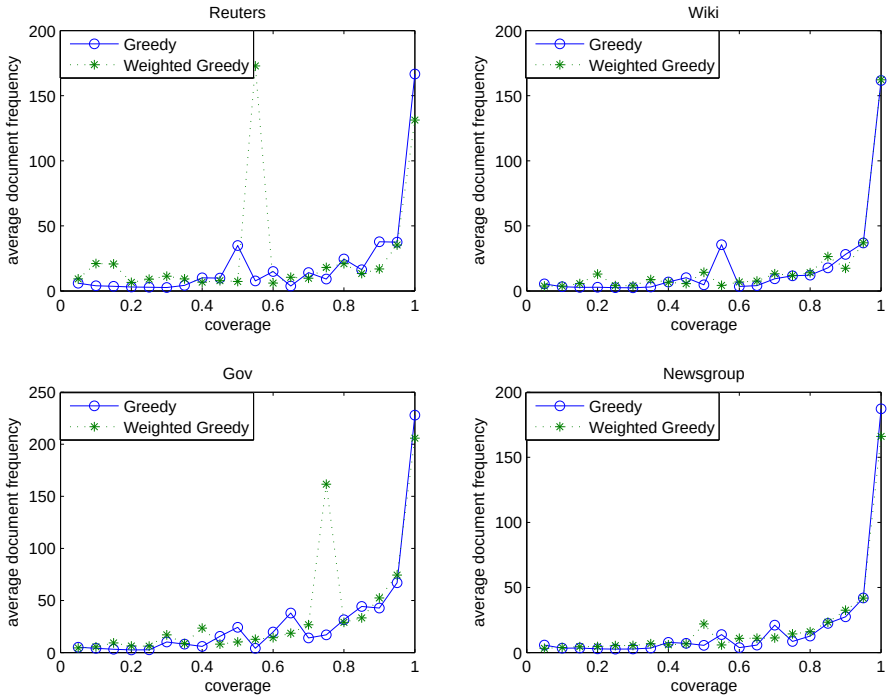

Figure 3.7: The average document frequencies of the selected documents in the greedy and weighted greedy query selection processes on our data sources. The bin size is 5%.

From Table 3.6 and Table 3.7, we can see that, for both algorithms with redundancy and algorithms without redundancy, our method outperforms the greedy method. There is less improvement by using our method, however, when redundancy is removed from the original result. The reason is that, when we remove redundancy from the original result, some columns causing overlapped coverage are removed. This causes the results from the greedy and weighted greedy methods to be closer to each other.

We have shown the experimental results of running the greedy and weighted greedy algorithms on the four corpus data sources for crawling. Similar experiments are also carried out on the Beasley data whose element degrees have a normal distribution and small dispersion of document degree.

Table 3.8 and Table 3.9 show the average results for each data set from the two methods with and without redundancy respectively (the corresponding complete tables are shown in Table A.1 and Table A.2). From the tables, we find that,

- for each data set, the average maximum cost of the weighted greedy method is better than the that from the greedy method;
- for around half of the data sets, the average minimal costs of the weighted greedy method are worse than those generated by the greedy method;
- for all data sets, the average improvement of the weighted greedy method is around 2%.

Overall, with or without redundancy, the improvement of our weighted greedy method is small. In some cases, the results from executing the greedy algorithm are better than those from executing ours.

In the Beasley data, for the results with redundancy (shown in Figure 3.8), the less improvment is caused by the less variance of the document degrees due to the normal distribution shown in Figure 3.3. Less variance of the document degrees leads to the less difference when documents with low degree are covered at earlier steps.

Figure 3.9 shows the document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on parts of the Beasley data (the solid and dashed line represent the results of the greedy and weighted greedy methods respectively). We find that, like our data, the weighted greedy method

Table 3.8: The average results with redundancy for each set of the Beasley data.

Data set	Greedy Method				Weighted Greedy				IMP(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVE
4	237.5	216.8	226.6	3.23	218.6	215.2	216.3	1.15	7.96	0.73	4.55
5	220.9	208.8	214.6	2.35	209.2	206	207.3	0.78	5.29	1.33	3.40
6	284.4	259	270.6	5.28	264	264	264	0	7.17	-1.94	2.44
a	344.4	334.2	342	3.62	335.6	334	334.6	0.44	2.39	0.06	2.16
b	440.2	410.4	424.6	6.06	420	419.8	419.8	0.1	4.59	-2.29	1.13
c	496.4	471.6	484.6	4.98	478.4	478.2	478.2	0.1	3.63	-1.40	1.32
d	631.2	590	611.4	7.98	605.6	605.6	605.6	0	4.05	-2.65	0.95
e	69.6	59.8	64.4	2.16	63.6	63.6	63.6	0	8.63	-6.36	1.25

Table 3.9: The average result without redundancy for each set of Beasley data.

Data set	Greedy Method				Weighted Greedy				IMP(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVE
4	233.1	215.8	223.8	3.4	217.2	211	213.7	1.57	6.82	2.22	4.51
5	219	207.2	212.8	2.36	208.2	203.1	205.4	1.17	4.92	1.97	3.40
6	283.2	257.4	270.4	5.24	263.4	263.4	263.4	0	6.99	-2.33	2.59
a	349.6	330.8	339.4	3.86	332.8	329.4	330.8	1.1	4.80	0.42	2.53
b	434	416	424.6	6.12	418.6	418.4	418.4	0.1	3.45	-0.64	1.46
c	489.8	474.6	482.8	4.92	475.4	473.8	474.4	0.44	2.88	0.15	1.74
d	631.2	590	611.2	8.02	605.6	605.6	605.6	0	4.05	-2.65	0.92
e	69.6	59.8	64.4	2.16	63.6	63.6	63.6	0	8.63	-6.36	1.25

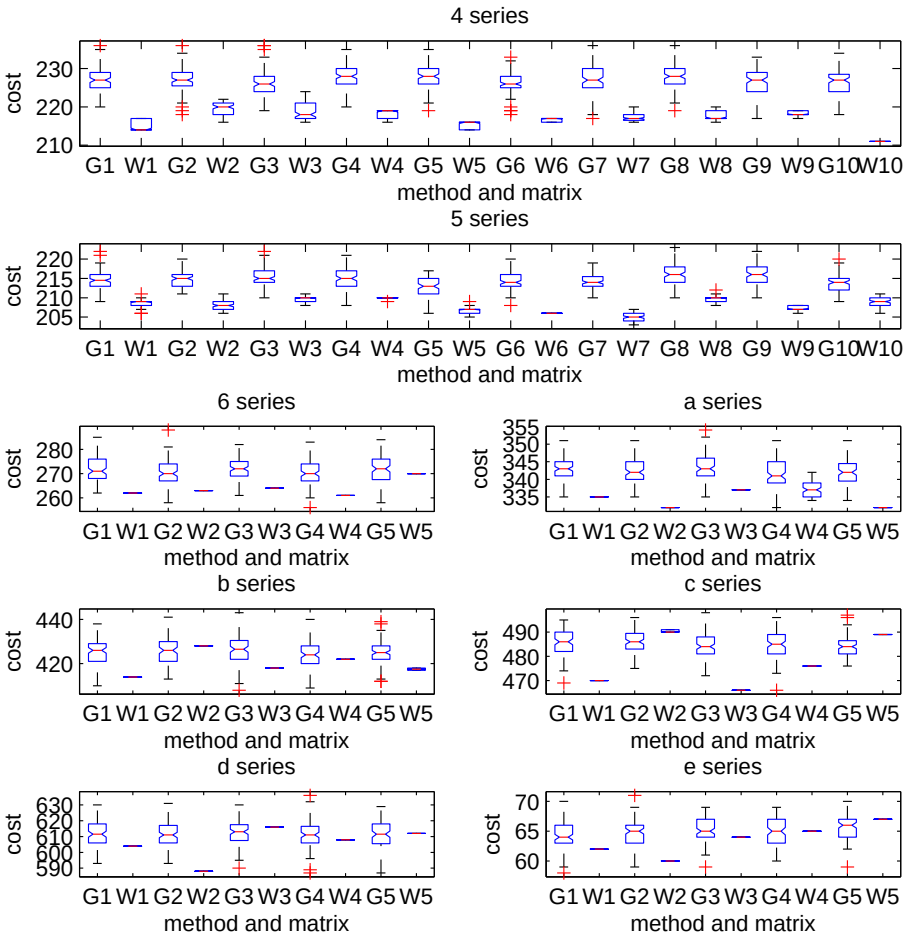

Figure 3.8: The results of all experiments on Beasley data with redundancy. G: the greedy method, W: the weighted greedy method.

covers the documents with high degree earlier than the greedy method, but the difference is much less than the one in our data sources, which can be shown in Figure 3.9. It shows the average document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on parts of the Beasley data.

Thus, the values of *new/cost* for each column in the greedy and weighted greedy query selection processes are close to each other, as shown in Figure 3.10.

Figure 3.11 shows the average document frequencies of the selected documents in the greedy and weighted greedy query selection processes on parts of the Beasley data. Like in our corpus data, they are quite similar to each other in the Beasley data as well.

For the results without redundancy (partly shown in Figure A.2), the improvement is almost the same as the one with redundancy. Since the original solutions from the two methods are close to each other, the solutions after redundancy removal are also similar to one another.

3.5.4 Impact of data distribution

The only difference in the query selection strategy between the weighted greedy and greedy algorithms is that the formula *new/cost* is replaced by the formula *qw/cost*. The key idea of the change is that all newly covered rows should not be considered totally in the same way (represented as unit cost 1) and there should be difference in covering different rows. In our context, such difference is caused by the covering sequence for each row, i.e., some rows should be covered earlier than some others by selecting the corresponding columns.

According to Definition 3, $dw(i) = \frac{1}{deg(i)}$ (the inverse of document degree). If all rows are covered by the same number of columns (that is $\forall i, deg(i) = c$ where c is a non-negative constant), all document weights are the same. In this setting, the weighted greedy algorithm turns out to be the same as the greedy method. Of course, there would be no saving in this case. We can only expect cost savings by using our algorithm when the document degrees are different, i.e. some documents have larger degrees than some others.

Therefore, we argue that the bigger dispersion of $deg(i)$ is the basis of the more improvement of the weighted greedy method.

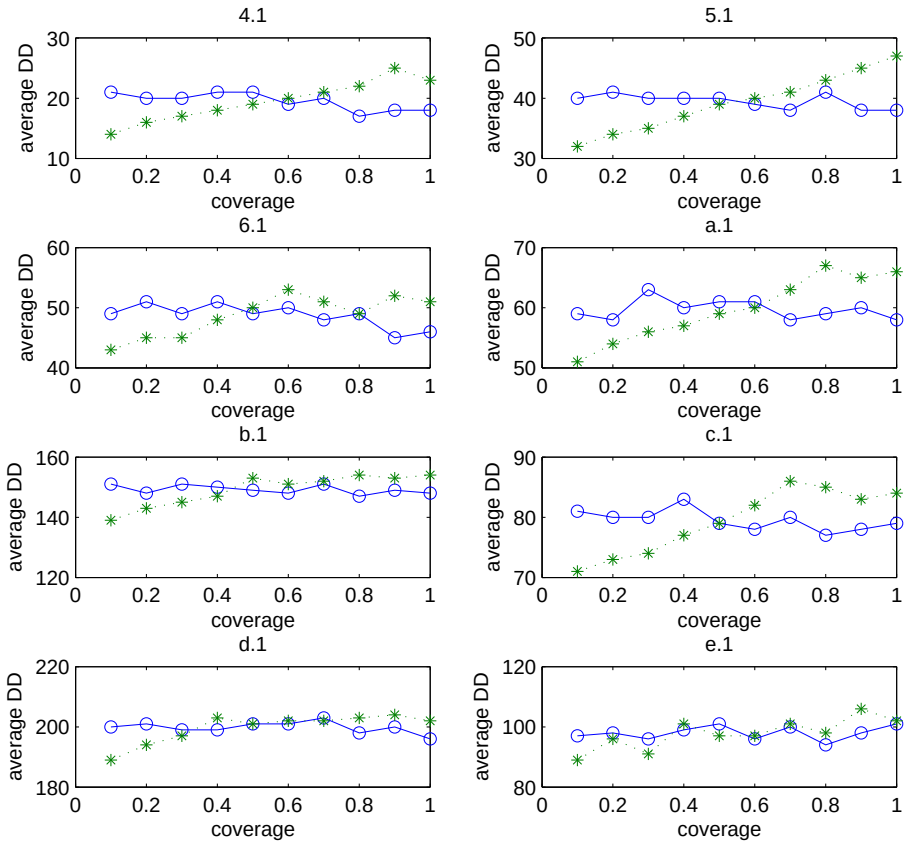

Figure 3.9: The average document degrees of the newly covered documents in the greedy and weighted greedy query selection processes on parts of the Beasley data. The bin size is 10%. The solid and dashed lines represent the results of the greedy and weighted greedy methods respectively.

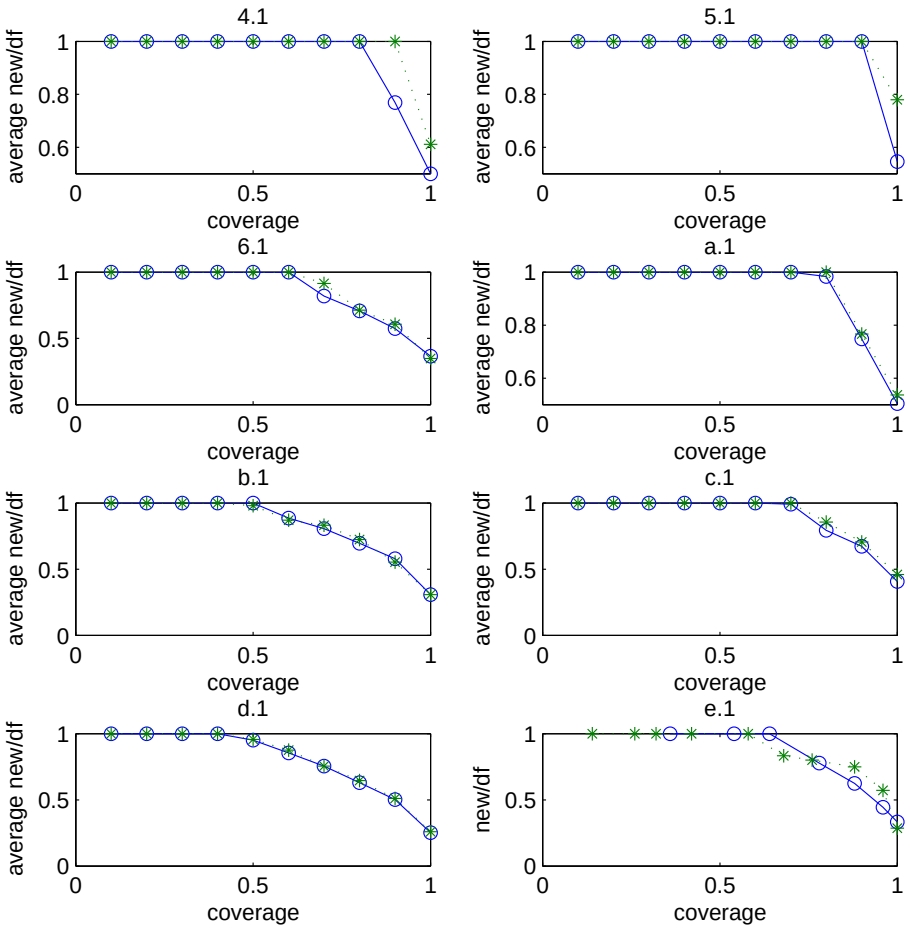

Figure 3.10: The average values of *new*/*cost* for each column in the greedy and weighted greedy query selection processes on parts of the Beasley data. The bin size is 10%. In subgraph e.1, the histogram is replaced with the scatter plot because some values of *new*/*cost* are zero and no column is selected in the corresponding coverage ranges.

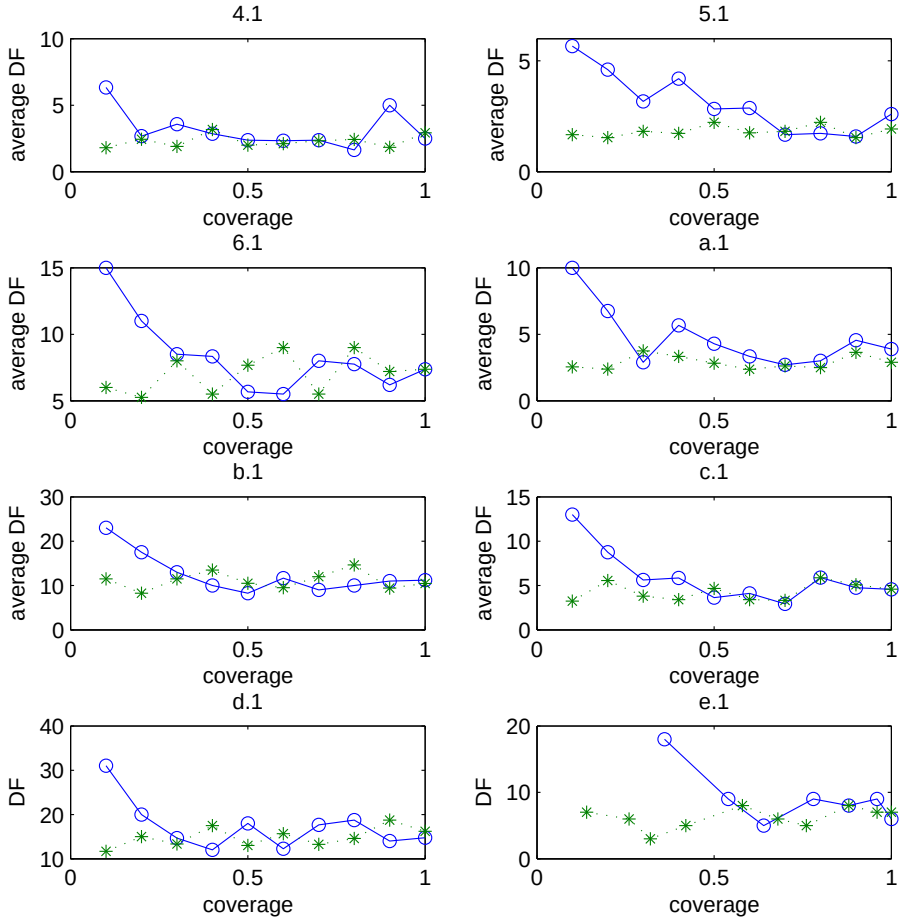

Figure 3.11: The average document frequencies of selected columns in the greedy and weighted greedy query selection processes on part of Beasley data. The bin size is 10%. In subgraph e.1, the histogram is replaced with the scatter plot and the reason is same as Figure 3.10.

The dispersion of the document degree heavily depends on its distribution of document degree. For example, with uniform distribution, there is no dispersion on the document degree. This is the reason that, for the experiments in this chapter, we use two different kinds of datasets with two different types of distribution data. For our data, the distribution of the document degrees is close to the lognormal distribution shown in Figure 3.2. Such a distribution could cause a high dispersion of the document degrees; for the Beasley data, its distribution of the document degrees is similar to the normal distribution shown in Figure 3.3. Its dispersion of the document degrees is much smaller than that of our data.

Note that Figure 3.2 is based on log-log scale and Figure 3.3 is not. Although their shapes look similar, our data has a much bigger dispersion of the document degrees than the Beasley data.

The dispersion of $deg(i)$ in a data source (doc-term matrix), it can be measured by the *Coefficient of Variation* (CV) defined in equation 3.8.

From Table 3.4 and Table 3.5, we find that the dispersions of $deg(i)$ in our data are much bigger.

For the results with redundancy, in our data, the weighted greedy method outperforms the greedy method at around 15%; in the Beasley data, there are less improvements ($\leq 7\%$). As the improvement from using our algorithm is based on covering documents with small document degrees earlier, if there is no much difference among the document degrees, we cannot gain much improvement. Compared to the lognormal distribution, the normal distribution has a much smaller dispersion of the document degrees, thus there is not much savings when using the weighted greedy algorithm.

Having shown our results on two kinds of datasets with different data distributions, we quantitatively measure the improvement according to the dispersion of document degrees: we will measure the improvement by using the weighted greedy algorithm according to the CV of the document degree. To do so, the *Pearson Product Moment Correlation Coefficient* (PC for short) is used here. PC is a measure of the linear correction between two variables X and Y, giving a value between -1 and +1 inclusive. The correlation is 1 in the case of an increasing linear relationship and is -1 in case of a decreasing linear relationship, and a value between -1 and 1 indicates the degree of linear dependence between the variables. It is defined as follows:

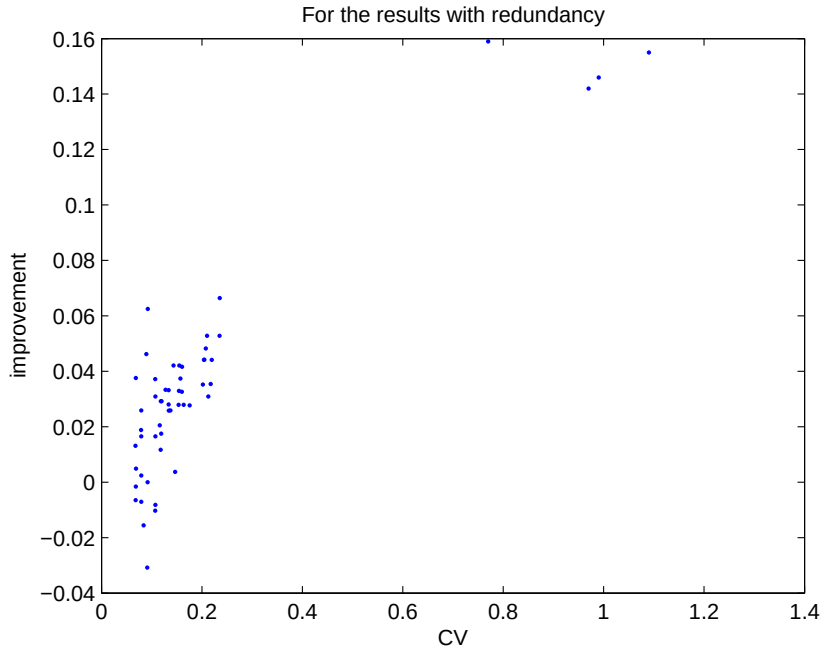

Figure 3.12: the relationship between the CV of document (element) degree and the average of the improvement of the weighted greedy method on our and Beasley data with redundancy.

Definition 5 ***(PC)*** *Given two random variables X and Y, the Pearson's correlation coefficient ρ between the two variables is defined as the covariance of the two variables divided by the product of their standard deviations.*

$$\rho = \frac{\sum_{i=1}^{n}(X_i - \bar{X}) \times (Y_i - \bar{Y})}{\sqrt{\sum_{i=1}^{n}(X_i - \bar{X})^2}\sqrt{\sum_{i=1}^{n}(Y_i - \bar{Y})^2}}. \tag{3.9}$$

In our context, X is the CV and Y is the average of the improvements of the weighted greedy algorithm for each matrix.

Based on all the results with redundancy of the corpus and the Beasley data, we have $\rho = 0.88$. Figure 3.12 shows the relationship between the CVs and the average improvements of all the results with redundancy. From the figure, we can see that the improvement of the weighted greedy method has a positive linear correlation with the CV of the document degrees.

In addition, the relationship between the CV and the average improvements of

all results without redundancy is shown in Figure A.3 as well, and its $\rho = 0.33$.

3.6 Conclusion

In this chapter, we present a weighted greedy method to addressing the query selection problem. It is designed for textual deep web data sources whose document degrees follow a lognormal distribution. Targeting the lognormal distribution, the weighted greedy method shows a better performance than the greedy algorithm that is one of the most popular query selection methods. In addition, the weighted greedy method is efficient for large-scale deep web data sources (complexity $O(m \times n)$) and it is practical. Moreover, the performance of the weighted greedy method on average outperforms the greedy method on the Beasley data whose element degrees have a normal distribution and we further argue that the improvement of the weighted greedy method without redundancy removal has a positive linear correlation with the CV of the document degrees of the target data source.

Chapter 4

Sampling-based query selection method

4.1 Introduction

In the previous chapter, the query selection problem is represented by Set Covering Problem (SCP). The experimental results from the greedy and weighted greedy algorithms can be considered a baseline to compare with other query selection methods. However, for real cases, before crawling the deep web there is no input to the set covering algorithm, i.e., neither documents nor terms are available. To bootstrap the process, a sample of the data source is needed so that some good terms are selected and sent, and more documents are returned and added to the input of the set covering problem.

The challenge is to select a set of appropriate queries without the global knowledge of the data source.

As we mentioned in Chapter 2, there are two typical methods to learn queries. One is the incremental method [17, 7] and the other is the sampling-based method [16]. A shortcoming of the incremental method is the requirement of downloading and analyzing all the documents covered by current queries in order to select the next query to be issued, which is highly inefficient. Even when our final goal is to download documents instead of URLs, it would be efficient to separate the URLs collection from the document downloading itself. Usually, the implementation of a downloader should consider factors such as multi-threading and network exceptions,

and should not be coupled with link collection.

Targeting the disadvantage of the incremental method, we propose an efficient sampling-based query selection method for collecting the URLs of the documents inside a data source. We first collect from the data source a set of documents as a sample that represents the original data source. From the sample data, we select a set of queries that cover most of the sample documents with a low cost based on the greedy algorithm defined in Algorithm 1. Then we use this set of queries to extract data from the original data source.

The main contribution of our sampling-based method is the hypothesis that the queries working well on the sample will also induce satisfactory results on the original data source. More precisely, there are three hypotheses:

- The vocabulary learnt from the sample can cover most of the original data source;
- The cost of our method in the sample can be projected to the original data source;
- The sizes of the sample and the query pool for our method do not need to be very large.

While the first result can be derived from [1], the last two are not reported in the literature as far as we are aware of. As our method is dependent on the sample size and the query pool size, we have empirically determined the sizes for the sample and the query pool for effective crawling of a deep web data source.

In this chapter, we still focus on textual data sources with keyword-based query interfaces and without return limit.

4.2 Problem formalization

4.2.1 Hit rate and overlapping rate

Since the information of the original data source could barely be known, it is hard to retrieve all the documents inside it like the case in the last chapter. Thus, here the goal of the query selection problem is to retrieve most of the links of the documents. This is formalized as the *hit rate* that is defined below.

Definition 6 ***(Hit Rate, HR)*** *Given an $m \times n$ doc-term matrix $A = (a_{ij})$ derived from a data source $DB = \{d_1, \ldots, d_m\}$ and a query pool $QP = \{q_1, \ldots, q_n\}$ and a set of queries $Q = \{q^1, \ldots, q^k\}$ ($Q \subseteq QP$) represented by an n-binary vector $X = \{x_1, \ldots, x_n\}$, $x_j = 1$ if $q_j \in Q$, otherwise, $x_j = 0$. Let $Y = \{y_1, ..., y_m\}$ be an m-binary vector and $y_i = 1$ if $\sum_{j=1}^{n} a_{ij}x_j \geq 1$, otherwise, $y_i = 0$. The hit rate of Q in DB, denoted by $HR(DB, Q)$, is defined as the ratio between the number of unique documents (u) collected by queries in Q and the size of the data source DB, i.e.,*

$$u = \sum_{i=1}^{m} y_i,$$

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

Here the *overlapping rate* is used to measure the performance of a set of queries instead of the total cost, which is defined as follows:

Definition 7 ***(Overlapping Rate, OR)*** *Given a set of queries $Q = \{q^1, \ldots, q^k\}$, the overlapping rate of Q in DB, denoted by OR(DB, Q), is defined as the ratio of the total number of th collected documents (v) to the number of the unique documents retrieved by queries (u) in Q, i.e.,*

$$v = \sum_{j=1}^{n} c_j x_j,$$

$$OR(DB, Q) = \frac{v}{u},$$

where $c_j = \sum_{i=1}^{m} a_{ij}$.

Since data sources vary in their sizes, the total number of the documents collected by a set of queries Q for crawling a large data source with a larger v does not necessarily mean that its cost is higher than the cost of the set of queries Q' for crawling a small data source with a smaller v. Therefore, we normalize the cost by dividing the total number of the documents by the unique ones. When all the documents are retrieved, u is equal to the data source size.

Example 3 *Here we use the doc-term matrix A shown in Table 3.1 as an example,* $DB = \{d_1, \ldots, d_9\}$ *and the query pool* $QP = \{q_1, \ldots, q_5\}$*. OR and HR for queries* $\{q_4, q_3, q_5\}$ *and* $\{q_4, q_3, q_1\}$ *are calculated as follows:*

$$OR(DB, \{q_4, q_3, q_5\}) = \frac{4+5+5}{9} = \frac{14}{9}$$

$$HR(DB, \{q_4, q_3, q_5\}) = \frac{9}{9} = 1$$

$$OR(DB, \{q_4, q_3, q_1\}) = \frac{4+5+4}{9} = \frac{13}{9}$$

$$HR(DB, \{q_4, q_3, q_1\}) = \frac{9}{9} = 1$$

since $\{q_4, q_3, q_1\}$ has a lower OR than $\{q_4, q_3, q_5\}$ and they produce the same HR, we should use the former instead of the latter to retrieve the documents.

4.2.2 Relationship between HR and OR

Another reason to use HR and OR to evaluate the query selection method is that there is a fixed relationship between HR and OR when documents can be obtained randomly, i.e., if documents can be retrieved randomly with equal capture probability, it is shown in [53, 14] that

$$HR = 1 - OR^{-2.1}. \tag{4.1}$$

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

$$HR = 1 - OR^{-1}. \tag{4.2}$$

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

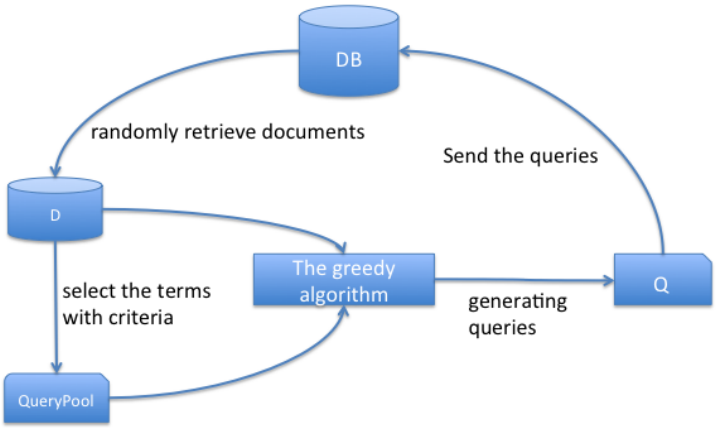

Figure 4.1: Our sampling-based query selection method.

4.3 Our method

4.3.1 Overview

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

With our query selection method for deep web crawling, we first download a sample set of documents from the original data source. From this sample, we select a set of queries that can cover most of the documents in the sample data source with low cost. We show that the same set of queries can also be used to cover most of the documents in the original data source with a low cost. This method is illustrated in Figure 4.1 and explained in Algorithm 4.

In this chapter, we use $DWC(DB, m, \mu)$ to denote the output obtained by running the algorithm with the input data source DB, the sample size m, and the average document degree μ.

Algorithm 4: Outline of Deep Web Crawling algorithm DWC(DB, s, μ).

Input: the original data source DB, the sample size m, the average document degree μ.

Output: a set of queries Q.

Process:

1. Create a sample data source D by randomly selecting m number of documents from the corpus;
2. Create a query pool QP from the terms that occur in D and their average document degree is μ;
3. Select a set of queries Q from QP that can cover at least 99% of the sample data source D by running the greedy algorithm defined in Algorithm 1;

In our algorithm and experiments, random samples are obtained by generating uniformly distributed random numbers within the range of the document IDs in the corpus. However, in practical applications, we do not have direct access to the whole corpus. Instead, only queries can be sent and the matched documents are accessed. In this scenario, the random sampling of the documents in a corpus is a challenging task, and has attracted substantial studies (for example in [39, 1, 38]). Since the cost of obtaining such random samples is rather high, our experiments skip the random sampling process and take the random samples directly from the corpus.

Not all the terms in D can be candidate queries. To avoid the scalability problem and reach a high coverage in DB without using stop words, only the terms whose sample document frequencies are within a certain range are qualified to be input into the query pool QP.

Finally, the set of queries in Q selected from the query pool depends on the sample by using the greedy algorithm.

To refine this algorithm, two more parameters need to be decided.

One is the number of documents that should be selected into the sample, i.e., the size of the sample data source D. Although in general a larger sample will always produce a better result in DB, we need to find an appropriate size for the sample so that it is amenable to efficient processing while still large enough to produce a satisfactory query list in QP by using the greedy algorithm.

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

The soundness of Algorithm 4 relies on the hypothesis that the vocabulary that works well for the sample will also be able to extract the data from the actual data source effectively. More precisely, this hypothesis says that

- the terms selected from the sample data source D will cover most of documents in the original data source DB, and
- the overlapping rate in DB will be close to the overlapping rate in D.

Before analyzing the correspondence between the sample and the original data source in detail, we first show how to address the problem of query selection from a sample data source.

4.3.2 Creating the query pool

To select the queries to issue, first of all we need to obtain a query pool QP. Here QP is built from the terms in a sufficiently *unbiased* sample of the original corpus data source. We should be aware that random queries cannot produce random documents because large documents have the higher probability of being matched. It is a rather challenging task to obtain random documents from searchable interface.

Not every word in the first batch of the search results should be taken into our consideration, due to the time constraint we suffer in order to calculate an effective query set Q from D with high hit rate and low overlapping rate. As we mentioned before in Chapter 3, searching for an optimal query set can be viewed as a set covering problem. Set covering problem is NP-hard, and even the greedy algorithm shown in Algorithm 1 has a time complexity that is at least quadratic to the number of terms in QP. This determines that we are able to calculate Q only with a query pool of limited size. The first batch of the search results, on the contrary, may well-exceed such a limit. For instance, the first-batch of results (around 3000 documents) selected randomly from the newsgroup corpus contains more than 80,000 unique words. Therefore, we consider only a subset of words from the sample documents as a query pool.

Apparently, the size of this subset affects the quality of the set of queries in Q we generate. Moreover, it should be measured relative to the sample size and the document frequencies of the terms.

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

There is another factor to consider when selecting queries in the query pool, i.e., the document frequencies of the terms in the sample. There are several options:

- **Random terms** Randomly selecting the terms in the sample data source may be an obvious choice. However, it suffers from low hit rate in the original data source DB because most of the randomly selected queries are of low document frequencies. According to Zipf's Law [30], the distribution of words sorted by their frequencies (i.e. their number of occurrences) is very skewed. In one of our sample data source D, about 75% of the words have very low frequencies. Therefore, randomly pulling words from the vocabulary to reach a high coverage in DB seems like very hard. The reasons are that 1) we will use too many such queries with small number of returns to create the query pool QP and it is unaffordable for most of the set covering algorithms to deal with such a big QP; 2) even if D can be fully covered by the low frequency queries, the hit rate in DB caused by them could be rather low because the sample document frequency of the low frequency term usually cannot be proportionally enlarged in DB. For example, if the df of a low frequency term in D is "1", it is highly likely that its df in DB is still 1 or 2, not the proportionally enlarged document frequency (1*$|DB|/|D|$).

- **Popular terms** Another possible choice is the popular words in the sample, which are terms with high document frequencies. Although high document frequency terms in D are easier to keep their properties in DB than low frequency terms, popular words such as stop words (words filtered out prior to) are not selected for two reasons. One is that many data sources do not index stop words. Hence using stop words to extract documents may return nothing. The second reason is that, if only high frequency terms are used as queries, they could cause more duplicates.

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

The size of the query pool should also be measured relative to the document frequencies of its terms: terms with low frequencies contribute less to the coverage of the document set. Taking into account the sample size and the document frequencies, here we use the average document degree u defined in equation 3.7 to measure the pool size. The average document degree represents the total number of documents that can be retrieved by the queries in the query pool, normalized by the sample size. For example, if $\mu = 20$, on average each document is captured 20 times if all queries in QP are used. We will discuss exactly proper value for μ below.

4.3.3 Selecting queries from the query pool

Once the query pool is established, we need to select from this pool some queries that will be sent to the original data source DB.

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

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

$$HR(D, Q) = p,$$

and

$$OR(D, Q)$$

is minimal.

Table 4.1: Summary of test corpora

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

According to Definition 7, if $OR(D, Q) \times |D|$ and $p = 1$, it is the total cost of Q and the problem is exactly the same as the set covering problem defined in Definition 1. Here the greedy algorithm defined in Algorithm 1 can be used by slightly changing the requirement (line 2) *While*$(\sum_{i=1}^{m} y_i < m)$ into *While*$(\sum_{i=1}^{m} y_i < p \times m)$ where $0 \leq p \leq 1$.

4.4 Experiment environment

We carried out our experiments on the same corpus data in Chapter 3. The four corpora are Reuters, Gov, Wikipedia, and Newsgroup. They contain different numbers of documents ranging between 30 thousand and 1.4 millions. Their characteristics are summarized in Table 4.1. Again these are standard test data that are used by many researchers in information retrieval.

- *Reuters* is a TREC data set that contains 806,790 news stories in English (http://trec.nist.gov/data/reuters/reuters.html).
- *Gov* is a subset of Gov2 that contains 1 million documents. Gov2 is a TREC test data collected from .gov domain during 2004, which contains 25 million documents. We used only a subset of the data for efficiency consideration.
- *Wikipedia* is the corpus provided by wikipedia.org which contains 1.4 million documents.
- *Newsgroups* includes 1,372,911 posts in various newsgroups and only 30,000 posts report here.

In the experiments, we built our own search engine using Lucene [54] to obtain the details of a data source, such as its size. In the real deep web data sources, usu-

ally the total number of documents is unknown, hence it is impossible to calculate HR and evaluate the methods.

4.4.1 Hypothesis I

Our first hypothesis is that in general the HR in a small sample data source D can be projected to the original data source DB. Namely, the queries learnt from a small sample data source covers more than 99% data in D and can cover most of the data in the original data source as well, i.e., the queries can be used to retrieve most of the documents in DB.

Hypothesis 1 *Suppose that the sample data source D from the original data source DB and the set of queries Q are created by our algorithm. If $|D| >$ 1,000 and $\mu = 20$, then*

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

Here we assume that the size of DB is a very large number, i.e., $|DB| >> 1000$.

We tested the cases where the sample size range between 100 and 4,000 documents for the four corpora studied. Figure 4.2 shows HR in D and DB as a function of the sample size. It demonstrates that our method quickly finds the queries that can account for 90% of the documents in DB and the HRs in D and DB are close to each other when the sample size is bigger than 1,000. On the other side, it is shown that at the beginning when the sample size increases, HR in the original data source is higher. After about 1,000 documents, the gain in HR tapers and the increase of the sample size has little impact on the HR in DB.

This phenomenon can be explained by three reasons: 1) Heaps' law states that when more documents are gathered, there are diminishing returns of new words. When 1,000 documents are checked, most common words are already recovered from the sample. There are very few useful words left outside that 1,000 sample; 2) the dfs of common words in D usually can be proportionally enlarged in DB; 3) the set of selected queries from the greedy algorithm mostly contains enough common words if $|D|$ is bigger than 1,000.

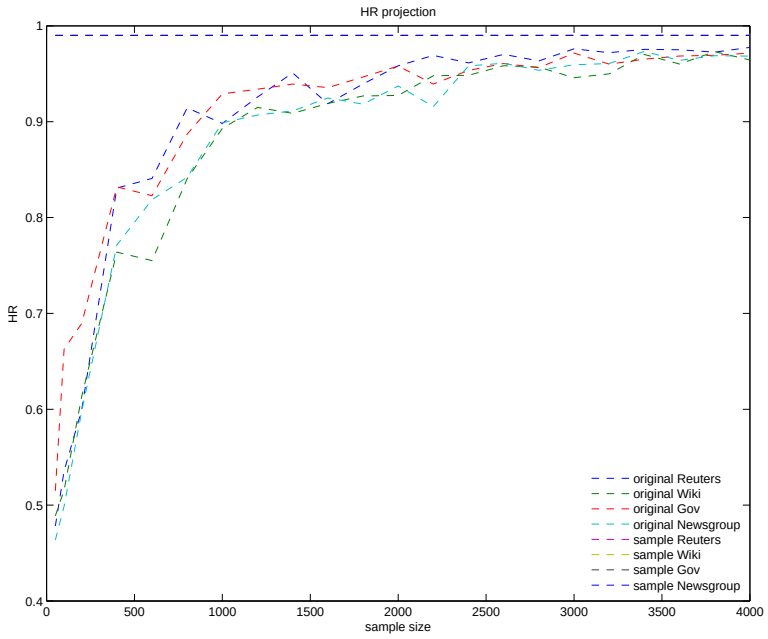

Figure 4.2: Impact of sample size on HR projection. The queries are selected from D and cover above 99% of the documents in D. The HR in DB is obtained when those queries are sent to the original data source DB. $\mu = 20$.

4.4.2 Hypothesis II

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

Although for the sample data source D, we make sure that the selected queries have a low cost by applying the greedy algorithm, we are not sure yet whether the cost would also be low for the original data source DB. Hence we need to verify our second hypothesis, i.e., the queries selected by Algorithm 4 from D will also result in low overlapping rate in DB. More precisely, it can be described as:

Hypothesis 2 *Suppose that queries Q are selected by our method, i.e., $Q = DWC(DB, m, \mu)$, where $m > 1,000$ and $\mu > 10$, then*

$$1.5 \times OR(D, Q) > OR(DB, Q).$$

To verify this, we conducted a series of experiments to compare ORs in sample D and the original data source DB. Figure 4.3 illustrates the experimental result. In this experiment, the range of the size of sample is between 100 and 4000 and the average document degree μ is set to 20. From Figure 4.3, we find that the overlapping rates in D and DB are close to each other when sample size is bigger than 1,000. The OR is caused by co-occurrence words and the result demonstrates that the patterns of co-occurrence words in DB can be successfully learnt by sampling. This is the basis that the OR in sample can be projected to the original data source.

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

4.4.3 Hypothesis III

The previous two hypotheses have shown that

- we can cover most of the documents based on a sample;
- we can do that with a low cost.

The next concern is exactly how large the sample and what the proper value for the average document degree of the query pool should be. Our third hypothesis is

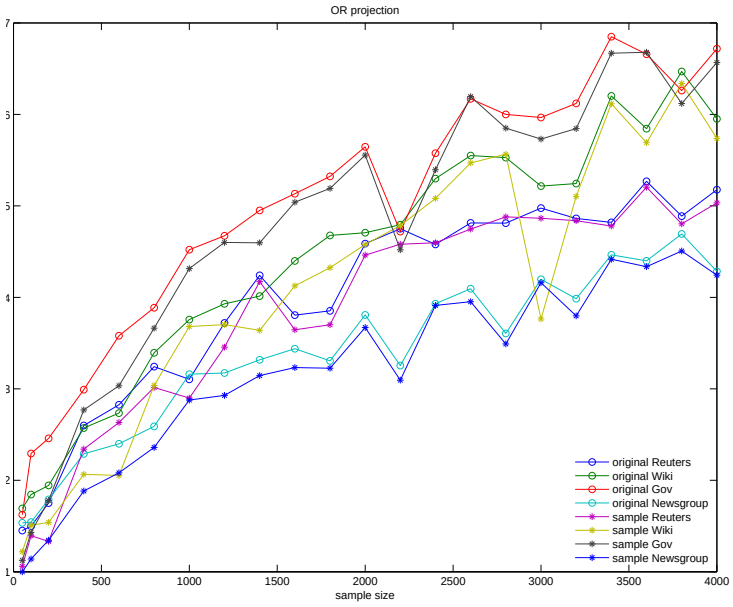

Figure 4.3: Impact of sample size on OR projection. X axis is sample size, Y axis is HR. Sample size is from 100 to 4,000 documents and $\mu = 20$.

that, to download most of the documents with low overlapping, the sample size and the value for the average document degree of the query pool do not need to be very large. This will be elaborated in two aspects, i.e., the sample size and the value for the average document degree.

Sample size

Figure 4.2 shows that a few thousands of sample documents are good enough to harvest most of the documents in DB. Increasing sample may not always lead to better coverage when the sample size is bigger than 1,000. We have the following hypothesis:

Hypothesis 3.1 *For any data source DB, there exists* $m_1 > m_2 > 1000$ *and* $\mu > 10$ *s.t. let*

$$Q_1 = DWC(DB, m_1, \mu),$$

$$Q_2 = DWC(DB, m_2, \mu),$$

then

$$HR(DB, Q_1) < HR(DB, Q_2).$$

Similarly, Figure 4.3 shows that the change of the overlapping rate becomes little when the sample size is bigger than 1,000. The impact of the sample size on OR can be summarized as following:

Hypothesis 3.2 *For any data source DB, there exists* $m_1 > m_2 > 1000$ *and* $\mu > 10$ *s.t. let*

$$Q_1 = DWC(DB, m_1, \mu),$$

$$Q_2 = DWC(DB, m_2, \mu),$$

then

$$OR(DB, Q_1) > OR(DB, Q_2).$$

Intuitively, Hypothesis 3.1 and 3.2 say that a larger sample size does not guarantee a better result. For Hypothesis 3.2, someone may say the OR is subject to the HR and it is meanless to list the OR without the reference to HR. In fact, here we assume the HRs of two samples are very close to each other and this is shown in Figure 4.3.

We conducted more experiments to investigate the impact of sample size on

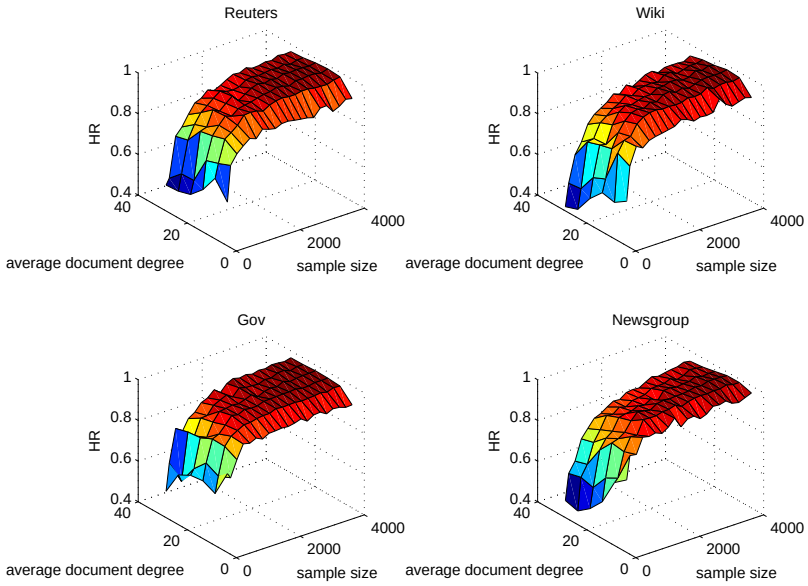

Figure 4.4: The impact of the sample size and the average document degree on HR.

the hit and overlapping rate respectively. Figure 4.4 and Figure 4.5 show the HR and OR in DB as functions of the sample size and the average document degree. The average document degree is in the range of between 5 and 30. We ignored the value smaller than five because it will not produce enough queries to cover all the documents.

The experimental results conform to Hypothesis 3.1 and 3.2 as well. Sometimes a larger sample size may induce a higher overlap in DB. Although it is counter-intuitive, the reason is that with more documents in the sample, there are more words to choose from, most of them having low frequencies according to Zipf's law. When those low frequency words are selected by the greedy algorithm, they result in a low OR in the sample data source D. However, when they are mapped to the original data source DB, the patterns of their co-occurrence in D and DB can be very different and then the OR could get worse.

Average document degree

This experiment investigates the impact of the query pool size represented by the average document degree on HR and OR. Since HR and OR will also vary with

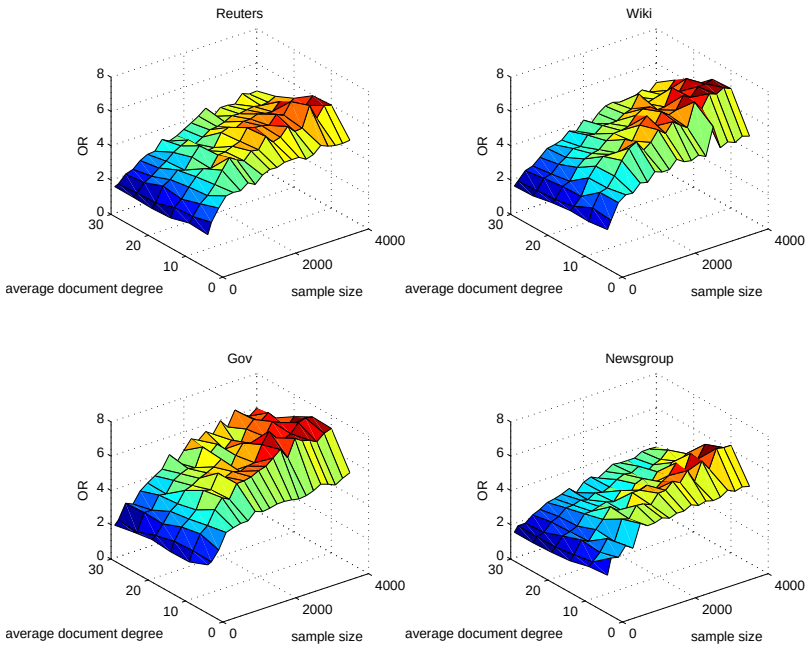

Figure 4.5: The impact of the sample size and the average document degree on OR.

different sample sizes, we include sample sizes as another dimension of input in our experiments.

First we need to investigate the impact of the value of the average document degree on HR. Our hypothesis can be formulated as below:

Hypothesis 3.3 *For any data source DB, there exists* $\mu_1 > \mu_2 > 10$ *and* $m > 1000$ *s.t. let*

$$Q_1 = DWC(DB, m, \mu_1),$$

$$Q_2 = DWC(DB, m, \mu_2),$$

then

$$HR(DB, Q_1) < HR(DB, Q_2).$$

In particular, $HR(DB, Q)$ achieves the highest value if μ is between 10 and 20.

From Figure 4.4, it can be seen that HR is low only when $\mu < 10$. When the value of μ becomes larger ($\mu > 20$), HR decreases because the inclusion of more low frequency words in D, which are most probably still of low frequencies in DB and this makes the HR becomes worse. The conclusion of this experiment is that the best performance is achieved when μ is set between 10 and 20.

Another investigation is the impact of the value of the average document degree on OR. Although Figure 4.5 shows that usually the OR decreases with the increase of μ, the OR is dependent of the HR which decreases simultaneously. Thus, in order to have an objective comparison, we measure the improvement of OR over the random method which obtains the same hit rate. Our empirical study shows the following results:

Hypothesis 3.4 *Suppose that two sets of queries Q and Q' have the same hit rate. Q is selected by our algorithm, while Q' is randomly selected from the query pool. i.e.,*

$$Q = DWC(DB, m, \mu),$$

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

Let ORimprovement be

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

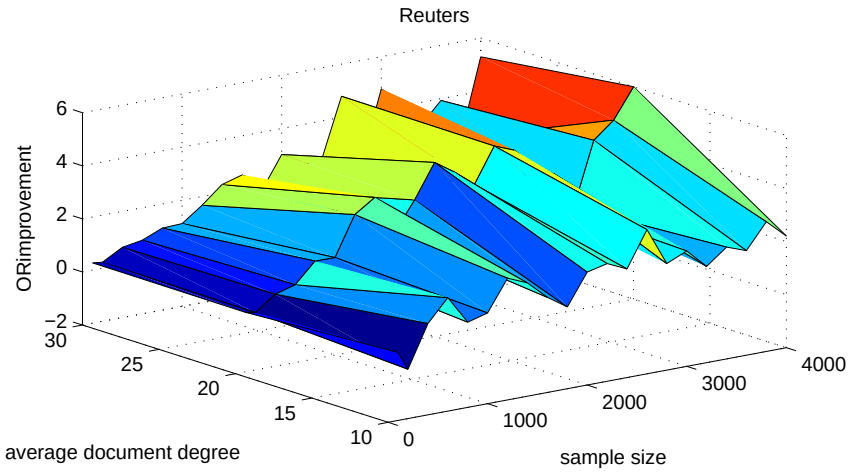

Figure 4.6: The impact of the value of the average document degree on OR improvement from the comparison between our and random methods for Reuters corpus.

Then the change of ORimprovement likes a concave function as μ grows.

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

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

Figure 4.6 depicts the improvement of OR while the sample size and the value of the average document degree vary for Reuters corpus. It shows that the value of the average document degree does not need to be very large - the best result is obtained while μ is around 20, which can be explained again by the inclusion of rare words.

Figure 4.6 also gives us an overall comparison between our method and the random method on Reuters corpus. First of all, no matter what the sample size and the query pool size are, the result of our method is always better than that of the random method. We can also see that usually the larger samples introduce more improvements in overlapping, while the value of the average document degree does not matter very much.

4.4.4 Comparing queries on other data sources

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

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

To be precise, suppose that Q is selected from one original data source DB based on our method, i.e.,

$$Q = DWC(DB, m, \mu),$$

$$m > 1000, \mu > 10.$$

Suppose that $Q^{'}$ and $Q^{''}$ are two subsets of Q that can achieve the same high HR in original data sources DB and another arbitrary data source $DB^{'}$ ($DB^{'} \neq DB$). i.e., $Q^{'} \subseteq Q$, $Q^{''} \subseteq Q$, such that

$$HR(DB, Q^{'}) = HR(DB^{'}, Q^{''}) > 50\%.$$

We demonstrate that

$$OR(DB, Q^{'}) < OR(DB^{'}, Q^{''}).$$

Figure 4.7 shows the results of applying queries to other corpora. For example, the sub figure from the Reuters corpus shows the OR/HR relationship for the queries selected from a sample of Reuters. Those queries are subsequently sent to all the four corpora. The charts show that in three cases queries selected from D_A will perform better in DB_A than other DBs, with the exception of Wikipedia. Wikipedia contains a variety of topics, resulting in samples that are representative for a variety of corpora. Hence the learnt queries work equally well for the other corpora.

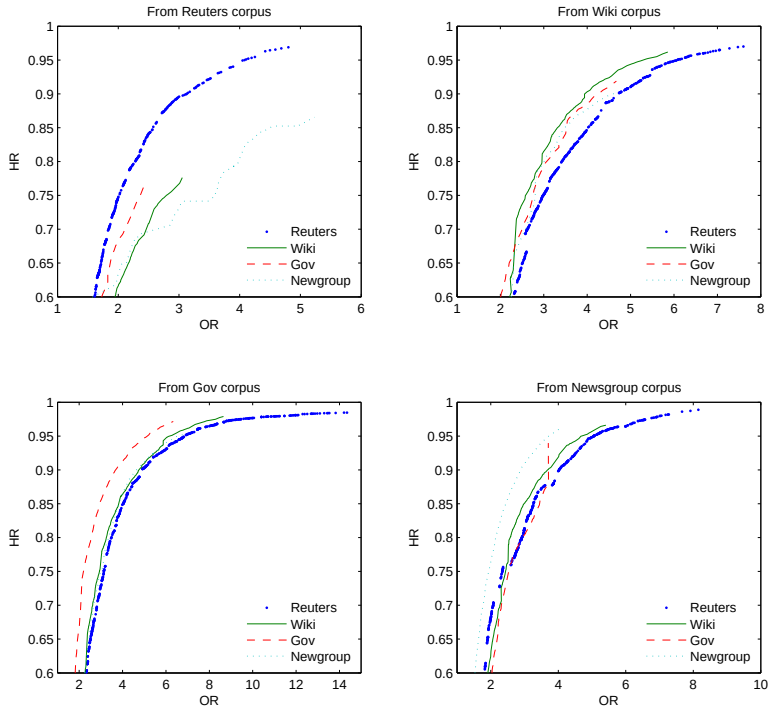

Figure 4.7: Apply queries selected in one corpus to other corpora. Sample size is 3,000, $\mu = 20$. Each sub figure shows the querying results for the four data sources with the queries selected from one particular corpus.

Comparing Figure 4.7 with the random method in Figure 4.3, we find that queries learnt from corpus A can also improve the performance of crawling corpus B, albeit not as good as learning from corpus B directly. An explanation for this is that some groups of words tend to co-occur more often in all corpora. By learning less overlapping words in one corpus, we break up those co-occurrence words and get better results in another corpus.

4.4.5 Comparing queries with Ntoulas' method

One of the elaborate query selection methods is provided by Ntoulas et al in [17], as mentioned in Chapter 2. This method is an incremental crawling method which selects queries from the documents that have been downloaded, and the number of the documents increases as more queries are sent. Their method selects the query

returning most new documents per unit cost iteratively. This can be represented by the following formula

$$\frac{N_{new}(q^k)}{Cost(q^k)} = \frac{P_{new}(q^k) \times |DB|}{c_q + c_r(P(q^k) \times |DB|) + c_d(P_{new}(q^k) \times |DB|)} = \frac{1}{\frac{c_q}{P_{new}} + c_r \frac{P(q^k)}{P_{new}(q^k)} + c_d} \tag{4.3}$$

where

- $N_{new}(q^k)$ is the numbers of the new returned documents by issuing q^k;
- $P(q^k)$ and $P_{new}(q^k)$ denote the fraction of all the returned documents and the new returned documents by issuing q^k to the original data source DB. $P_{new}(q^k) = P(q^k) - P(q_1 \vee \cdots \vee q_{k-1})P(q^k | q_1 \vee \cdots \vee q^{k-1})$ where $\{q_1, \ldots, q^{k-1}\}$ are all the previously selected queries;
- c_q, c_r, and c_d are the costs for sending q^k, retrieving all URLs of matched documents, and downloading all matched documents respectively.

Since there is no prior knowledge of all the document frequencies of the queries in the original data source DB, this method requires the estimate of the document frequency to calculate $P_{new}(q^k)$ based on the documents already downloaded. Because the MLE is used to estimate the document frequency for each term in the current downloaded documents in [17] (the detail is shown in Chapter 5), the selected query needs to maximize $P_{new}(q^k)$ according to equation 4.3. Namely, Ntoulas' method prefers popular terms as queries.

The results of the comparison on the four data sources are shown in Figure 4.8. In this experiment, our method sets the sample size at 3,000, $\mu = 20$, and the range of the sample document frequencies of the queries are between 2 and 500. For Ntoulas' method, the initial query is set to the randomly selected term "good" and all terms whose document frequencies in DB are more than $20\%|DB|$ are considered stop words. The stopping criterion is that no new document is returned after 10 consecutive queries sent.

From Figure 4.8, we find that our method outperforms Ntoulas' method in the four data sources. In addition, our method is much more efficient than Ntoulas' method since it is a sampling-based method rather than an incremental method.

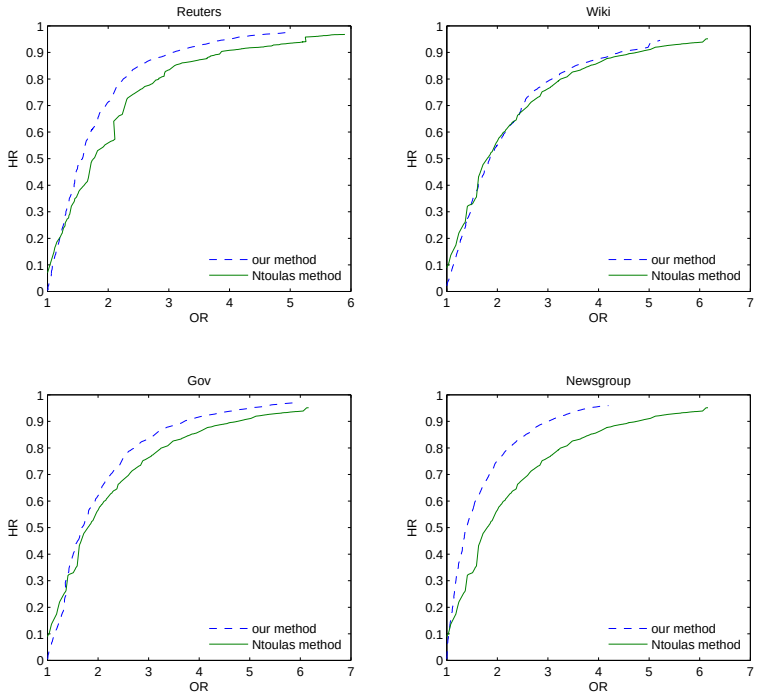

Figure 4.8: Comparison with queries selected by using Ntoulas' method. For our method, the sample size is 3000, $\mu = 20$ and the range of the sample document frequencies of the queries are between 2 and 500. For Ntoulas' method, the stopping criterion is that no new document is returned after 10 consecutive queries sent.

4.4.6 Selecting queries directly from DB

When queries are selected from the greedy algorithm that directly works on the original data source DB instead of a sample data source D, those queries would certainly perform better than our method. To learn whether the sampling method is effective, we would like to know how much better the direct selection method is than our sampling method. Figure 4.9 shows the difference when the queries are selected from the sample data source D and the original data source DB. In this experiment, our method sets the sample size at 4,000, $\mu = 30$, and the range of the document frequencies of the queries as between 2 and 800.

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

4.5 Conclusion

In this chapter, we propose an efficient and effective sampling-based query selection method for deep web crawling. It can retrieve most of the data in a text data source with a low overlapping rate. Our empirical study on the four corpus data shows that, using a sample of around 2,000 documents, we can efficiently select a set of queries that can cover most of the data source with a low cost. We also empirically identified the appropriate size for the sample and the value of the document degree of th query pool.

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

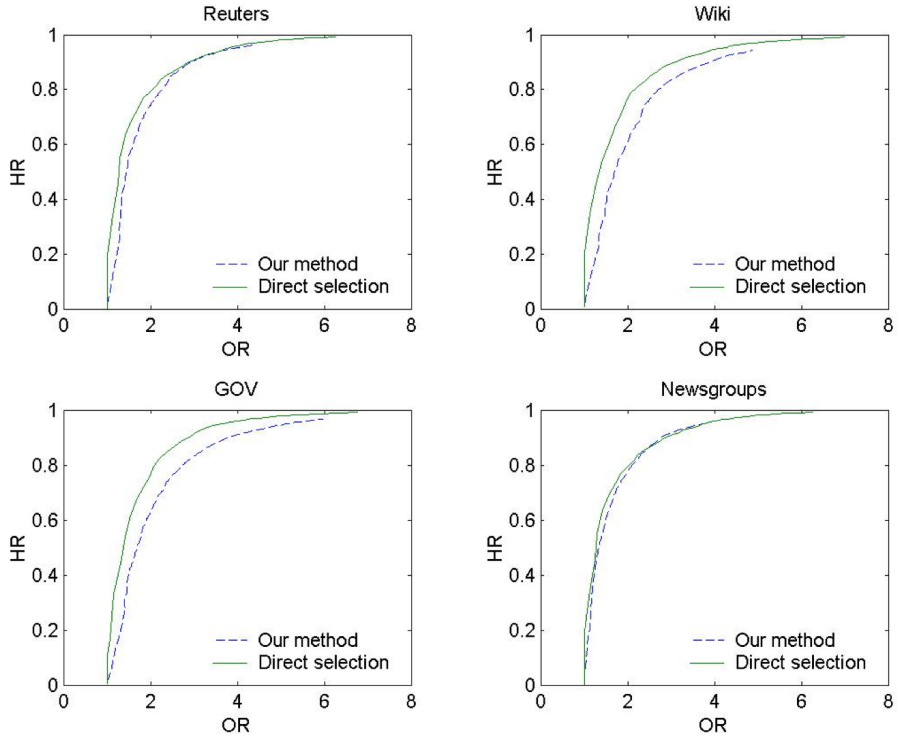

Figure 4.9: Comparison with queries selected directly from DB. Each sub figure shows the querying results for queries selected from one particular data source.

far as we are aware of.

Chapter 5

Ranked data source

5.1 Motivation

The set covering based crawling method introduced in the previous two chapters is based on the assumption that all the matched documents are returned. In the real deep web data sources, especially the large ones, documents are ranked and only the top k of them are returned even if there are more than k number of documents matching a query. For instance, Amazon.com only returns top 500 products (documents) to users by submitting queries at its home page. For this kind of data sources, the set covering approaches no longer works fine because highly ranked documents tend to be retrieved quite often, while lower ranked documents have less probability of being returned even if they are matched.

To explain the phenomenon, we have conducted three experiments which are illustrated in Figure 5.1. The data source is the Reuters corpus that contains 806,791 documents. Each document is assigned a randomly generated static no duplicated ranking value. In each experiment, 100 queries are sent. The return limit k is 20, and the document frequencies F of the queries are 40, 40-80 and 80 respectively for each experiment.

From Figure 5.1, we find that, if queries with large document frequencies, 1) documents ranked low may not be retrieved and thus, high coverage rates cannot be reached; 2) many documents ranked high are repeatedly retrieved, causing high redundancy. Here, document frequencies F are considered *large* compared to the return limit k. For example, if $k = 20$, $F = 40$ is considered large.

Figure 5.1(B) shows the ranks of the retrieved documents when $k = 20$ and 100

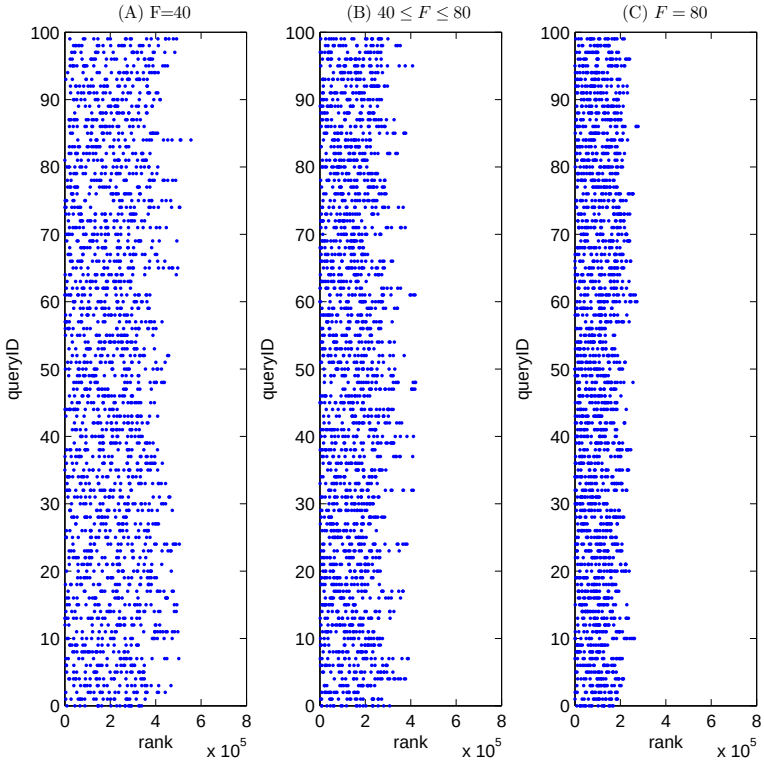

Figure 5.1: Scatter plots for the query results from different types of queries. For each experiment, 100 queries are sent. X axis represents the document rank and the return limit $k = 20$. Sub figure (A) queries with $F = 40$; (B) queries with $40 \leq F \leq 80$; (C) queries with $F = 80$. The data source is the Reuters.

queries with $40 \leq F \leq 80$ are sent. As we can see from this figure, documents with ranks between 430,000 and 806,791 are not retrieved at all. In addition, according to the data of this experiment, documents with ranking values between 0 and 410,000 are retrieved 1991 times.

More precisely, when the document frequencies of all the queries in Q are greater than k, the range of the document ranks of those that can be retrieved by the queries in Q can be determined by the range of the document frequencies of Q. This can be approximately expressed by the following formula:

$$max\{M_q|q \in Q\} \approx \frac{k}{min\{F(q)|q \in Q\}} \times |DB|$$

where M_q is the maximum rank of the documents returned by issuing q. $|DB|$ is the size of the data source. *F(q)* denotes the document frequency of the query q in the data source DB.

In the first experiment (Figure 5.1(A)),

$k = 20$,

$min\{F(q)|q \in Q\} = 40$,

$N = 806791$.

So we have $max\{M_q|q \in Q\} \approx (20/40) * 806791$. Thus, the biggest rank of the returned documents should be around 403,396. Most of the documents whose ranks are bigger than this are not retrieved by these terms.

Similarly, in the third experiment (Figure 5.1(C)),

$k = 20$,

$min\{F(q)|q \in Q\} = 80$,

$N = 806,791$.

So we have $max\{M_q|q \in Q\} \approx (20/80) * 806,791$. Thus, the biggest rank of the returned documents should be approximately 201,698. Most of documents whose ranks are bigger than this are hardly retrieved by these terms.

The above formula defines a limit on the size of the documents that can be returned by Q. It was given in [55] a special case of this formula when the document frequencies of all the queries are the same and the documents are ranked according to their document degrees.

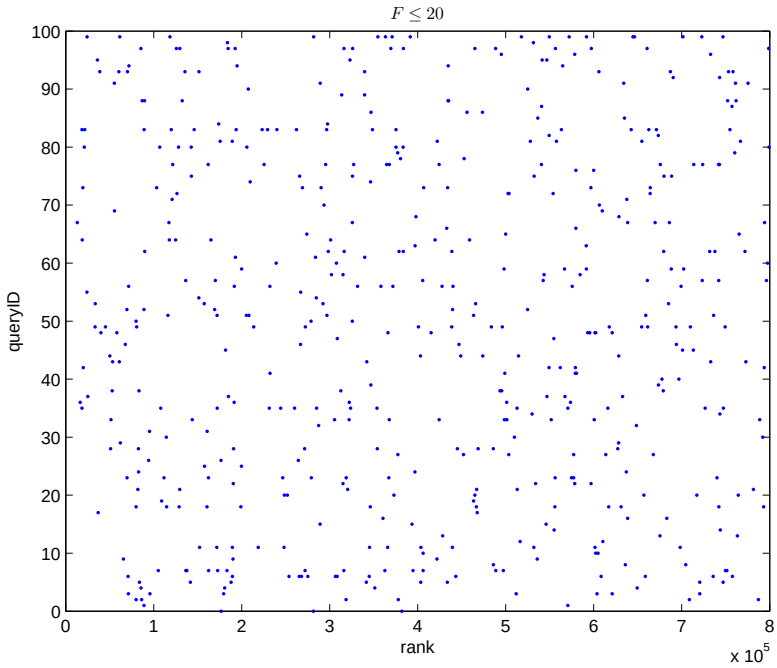

Figure 5.2: The scatter plot for the query result. 100 queries whose $F \leq 20$ are sent and the return limit $k = 20$.

Contrast to the three experiments shown in Figure 5.1, we have conducted another experiment with the same conditions, as shown in Figure 5.2. In this experiment, 100 queries whose document frequencies are less than the return limit $k = 20$ are sent. It can be seen that, compared to Figure 5.1, the ranks of the retrieved documents can reach the whole range of the rank values.

Thus, we say, for a query with a small document frequency, the documents matched by it are not affected by the return limit and ranking criteria. Here, the document frequencies are considered *small* compared to the return limit k. Any numbers smaller than k, for example, can be considered small. Any numbers not much bigger than k can also be considered small. As a consequence, if there are enough queries with small dfs, most of the documents inside a ranked data source can be returned.

Similar experiments were conducted previously [55] where the documents are ranked by their document degrees and there is no consideration of the return limit k.

According to the above observations, we can draw the following conclusions as guidelines for the query selection for ranked data sources.

- *To reach a high coverage rate, the set of selected queries should not contain only of those with large document frequencies.*
  This is because, according to the observation on Figure 5.1, some documents ranked low can rarely be returned by the queries with large document frequencies, at the same time, the queries with large document frequencies very often bring back the same documents ranked high.

- *We can reach a much better coverage rate by using the queries with small document frequencies.*
  This is straightforward from the observation on Figure 5.2.

The proposed query selection technique is based on the above guidelines. It is introduced in the next section.

5.2 Our crawling method

Our query selection method is shown and illustrated in Algorithm 5 and Figure 5.3. It consists of four steps.

Algorithm 5: Query selection for ranked data sources.

Input: the original data source DB, the sample size m and the return limit k.
Output: a collection of URLs S.
Process:

```
1       D = retrieveDocs(DB, m);
2       QP = retrieveTerms(D);
3       foreach q in QP
            if F̂(q) ≤ k
                add q into QP_k;
4       while(!requirment){
            randomly select q_u ∈ QP_k as query;
            S = S + retrieveURLs(DB, q_u);
        }
```

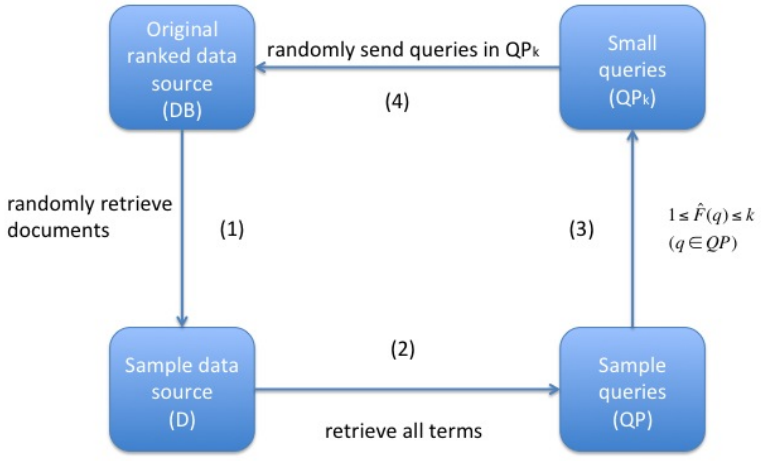

Figure 5.3: Our query selection method for ranked deep web data sources

Our method is sample-based: some documents are randomly extracted from the original ranked data source DB (Step 1). These sample documents form the sample data source D. The size of the sample data source should meet the requirement which ensures that there are enough queries to cover DB. Our algorithm runs on the sample to generate a set of queries which are then mapped into the original data source.

From the sample data source, we retrieve all the terms to obtain the set of terms QP for selection (Step 2).

According to the discussions in the previous section, we can choose queries only from among QP with small document frequencies and can reach a high coverage rate. Other than the return limit k, the limit on the document frequencies to be considered *small* depends on the given coverage rate as well. It is hard and beyond the scope of the present work to find such a limit. Here we assume that by using all the queries with document frequencies less than k they can cover most of the original data source, and we select from among these queries to reach a given coverage rate.

Note that most of the web-access databases do not provide the document fre-

quency for each query inside it, and it is time-consuming to obtain the document frequencies of all the queries in QP by issuing them. Here we use three different df estimators to estimate the document frequencies of all terms in QP. The detail is shown in the following section.

Let QP_k denote the subset of QP which contains all the terms in QP whose estimated document frequencies are no greater than k.

In Step 4, we randomly select queries from QP_k sequentially until the given coverage rate is reached.

5.3 Frequency estimation

5.3.1 Introduction

Maximum Likelihood Estimation

Document frequency estimation refers to estimating the document frequencies of the terms in a given sample D.

The most straightforward estimate is the *Maximum Likelihood Estimation* (MLE). Simply speaking, it enlarges the sample df of each term $f(q_j)$ proportionally and the ratio is $\frac{|DB|}{|D|}$. More precisely, given a sample data source D derived from one original data source DB, which contains m documents and n terms, for each term q_j $(1 \leq j \leq n)$, its probability p_j is the ratio of its document frequency $F(q_j)$ to the sum of the document frequencies of all the terms in DB and defined in equation 5.1:

$$p_j = \frac{F(q_j)}{\sum_{q \in DB} F(q)} \tag{5.1}$$

Then, p_j can be estimated by the ratio of the sample document frequency of q_j to the sum of the sample document frequencies of all the terms in D in the MLE. It is shown in equation 5.2

$$\widehat{p_j} = \frac{f(q_j)}{\sum_{q \in D} f(q)} \tag{5.2}$$

If it is supposed that the average document size of D is the same as that of DB, after

given $|DB|$, the estimation function of the MLE can be represented by equation 5.3.

$$\widehat{F}(q_j) = \widehat{p}_j \times \sum_{q \in DB} F(q) = f(q_j) \times \frac{|DB|}{|D|} \tag{5.3}$$

where $\widehat{F}(q_j)$ is the estimated document frequency for q_j. The detail of the MLE is shown in Appendix C.

Simple Good-Turing Estimation

According to equation 5.1 and equation 5.2, all estimated probabilities of all the terms in D are added up to "1" ($\sum_{q \in D} \widehat{p}(q) = 1$) and simultaneously the sum of the probabilities of all the terms in DB is "1" as well ($\sum_{q \in DB} p(q) = 1$). Thus, two problems remain: 1) in the MLE, there is no probability mass for "unseen" terms in DB not in D; 2) the MLE will give overestimates for the terms in D.

Smoothing techniques [56, 57, 58, 59, 60, 61, 62] can be used to address the two problems. The term *smoothing* describes the techniques to produce more accurate estimated probabilities by adjusting the maximum likelihood estimate of the probabilities. More precisely, these techniques tend to adjust low estimated probabilities upward and high estimated probabilities downward.

The Simple Good-Turing method [28] (SGT) is one of the smoothing techniques and the basis of many other smoothing estimators [63, 64]. Its basic idea applied to our problem is adjusting the document frequency of each term in D. It can be represented as equation 5.4. For each f, we have

$$f^* = (f+1) \times \frac{n_{f+1}}{n_f} \tag{5.4}$$

where f^* is the adjusted sample frequency, n_f and n_{f+1} are the numbers of the terms whose sample document frequencies are f and $f+1$ respectively.

With the adjusted frequency, the adjusted probability of each term p_j^* is defined in equation 5.5.

$$p_j^* = \frac{f^*(q_j)}{\sum_{q \in D} f(q)} \tag{5.5}$$

There are two expectations for equation 5.5. One is that $p^* < p$ and the other is that the ratio $\frac{p^*}{p}$ to increase to one as df increases. The first expectation is to cut down the estimated probability of each term in D and it is accomplished by

decreasing its sample document frequencies. In common sense, the high frequency terms usually are easier to obtain the more accurate estimates. Therefore, the second expectation is to take less and less probability away as sample document frequency increases.

Equation 5.4 requires that the sample document frequencies must be continuous. However, because of the sparse sample data, the requirement could be hard to satisfy, especially for the high frequency. Even though all frequencies are continuous, for high dfs, their corresponding n_f could be the same with each other and then it will lead to $f^* > f$. To deal with the problem caused by the sparse data, a linear smoothing algorithm is given in [28] and it uses a new variable Z_f to replace n_f in equation 5.4, then we have a new equation for the adjusted sample document frequency shown in equation 5.6. For f,

$$f^* = (f+1) \times \frac{Z_{f''}}{Z_{f'}} \tag{5.6}$$

where $Z_f = \frac{2n_f}{f''-f'}$, f' is the nearest lower sample document frequency and f'' is the nearest higher sample document frequency such that both of $n_{f'}$ and $n_{f''}$ are rather than zero. For low f, f' and f'' will be immediately adjacent to f, so that $f'' - f'$ will be 2 and Z_f will be the same as n_f; for high f, Z_f will be a fraction, sometimes a small fraction.

Now, the detail of the SGT method applied to estimate df is shown in Algorithm 6.

In Step 1, for calculating each Z_f, if $f = 1$, let f' be 0; if f is the largest frequency, f'' is set to $2f - f'$.

In Step 2, here we use the off-the-shelf curve-fitting algorithms provided by Matlab curve-fitting toolbox [65].

In Step 3, there are one inequation and two values x and y defined as follows:

$$X = (f+1)\frac{n_{f+1}}{n_f}. \tag{5.7}$$

$$Y = (f+1)\frac{S(f+1)}{S(f)}. \tag{5.8}$$

$$|X - Y| > 1.96 \times \sqrt{(f+1)^2 \frac{n_{f+1}}{n_f}(1 + \frac{n_{f+1}}{n_f})}. \tag{5.9}$$

Algorithm 6: SGT applied to df estimation

Input: a sample data source D (containing m documents and n terms) and the size of the original data source $|DB|$.

Output: a set of estimated document frequencies for all terms in D.

1. Calculate $Z_f = \frac{2n_f}{f''-f'}$ for each available sample document frequency f where n_f is the total number of the terms that occur f times in D and f' (f'') is the nearest lower (higher) sample document frequency to f;

2. Find the line of best fit ($y = A + \alpha * log(x)$) to all pairs ($log(f)$, $log(Z_f)$) using a curve fitting algorithm where A and α are parameters;

3. Calculate f^* for all various f beginning with $f = 1$ in accordance with different rules, $f^* = (f+1)\frac{n_{f+1}}{n_f}$ if inequation 5.9 is satisfied, otherwise, $f^* = (f+1)\frac{S(f+1)}{S(f)}$ where $S(f)$ is the antilog function of $y = A + \alpha * log(x)$;

4. Calculate $\widehat{F}(q_j) = f^*(q_j) \times (\frac{|DB|}{m})$ for each term, where $f^*(q_j)$ is the adjusted frequency of the term q_j.

If the inequality 5.9 holds, then let f^* be the value of X; if the inequality does not hold, f^* is assigned to the value of Y and cease to calculate X values for all subsequent terms, and their adjusted sample document frequencies always are the values of Y. In equation 5.8, $S(f)$ is the antilog function of $y = A + \alpha * log(x)$.

Example 4 *In this example, the original data DB is a small size Gov corpus, it contains 2975 documents($|DB| = 2975$) and 4604 terms. The sample D contains 593 documents (m=593) and 2535 terms ($n = 2535$), and the sum of the sample dfs of all terms is 19,065. Table B.1 shows the information f, n_f, Z_f, $log10(f)$, $log10(Z_f)$ and f^* in ascending order of f values. Figure 5.4 plots n_f against f. Figure 5.5 shows Z_f again r and the line best fit ($y = 3.121 - 1.707 * log10(x)$) superimposed on all pairs (f, $log(Z_f)$). Both of figures are in log-log scales. According to Table B.1, we can calculate the estimated document frequency for each term. For example, the estimated document frequency of the term "abby" whose sample document frequency is 2 can be calculated by $\widehat{F}(\text{"}abby\text{"}) = f^*(q_j) \times \frac{|DB|}{m} = 1.62 \times \frac{2975}{593}$ = 8.12 (its df is 4).*

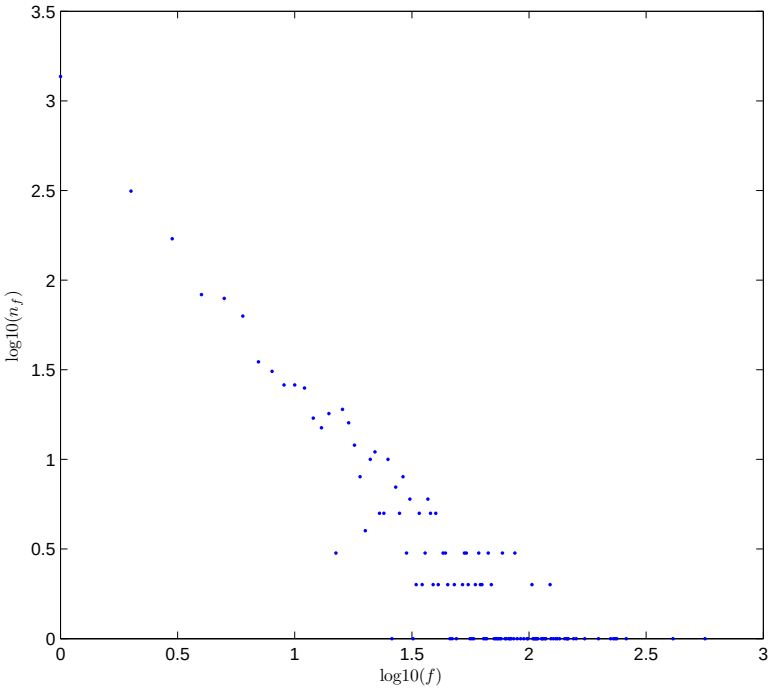

Figure 5.4: The trend of sample document frequency in Example 4 on log-log scales. High sample document frequencies becomes horizontal along the line n_1.

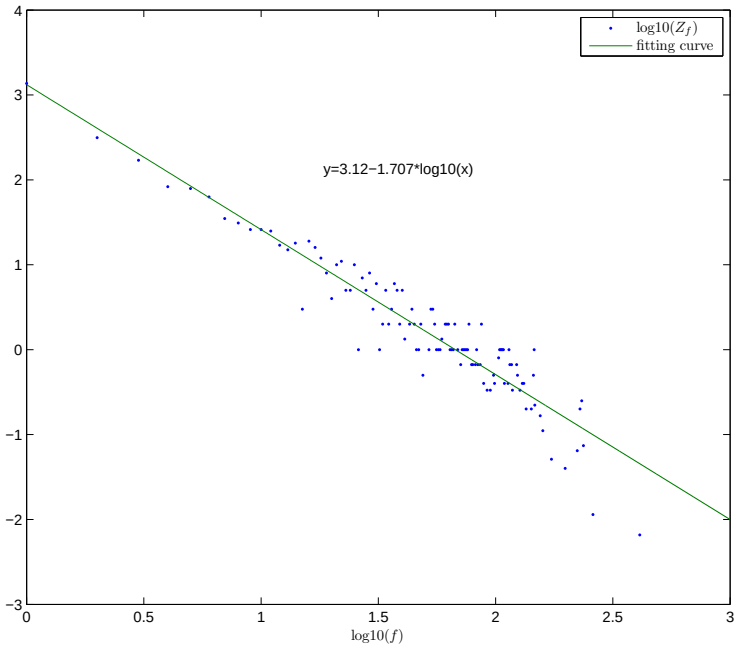

Figure 5.5: The smoothing result of n_f shown in Figure 5.4 for our SGT example on log-log scales. $Z_f = \frac{2n_f}{f''-f'}$ and the line of best fit $y = 3.12 - 1.707 \times log_{10}(x)$ is obtained by using the least-squares curve fitting algorithm provided in Matlab.

Zipf's-law-based estimation

Compared to the MLE and the SGT, the Zipf's-law-based estimate is a different technique to address df estimation problem by using Zipf's law that is an empirical linguistic law. Ipeirotis et al. [29] first presented a Zipf's-law-based estimation method to estimate document frequency.

In 1932, George Zipf put forward an empirical observation on certain statistical regularities of human writings that has become the most well-known statement of quantitative linguistics. It states that, given a certain corpus of a natural language, the *occurrence frequency* of any word is approximately inversely proportional to its rank in the frequency list and it can be described as follows:

$$F(r) = \frac{A}{r^{\alpha}},$$

where A is a constant and the characteristic exponent α takes on a value slightly greater than 1. Later, in [66], Mandelbrot slightly modified Zipf's law to improve its performance and the function is shown here:

$$F(r) = \frac{A}{(r+b)^{\alpha}},$$

where A, b and α are constants. The Zipf-Mandelbrot law can more precisely describe the relationship between the ranks and frequencies of the terms ranked high ($r < 100$).

According to the Zipf-Mandelbrot law, the basic idea of this estimation method is that

- exploiting the sample document frequencies f derived from the sample D to rank all terms from most frequency to least frequency;
- exploiting the document frequencies of probing queries to potentially boost the document frequencies of "nearby" terms for which we only know their sample document frequency but not their document frequency in DB.

In a word, the method tries to "propagate" the known document frequencies to "nearby" terms with similar sample document frequencies according to Zipf-Mandelbrot law. Figure 5.6 illustrates the basic idea of Ipeirotis' estimator.

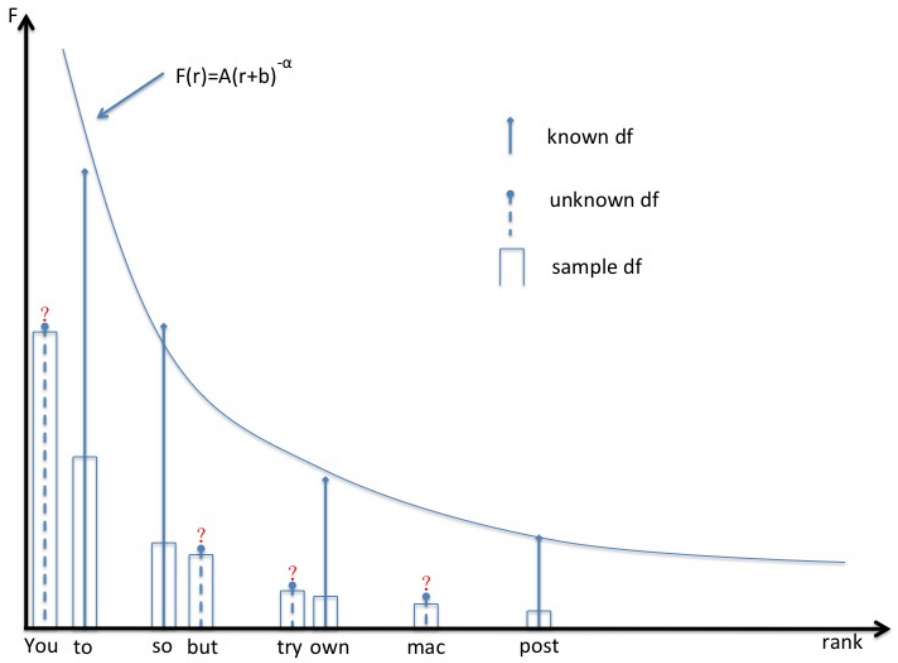

Figure 5.6: An artificial example to illustrate the basic idea of Ipeirotis' estimator.

Although the Zipf's-law-based estimator in [29] is a novel method to address df estimation problem, the authors did not expose the detail of their estimation method (such as how to select probing queries) and no solid estimate results were provided. According to the key idea in [29], here it is implemented in Algorithm 7.

In step 3, although intuitively more probing queries are sent and better results for curve fitting are produced, sending many probing queries for their document frequencies could be time-consuming in reality. Here empirically we set $h = 30$ for our experiments. and all selected queries should have different ranks in D.

In step 4, the curve-fitting toolbox of Matlab is used to do regression and it makes use of the least-squares formulation to fit our sample data.

Example 5 *Example 4 is used to show how the Zipf's-law-based estimation method works. Figure 5.7 shows each sample data, 30 probing data and the generated fitting curve in log-log scales. X axis represents the ranks in the sample D ($m = 593$, $n = 2535$) and y axis represents the document frequency in DB ($|DB| = 2975$). From this figure, for sample data, the terms ranked low can have many different F (dotted vertical lines) because of a lack of sufficient statistics in D, on the other side,*

Algorithm 7: The Zipf's-law-based document frequency estimation

Input: DB(original data source), m(sample size), and h(probing query number).

Output: A set of estimated document frequencies for all terms in D.

1. Randomly collect from DB a sample data source $D = \{d_1, ..., d_m\}$;
2. Rank all the terms in the query pool QP derived from D in descending order according to their document frequencies in D;
3. Randomly select h number of probing queries from QP and send them to DB to obtain their document frequencies in DB;
4. Based on the ranks in D and the document frequencies of the issued queries, calculate the values for all parameters A, b and α of the estimation function $\widehat{F}(r) = \frac{A}{(r+b)^{\alpha}}$ by running the least-squares curve fitting algorithm provided in Matlab [65];
5. Based on the estimation function $\widehat{F}(r)$ and the ranks of all the terms in QP, a set of the estimated document frequencies for all the terms are calculated and returned.

the terms ranked high are much better. For probing data, most of them are the terms ranked low due to the power law distribution [67], which states that most of terms are low frequency terms. For the generated curve, it fits the terms with moderate and low frequencies better than the high frequency terms because most of probing terms are from the low frequency terms. According to the fitting curve $F(r) = \frac{10^{6.334}}{(r+64.66)^{1.892}}$, *we can calculate the estimated document frequency for each term. For example, the estimated document frequency of the term "abby" whose sample document frequency is 2 and its rank is 1852 can be calculated by* $\widehat{F}(r) = \frac{10^{6.334}}{(1011+64.66)^{1.892}} \approx 3.963$ *(its df is 4).*

In the following section, we will evaluate the performances of the three estimation methods on the four corpus data sources.

5.3.2 Evaluation of the estimation methods

In this section, we run our experiments on the same data as the ones in Chapter 4 to show the performances of the three estimators. The only difference is that the

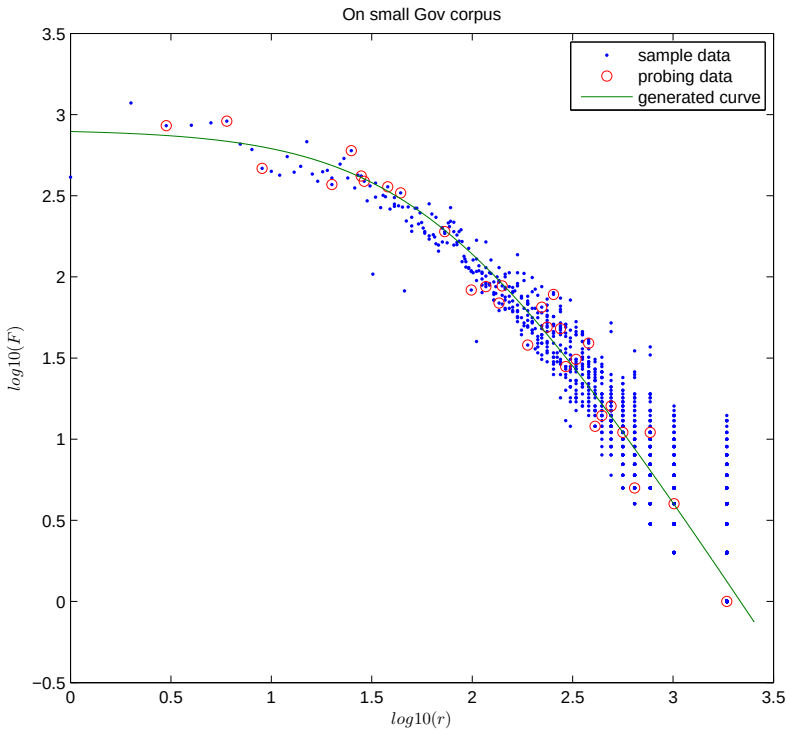

Figure 5.7: The example of the Zipf-law-based estimation method.

Table 5.1: The statistics of the samples for the experiments (m:the number of the documents in D, n: the number of the terms in D, r_1:$\frac{m}{|DB|}$, r_2: the ratio of the number of all the terms in D to the number of all the terms in DB).

corpus	m	$avg(n)$	$avg(r_1)$	$avg(r_2)$
Reuters	3000	30894	0.00372	0.0081
Wiki	3000	103296	0.00219	0.0156
Gov	3000	83422	0.00300	0.0415
Newsgroup	3000	91401	0.00218	0.0303

Table 5.2: The average of the parameter values for the Zipf-law-based estimation and SGT based on 10 3000-document samples.

corpus	Zipf			SGT	
	A	b	α	A	α
Reuters	$10^{9.75}$	613	-1.71	4.607	-1.869
Wiki	$10^{13.2}$	2953	-2.34	5.060	-2.007
Gov	$10^{15.73}$	4029	-2.90	5.014	-1.874
Newsgroup	$10^{12.44}$	1907	-2.18	4.909	-1.901

Newsgroup corpus contains 1,372,911 not 30,000 documents. In our experiments, there are 10 sample data sources randomly retrieved from each original data source and their size m are 3000. The statistics of the samples are shown in Table 5.1.

Table 5.2 shows the average of the parameter values for the Zipf-law-based estimation and SGT based on 10 sample data sources derived from one particular original data source.

Since the document frequencies and their estimated ones can vary by several orders of magnitude, it is convenient to express the error in the logarithm of the ratio of an estimated document frequency to its document frequency. To measure the overall error of a set of given terms in D, the root mean square of the base-10 logarithms of their ratios is used, which is called *Average Error* defined in [28] and shown in Definition 8.

Definition 8 ***(Average Error)*** *Given a set of terms $Q = \{q_1, ..., q_n\}$ from D, its average error can be calculated by equation 5.10*

$$AE(Q) = \sqrt{\frac{\sum_{j=1}^{n}(lg\widehat{F}(q_j) - lgF(q_j))^2}{n}}, \tag{5.10}$$

Table 5.3: The average errors for three estimators according to different corpus samples and sample df ranges.

Ranges	Methods	Reuters	Wiki	Gov	Newsgroup
$1 \leq f \leq 10$	MLE	0.797	1.277	0.923	1.123
	SGT	0.676	1.003	0.757	0.852
	Zipf	0.708	1.008	0.842	1.014
$10 < f < 500$	MLE	0.093	0.096	0.087	0.090
	SGT	0.090	0.092	0.086	0.088
	Zipf	0.103	0.103	0.092	0.090
$500 \leq f$	MLE	0.013	0.014	0.013	0.011
	SGT	0.013	0.014	0.013	0.011
	Zipf	0.565	0.181	0.084	0.221
$1 \leq f$	MLE	0.740	1.220	0.855	1.065
	SGT	0.629	0.958	0.702	0.808
	Zipf	0.659	0.963	0.781	0.963

where lg is the base-10 logarithm, $F(q_j)$ and $\widehat{F}(q_j)$ are the document frequencies and estimated document frequency of the term q_j ($q_j \in Q$).

According to Definition 8, we separately calculate the average errors based on the different sample df ranges shown in Table 5.3. In Table 5.3, for each corpus, all terms inside a 3000-document sample are used to calculate the average errors but they are divided into four different groups. Here the first three groups represent high, moderate and low frequency terms respectively and the last one stands for an overall result. From the table, we find that

- for all estimators, the accuracy of their estimates is improved while f increases;
- for the SGT estimation, it achieves the best results in all kinds of the ranges and particularly outperforms the others at the low frequency terms ($f < 10$);
- for the MLE, compared to the others, it has the worst performance on the low frequency terms ($f < 10$) but the good results on the moderate and the high frequency terms, especially for the high frequency, it can obtain the same results as the ones from the SGT;
- for Zipf's-law-based estimation, it has the reversed performance of the MLE and is good at estimating the low frequency not the high frequency terms.

The results from the three estimators are not unexpected. As mentioned before, the MLE is notorious for overestimating the low frequency terms, not the high frequency ones and the SGT is designed to correct such a problem by cutting down the estimated probability of each term. Since the SGT takes less and less probability away as the sample document frequency increases, we find the SGT is much better than the MLE for the low frequency terms but has the same performance as the MLE for the high frequency terms. The Zipf's-law-based estimation is good at estimating the low frequency terms since the generated curve ($\widehat{F} = \frac{A}{(r+b)^{\alpha}}$) fits the moderate-low frequency terms better than the high frequency terms. In fact, recent studies [68, 69, 70, 71, 72] show that the Zipf-Mandelbrot law is not fit for web-scale corpus and the terms inside a large corpus should be divided into different groups.

In common, it is much easier to have accurate estimates on the high frequent terms than the low frequent terms and thus the performances of the three estimators in small f are displayed in more detail in Figure 5.8. In this figure, the x and y axes represent the sample document frequency and the average error.

Average Error successfully measures the performances of the three estimators. Namely, it shows how accurate the estimated document frequencies are. However, it cannot demonstrate the overestimation and underestimation of the estimated document frequencies. Thus, Figure 5.9 shows the average of the document frequencies and estimated document frequencies of the three estimators for the low frequency terms. From the figure, we can see that

- the MLE gives much overestimated results especially when $f < 5$;
- the estimates of the SGT are closest to the averages of dfs and particularly the results with $f = 1, 2$ are almost same as the true values;
- the Zipf's-law-based estimator usually underestimates the low frequency terms except $f = 1$.

Finally, we randomly select 20 terms from one Newsgroup 3000-document sample as concrete examples to illustrate the performance of the three estimation methods shown in Table 5.4.

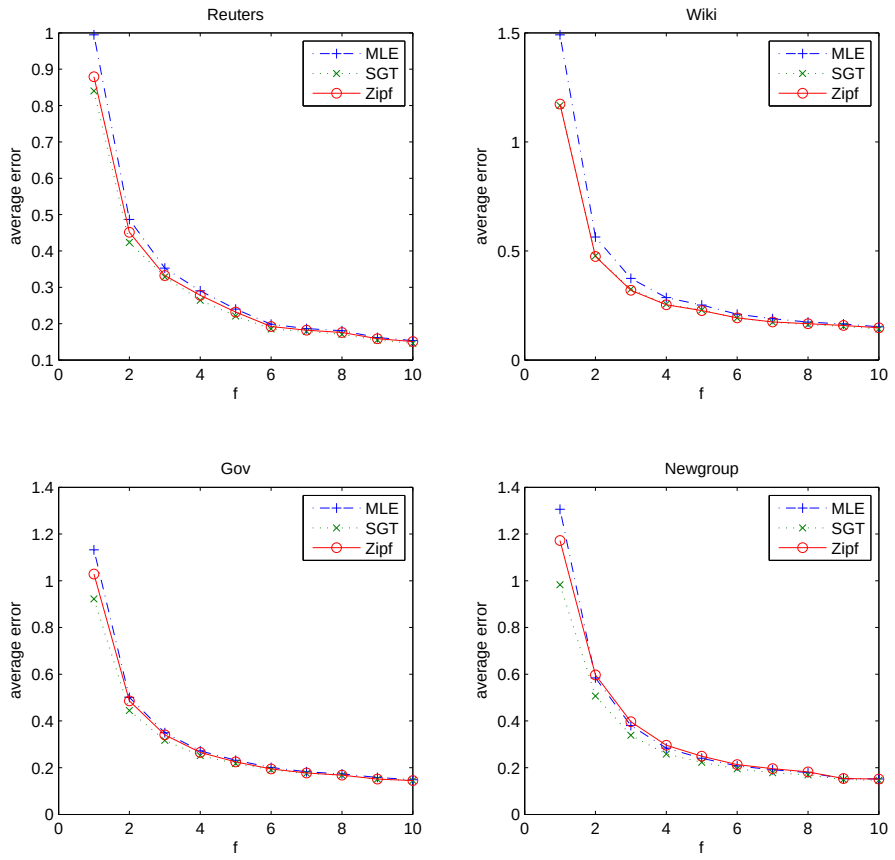

Figure 5.8: The average errors of the three estimators for the terms with $1 \leq f \leq 10$ based on four different corpus samples.

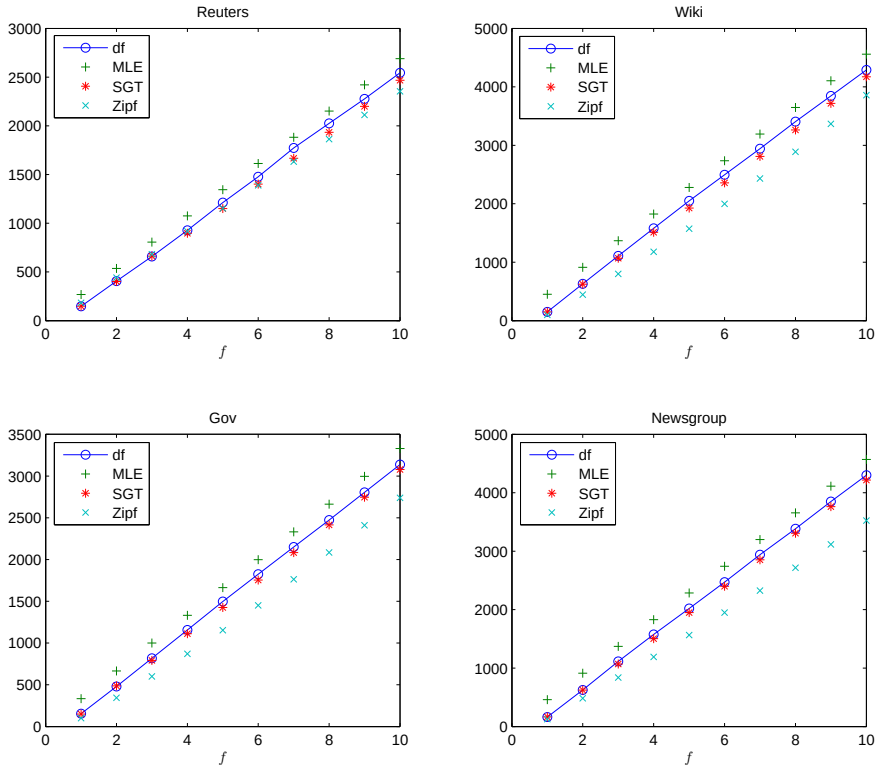

Figure 5.9: The average of the document frequencies and the estimated document frequencies of the three estimators for the low frequency terms based on four 3000-document samples. The MLE: the average of estimated document frequencies from the MLE, the SGT: the average of estimated dfs from the SGT, Zipf: the average estimated dfs from the Zipf's-law-based estimation.

Table 5.4: The results of twenty randomly selected terms from a Newsgroup 3000-document sample (f: sample document frequency).

terms	f	F	$\widehat{F}$		
			MLE	SGT	$Zipf$
integration	43	19979	19651	19342	18507
arrange	10	5754	4570	4216	4568
aeons	3	302	1368	1064	1470
stupor	3	935	1371	1064	1470
oneshould	2	263	914	638	959
muche	2	640	914	638	959
prisonexperience	1	5	457	156	304
phineas	1	573	457	156	304
polevoy	1	151	457	156	304
tensionhid	1	4	457	156	304
teamis	1	69	457	156	304
songentitled	1	37	457	156	304
compassionately	1	418	457	156	304
armourand	1	18	457	156	304
aclosed	1	166	457	156	304
alcornwhose	1	5	457	156	304
madeof	1	176	457	156	304
lastline	1	24	457	156	304
haganot	1	39	457	156	304
jewsbut	1	127	457	156	304

5.4 Crawling evaluation

To evaluate the performance of our query selection method, we compare it with the following two existing ones: the *random method* and the *popular word method* [16, 17].

1. *random method*: we use a middle-sized Webster English dictionary with 51,541 words and randomly select them as queries from it;

2. *popular word method*: given the document frequencies of all the terms in D, we directly collect all the high frequency terms whose document frequencies are higher than $2k$. However, we do not use stop words. Then we randomly pick these queries one by one to send to DB;

3. *our df-based method*: each term in D whose $\widehat{F}(q) \leq k$ is sent to DB randomly until the given coverage rate is reached or all the terms are used up. Here $\widehat{F}(q)$ is calculated by the three estimation methods respectively.

We have run these methods on four different 3000-document sample data sources obtained from the four different ranked data sources. For each experiment, the return limit k is set to 1000. Each document in a corpus has a unique randomly generated rank value. A document with a smaller rank value is ranked higher than a document with bigger a rank value.

Table 5.5 shows the number of the candidate queries of the three methods in one sample derived from one certain corpus data source. From this table, we can see that df-based method with different estimation methods could have exactly the same candidate queries in QP_k, such as, for Gov and Newsgroup sample data sources, the three estimation methods select totally same terms into QP_k. Thus, for our method with the same candidate terms in QP_k, only one experiment is carried out.

From Figure 5.10, we can see that, 1) in the presence of the return limit, the proposed method with any of the three estimation methods is much better than the *random method*, which is better than the *popular word method*; 2) the results of our df-based method with different estimates are close to each other.

Note that all result curves look much more smoother than the other results, such as Figure 4.9. The reason is that, due to the return limit, the improvement of HR

Table 5.5: The number of the candidate terms of the three methods in each experiment.

corpus	random method	popular word method	our method		
			MLE	SGT	Zipf
Reuters	51541	5441	22330	23744	23744
Wiki	51541	16030	78405	78405	83575
Gov	51541	15476	56371	56371	56371
Newsgroup	51541	15478	67860	67860	67860

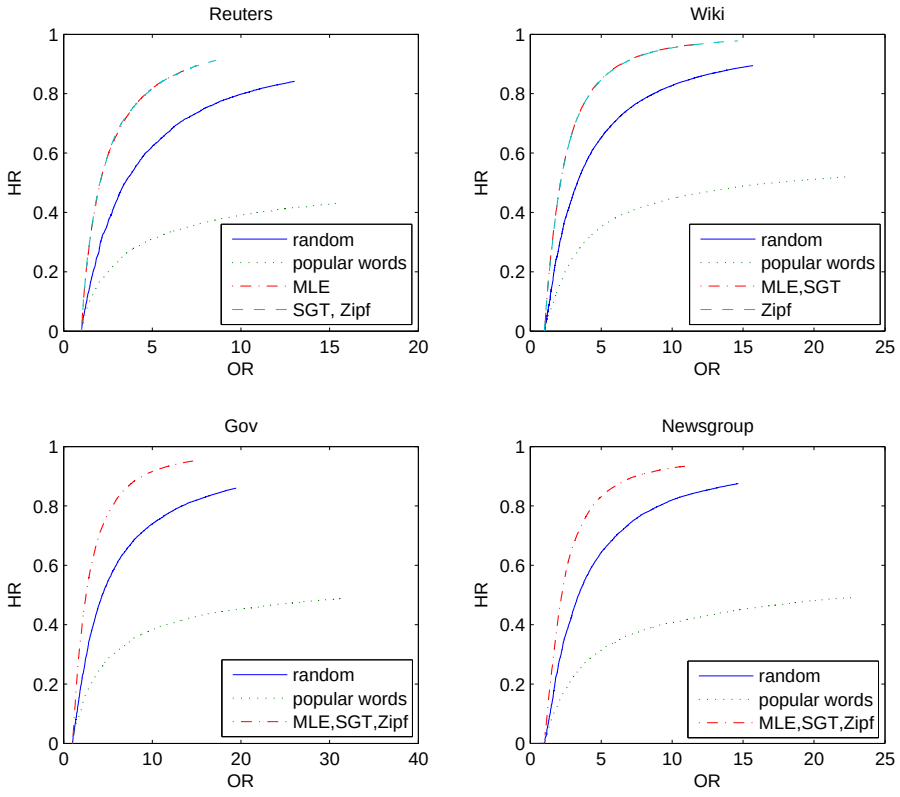

Figure 5.10: The results of the three methods on different corpora. The return limit k is 1000. Documents are statically ranked. MLE, SGT, Zipf: our methods with the MLE, the SGT and the Zipf-law-based estimation methods respectively.

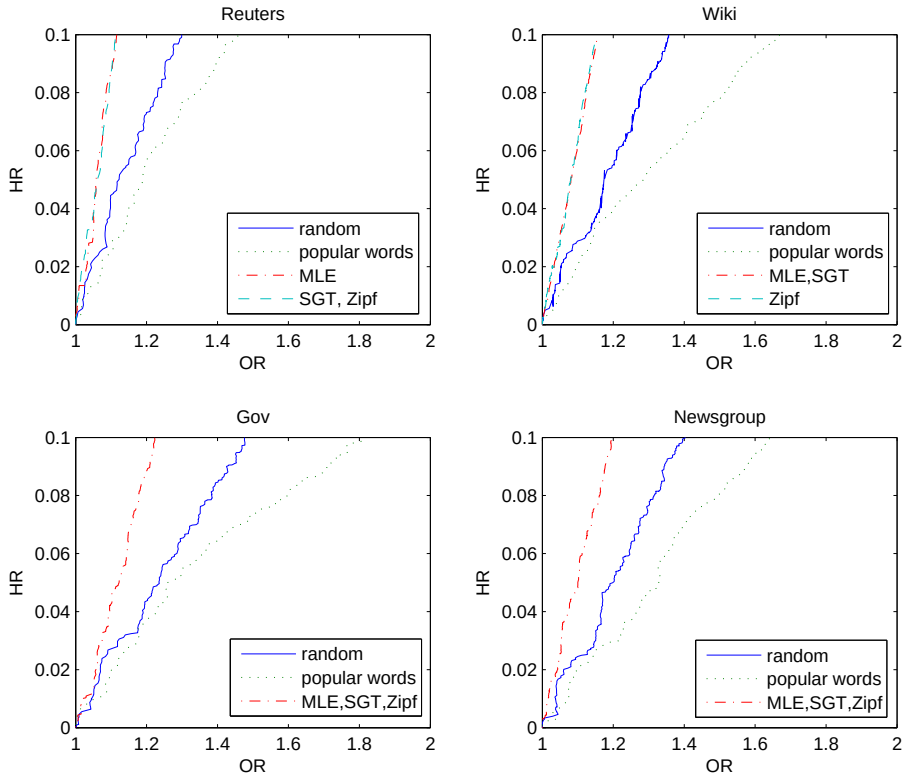

Figure 5.11: The zoomed-in area of each subgraph in Figure 5.10. The range of HR is up to 10%.

at each iteration could be very low (maximum 1000 new returned documents) and thus the 80% coverages of all results are based on a large number of sent queries issued. Figure 5.11 shows a zoomed-in area of each subgraph in Figure 5.10 and the fluctuation of each result is demonstrated.

Since the results of our df-based method with different estimates are close to each other, Table 5.6 shows the comparison of the results of the random method, the popular word method and our method with the MLE. From Table 5.6, we can see the difference between the performance of our method compared to the other two:

- The *random method* can also reach a high coverage rate as ours, but with 85% coverage rate, our method gives around 58% savings on the overlapping rate.

Table 5.6: Comparison of the three methods ($Imp = \frac{OR_{random}(OR_{popular}) - OR_{ours}}{OR_{random}(OR_{popular})}$).

corpus	$HR(\%)$	Our method	Random method		Popular word method	
		OR	OR	Imp(%)	OR	Imp(%)
Reuters	10	1.11	1.20	8.3	1.47	24.4
	20	1.26	1.48	14.8	2.56	50.7
	30	1.47	1.87	21.3	4.78	69.2
	40	1.73	2.40	27.9	12.55	86.2
	42	1.79	2.49	28.1	16.08	88.8
	50	2.06	2.96	30.4	-	-
	60	2.54	3.87	34.3	-	-
	70	3.33	5.33	37.5	-	-
	80	4.65	8.21	43.3	-	-
	84	5.53	12.87	57.0	-	-
Wiki	10	1.15	1.32	12.8	1.67	31.1
	20	1.33	1.65	19.3	2.55	47.8
	30	1.55	2.13	27.2	4.52	65.7
	40	1.80	2.68	32.8	8.77	79.4
	50	2.12	3.40	37.6	24.48	91.3
	60	2.54	4.41	42.4	-	-
	70	3.17	6.04	47.5	-	-
	80	4.19	9.04	53.6	-	-
	89	6.05	15.09	59.9	-	-
Gov2	10	1.22	1.36	10.2	1.81	32.5
	20	1.45	1.84	21.1	3.13	53.6
	30	1.73	2.41	28.2	6.18	72.0
	40	2.08	3.10	32.9	15.27	86.3
	45	2.28	3.52	35.2	34.56	93.4
	50	2.53	3.99	36.5	-	-
	60	3.13	5.33	41.2	-	-
	70	3.98	7.44	46.5	-	-
	80	5.53	11.75	52.9	-	-
	85	6.70	18.15	63.1	-	-
Newsgroup	10	1.19	1.29	7.75	1.62	26.5
	20	1.41	1.57	10.1	2.74	48.5
	30	1.65	1.97	16.2	5.16	68.0
	40	1.91	2.44	21.7	13.90	86.2
	44	2.01	2.68	25.0	25.66	92.1
	50	2.21	3.02	26.8	-	-
	60	2.61	3.84	32.0	-	-
	70	3.27	5.24	37.5	-	-
	80	4.39	7.75	43.3	-	-
	87	6.09	14.05	56.6	-	-

- The *popular word method* cannot reach beyond 50% coverage rate and our method gives around 90% savings at its highest coverages on the four corpora.

As we have explained before, in the presence of a return limit and with our static ranking criterion, choosing many high frequency queries will result in a low coverage rate and a high overlapping rate, because (i) some documents with high rank value may not be retrieved at all and as a consequence, the desired coverage rate cannot be reached; (ii) some documents with low rank value may be repeatedly retrieved, causing the retrieval of many redundant documents.

On the other hand, choosing many low frequency queries will result in a high coverage rate and a low overlapping rate, because (i) when we use queries with small document frequencies, any document can be returned by sending some of the queries; (ii) the coverage of the returned documents is more even which means less an overlapping rate.

Thus, choosing many low frequency terms as we do in the present work is much better than choosing many high frequency terms as proposed in the *popular word method* both in terms of coverage rate and in terms of overlapping rate.

With the *random method*, we have a mix of high and low frequency terms selected. Thus, its result both in terms of coverage rate and in terms of overlapping rate is between the *popular word method* and our method.

5.5 Conclusion

In this chapter, we presented a novel method to crawl ranked data source, which only selects the queries whose estimated dfs are less than the return limit using a df estimator. Compared to the traditional methods, our method works well in the presence of a return limit and a static ranking criterion. Given a sample containing enough small queries to cover DB, it can reach a high coverage (over 85%). With the same coverage, our method separately outperforms the random and popular word methods 58% and 90% at most.

We are interested in extending this work for other ranking criteria in the original data source. We have considered original data sources with fixed sizes and a fixed number as the *return limit*. For future work, we are interested in knowing the performance of this present method with the change of the size of the original data

source and the change of the return limit k. Note that in particular, when k is infinite, it corresponds to the case that there is no return limit, which is the setting used in many previous work. On the other hand, when k is pretty small and the size of ranked data source is large, it may be hard to find enough queries whose document frequencies are smaller than k in order to reach a high *hit rate*. In this chapter, we have assumed that by using all queries with document frequencies less than k we can reach a high coverage, and we could select among these queries to reach a given coverage rate. It remains an interesting problem to find the limit of the number of queries we need in order to reach a given coverage rate. In the case we have to work with a set of queries with document frequencies below this limit, we will try to use multiply keywords.

Chapter 6

Conclusions

Deep web crawling is the process of collecting data from search interfaces by issuing queries. It consists of several subtasks, including 1) estimating and locating the deep web data sources [11, 4]; 2) understanding the HTML forms of each data source [15, 73]; 3) selecting appropriate queries [17, 74, 16]; 4) retrieving the returns and extracting the relevant content if it is embedded inside HTML pages. With the adoption of the Web services and the improvement of the information extraction techniques [34, 75, 36, 6], the other subtasks can be addressed better. Query selection problem becomes a main challenge of the deep web crawling.

There are several challenges in the query selection problem including 1) *cold start problem*: there are no queries to select from at the beginning; 2) *optimization problem*: finding out a better optimization algorithm for query selection; 3) *return limit problem*: it is hard to retrieve data from ranked deep web data sources.

Targeting those challenges, we present a novel technique to address the query selection problem from three aspects.

1. *Query selection with sampling:* for most of the query selection algorithms, one of the challenges is the selection of the input. At the beginning of the crawling, there are no queries to select from. Namely, neither documents nor terms are available. One method [17, 7] to solve this problem is the incremental method that selects queries from the documents that have been downloaded. The number of documents increases as more queries are sent. However, this method needs the downloading and analyzing of all retrieved documents in order to generate the next query to be issued. This is highly

inefficient.

Here we argue that a fixed small sample is sufficient to generate high quality queries. The sample is derived from the original data source. Thus, we proposed a sampling-based query selection method. It first creates a sample and chooses a set of terms fully covering the sample. Then it uses the greedy algorithm to generate queries based on the sample and the chosen terms.

We conducted experiments on four different standard corpus data and the results showed that, with small sample and query pool ($D = 3000$ and $\mu = 20$), i) the queries select from our method can cover most of the original data source; ii) the overlapping rate in original data source is better than that of Ntoulas' method [17] and outperforms that of the random method.

2. *Query selection utilizing log-normal data distribution:* The performance of a query selection algorithm for the deep web crawling is normally evaluated by the cost over the coverage. The optimization of the query selection is usually modeled as Set Covering Problem. Most previous query selection algorithms minimize the total number of submitted queries [16, 17]. In this dissertation, we show that the network traffic is the major cost of a query selection algorithm and it can be represented by the total number of retrieved URLs. Under this definition, the goal of the query selection is to minimize the sum of the cardinalities of the queries. More importantly, we find that the distribution of the document degrees could have a great impact on the performance of the conventional greedy algorithm.

   Although there are many elaborate set covering algorithms [49, 45, 76, 46], the greedy algorithm is still a better choice because it is scalable. Conventional greedy algorithm assigns each document with the same weight, which may be good in other domains but not in our application. Unlike the Beasley data, one of standard benchmarks for SCP, in the deep web data sources, the document degree is distributed log-normally, not normally, and it makes the possibility of each document to cause overlap varies from each other. Documents with larger degrees can be covered by more queries and they should be covered later to avoid repeated coverage by the following queries. Thus, we assign the reciprocal of the document degree as the weight of a document and introduce our weighted greedy algorithm. It iteratively selects the query that maximizes

the average of the document weights of all uncovered documents covered by it.

The experiments were carried out on our corpus data and the Beasley data. The results show that i) on average, the weighted greedy algorithm outperforms the greedy algorithm on our corpus data around 15% and on the Beasley data around 5%; ii) the average improvements of the weighted greedy algorithm has a positive linear correlation with the dispersion of document degrees.

3. *Query selection for ranked data sources:* there are many ranked data sources in the deep web, which ranks the matched documents and only returns the top k documents when the number of matched documents is larger than k. In this setting, the queries selected from the algorithms [16, 17] preferring popular terms cannot reach a good result: documents ranked low may not be retrieved and thus it may not be possible to reach a desired high coverage while many documents ranked high will be repeatedly retrieved causing the retrieval of many redundant documents.

   To address this problem, one solution is to select queries with appropriate document frequencies F such that $F \leq k$. However, it is hard to have such information from the websites holding ranked data sources. Thus, our df-based method obtains a sample first and then estimate the document frequency for each term inside it to remain the qualified ones, finally the remaining terms are randomly selected to be sent to the original data sources until the desired coverage is reached. We tested various estimation method including the MLE [25], the SGT [28] and the Zipf's-law-based estimation [29] and, found that SGT outperformed the others.

   To show the performance of our method, we compared it with the random method and the popular word method. The former randomly selects queries from a middle-sized Webster English dictionary and the latter only chooses the terms whose frequencies are more than $2k$ as queries. The experimental results show that i) both of our and the random methods can reach a high coverage (HR ¿ 84%) and our method is better on average 59% on OR; ii) the popular word method has the worst performance and it cannot even reach more than 50% coverage with too much redundant retrieval, and our method outperforms it around 90% on OR. These results conform to the theoretical

guide shown in equation 5.1 especially for the second one. It says that the range of returned documents can be approximately decided by the document frequencies of queries and the return limit.

Appendix A

Details of the comparison between greedy and weighted greedy algorithms

This appendix contains the complete tables and supplementary figures for the experiments in Chapter 3 and Chapter 5. Table A.1 and Table A.2 are the complete results of the experiments shown in Table 3.8 and Table 3.9. Figure A.1 and Figure A.2 are the supplementary illustrations for the experimental results of the greedy and weighted greedy methods on our corpus data and Beasley data respectively, which have been discussed in Section 3.5.3. Figure A.3 is shown here as the supplement of Figure 3.12 for the relationship between the CV of document degree and the average of the improvement of the weighted greedy method on our and Beasley data.

Table A.1: Greedy vs weighted greedy. The results with redundancy based on 100 times running experiments on the Beasley data.

Data set	Greedy Method				Weighted Greedy				Improvement(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVG
4.1	236	220	227	3.1	217	214	215	1.5	8.05	2.72	5.28
4.2	236	218	227	3.1	222	216	219	1.9	5.93	0.92	3.52
4.3	236	219	226	3.2	224	216	219	2.4	5.08	1.36	3.09
4.4	235	220	227	3.3	219	216	217	1.3	6.81	1.86	4.41

4.5	235	219	227	3.2	216	214	215	1	8.08	2.28	5.28
4.6	233	218	226	3.1	217	216	216	0.5	6.86	0.91	4.42
4.7	236	217	227	3.7	220	216	217	1.2	6.77	0.47	4.41
4.8	236	219	228	3.4	220	216	217	1.2	6.77	1.36	4.82
4.9	233	217	226	3.3	219	217	218	0.6	6.01	0	3.54
4.10	234	218	226	3.2	211	211	211	0	9.82	3.21	6.64
5.1	222	209	214	2.5	211	206	208	1.0	4.95	1.43	2.80
5.2	220	211	215	2.0	211	206	208	1.3	4.09	2.36	3.26
5.3	222	210	215	2.4	211	208	209	0.7	4.95	0.95	2.79
5.4	221	208	215	2.5	210	209	209	0.4	4.97	-0.48	2.79
5.5	217	206	213	2.2	209	205	206	1.1	3.68	0.49	3.29
5.6	220	208	214	2.3	206	206	206	0	6.36	0.96	3.74
5.7	221	206	214	2.3	207	203	205	0.9	6.33	1.46	4.21
5.8	219	210	214	2.1	207	203	205	1.0	5.48	3.33	4.21
5.9	225	210	216	2.5	212	208	210	0.8	5.78	0.95	2.77
5.10	222	210	216	2.7	208	206	207	0.6	6.31	1.90	4.16
6.1	285	262	271	5.2	262	262	262	0	8.07	0	3.32
6.2	288	258	270	5.4	263	263	263	0	8.68	-1.94	2.59
6.3	282	261	271	4.7	264	264	264	0	6.38	-1.15	2.58
6.4	283	256	270	5.3	261	261	261	0	7.77	-1.95	3.33
6.5	284	258	271	5.8	270	270	270	0	4.93	-4.65	0.37
a.1	351	335	342	3.6	335	335	335	0	4.56	0	2.05
a.2	351	335	342	3.1	332	332	332	0	5.41	0.90	2.92
a.3	354	335	343	3.7	337	337	337	0	4.80	-0.60	1.75
a.4	351	332	341	3.8	342	334	337	2.2	2.56	-0.60	1.17
a.5	315	334	342	3.9	332	332	332	0	-5.40	0.60	2.92
b.1	438	410	425	5.7	414	414	414	0	5.48	-0.98	2.59
b.2	441	413	425	6.2	428	428	428	0	2.95	-3.63	-0.71
b.3	443	408	426	6.7	418	418	418	0	5.64	-2.45	1.88
b.4	440	409	423	6.3	422	422	422	0	4.09	-3.18	0.24
b.5	439	412	424	5.4	418	417	417	0.5	4.78	-1.21	1.65
c.1	495	469	485	5	470	470	470	0	5.05	-0.21	3.09
c.2	496	475	486	4.6	491	490	490	0.5	1.01	-3.16	-0.82
c.3	498	472	484	5.3	466	466	466	0	6.43	1.27	3.72

c.4	496	466	484	5.7	476	476	476	0	4.03	-2.15	1.65
c.5	497	476	484	4.3	489	489	489	0	1.61	-2.73	-1.03
d.1	630	593	612	7.8	604	604	604	0	4.13	-1.85	1.31
d.2	631	593	611	8.1	588	588	588	0	6.81	0.84	3.76
d.3	630	590	612	7.4	616	616	616	0	2.22	-4.41	-0.65
d.4	636	587	611	8.2	608	608	608	0	4.40	-3.58	0.49
d.5	629	587	611	8.4	612	612	612	0	2.70	-4.26	-0.16
e.1	70	60	65	2.2	62	62	62	0	11.43	-3.33	4.62
e.2	69	60	64	2	60	60	60	0	13.04	0	6.25
e.3	69	59	64	2.1	64	64	64	0	7.25	-8.47	0
e.4	69	60	64	2.2	65	65	65	0	5.80	-8.33	-1.56
e.5	71	60	65	2.3	67	67	67	0	5.63	-11.67	-3.08

Table A.2: Greedy vs weighted greedy. The results without redundancy based on 100 times running experiments on Beasley data.

Data set	Greedy Method				Weighted Greedy				Improvement(%)		
	MAX	MIN	AVE	SD	MAX	MIN	AVE	SD	MAX	MIN	AVG
4.1	233	217	224	3.3	217	213	214	1.4	6.86	1.84	4.46
4.2	232	216	224	3.3	220	211	216	2	5.17	2.31	3.57
4.3	235	215	223	3.6	222	212	216	2.6	5.53	1.40	3.13
4.4	233	217	224	3.3	214	209	211	1.6	8.15	3.68	5.80
4.5	234	217	224	3.4	214	209	211	1.2	8.55	3.68	5.80
4.6	231	215	224	3.1	216	214	215	0.7	6.49	0.46	4.02
4.7	234	215	224	3.6	220	211	214	2.0	5.98	1.86	4.89
4.8	234	215	225	3.6	220	210	215	2.3	5.98	2.32	4.44
4.9	235	215	223	3.6	219	213	216	1.2	6.81	0.93	3.14
4.10	230	216	223	3.2	210	208	209	0.7	8.69	3.70	6.28
5.1	221	207	213	2.5	209	204	206	1.2	5.42	1.45	3.28
5.2	217	208	213	2.2	209	202	206	1.4	3.68	2.88	3.28
5.3	219	207	213	2.5	209	202	206	1.6	4.56	2.42	3.28
5.4	219	207	213	2.7	210	206	208	1.1	4.11	0.48	2.35
5.5	216	206	211	2.2	207	202	204	1.1	4.17	1.94	3.32

5.6	218	206	212	2.2	205	204	204	0.4	5.96	0.97	3.77
5.7	219	208	213	2.1	206	202	203	1	5.93	2.88	4.69
5.8	222	208	214	2.4	210	203	206	1.4	5.41	2.40	3.73
5.9	220	209	214	2.4	207	203	205	1.0	5.91	2.87	4.21
5.10	219	206	212	2.4	210	203	206	1.5	4.11	1.46	2.83
6.1	285	257	271	5.3	262	262	262	0	8.07	-1.95	3.32
6.2	283	258	270	5.2	263	263	263	0	7.07	-1.94	2.59
6.3	281	258	271	4.7	264	264	264	0	6.05	-2.33	2.58
6.4	283	256	269	5.3	261	261	261	0	7.77	-1.95	2.97
6.5	284	258	271	5.7	267	267	267	0	5.99	-3.49	1.48
a.1	349	332	340	3.8	333	332	332	0.5	4.58	0	2.35
a.2	350	332	339	3.5	330	329	329	0.5	5.71	0.90	2.95
a.3	352	332	340	4	333	333	333	0	5.40	-0.30	2.06
a.4	348	328	339	4.1	337	328	332	2.9	3.16	0	2.06
a.5	349	330	339	3.9	331	325	328	1.6	5.16	1.52	3.24
b.1	438	410	425	5.7	414	414	414	0	5.48	-0.98	2.59
b.2	441	410	425	6.3	428	428	428	0	2.95	-4.39	-0.71
b.3	443	408	426	6.7	411	411	411	0	7.22	-0.74	3.52
b.4	409	440	423	6.4	422	422	422	0	-3.18	4.09	0.24
b.5	439	412	424	5.5	418	417	417	0.5	4.78	-1.21	1.65
c.1	494	468	483	4.8	465	461	463	1.3	5.87	1.50	4.14
c.2	494	474	484	4.6	490	486	487	0.9	0.81	-2.53	-0.62
c.3	496	471	482	5.2	463	463	463	0	6.65	1.70	3.94
c.4	493	465	483	5.7	473	473	473	0	4.06	-1.72	2.07
c.5	472	495	482	4.3	486	486	486	0	-2.97	1.82	-0.83
d.1	630	593	612	7.7	604	604	604	0	4.13	-1.85	1.31
d.2	631	593	610	8.1	588	588	588	0	6.81	0.84	3.61
d.3	630	590	612	7.5	616	616	616	0	2.22	-4.41	-0.65
d.4	636	587	611	8.4	608	608	608	0	4.40	-3.58	0.49
d.5	629	587	611	8.4	612	612	612	0	2.70	-4.26	-0.16
e.1	70	60	65	2.2	62	62	62	0	11.43	-3.33	4.62
e.2	69	60	64	2	60	60	60	0	13.04	0	6.25
e.3	69	59	64	2.1	64	64	64	0	7.25	-8.47	0
e.4	69	60	64	2.2	65	65	65	0	5.80	-8.33	-1.56

e.5	71	60	65	2.3	67	67	67	0	5.63	-11.67	-3.08

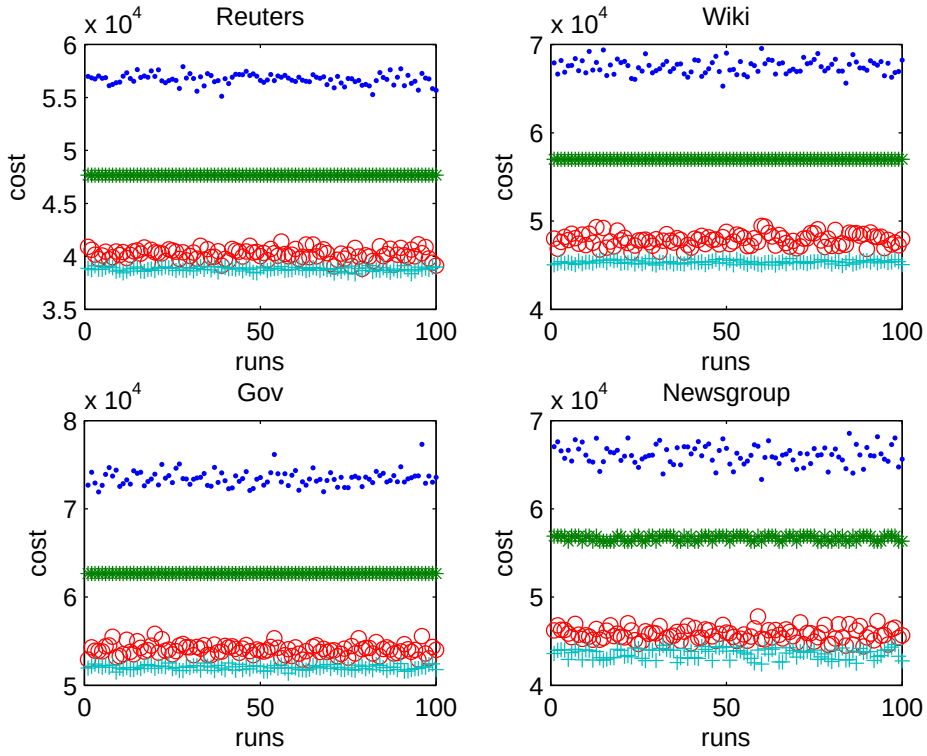

Figure A.1: The results of all the experiments on each corpus data based on 100 runs. '.': greedy with redundancy, '*': weighted greedy with redundancy, 'o': greedy without redundancy, '+': weighted greedy without redundancy.

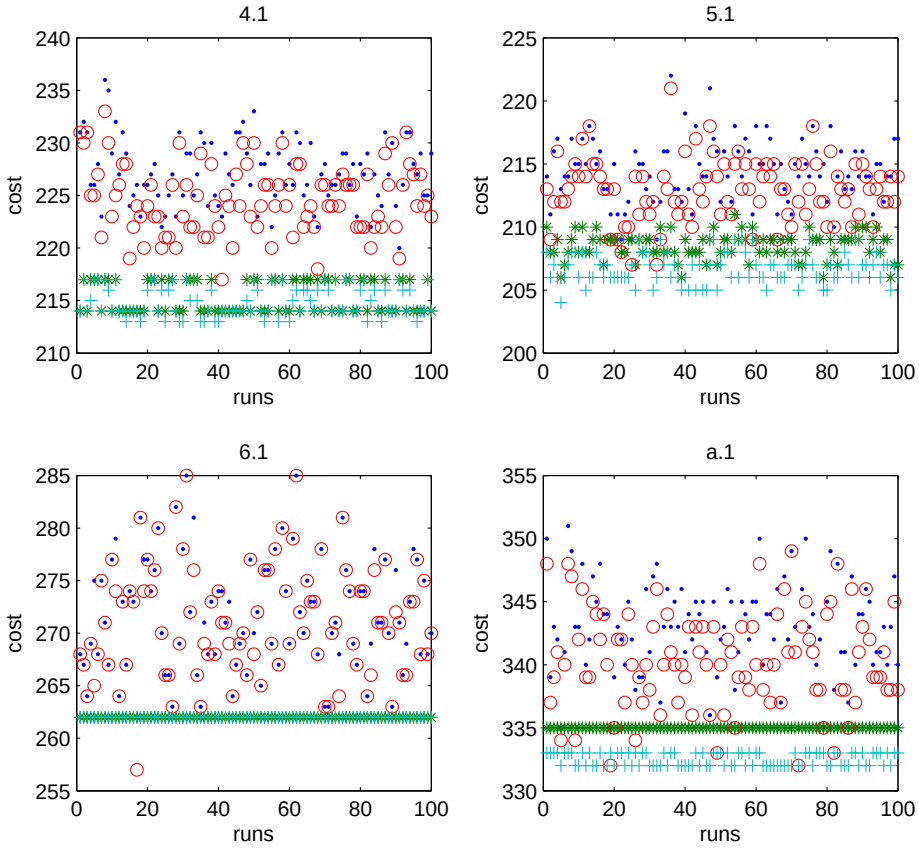

Figure A.2: Part of the results of all the experiments on the Beasley data based on 100 runs. '.': greedy with redundancy, '*': weighted greedy with redundancy, 'o': greedy without redundancy, '+': weighted greedy without redundancy.

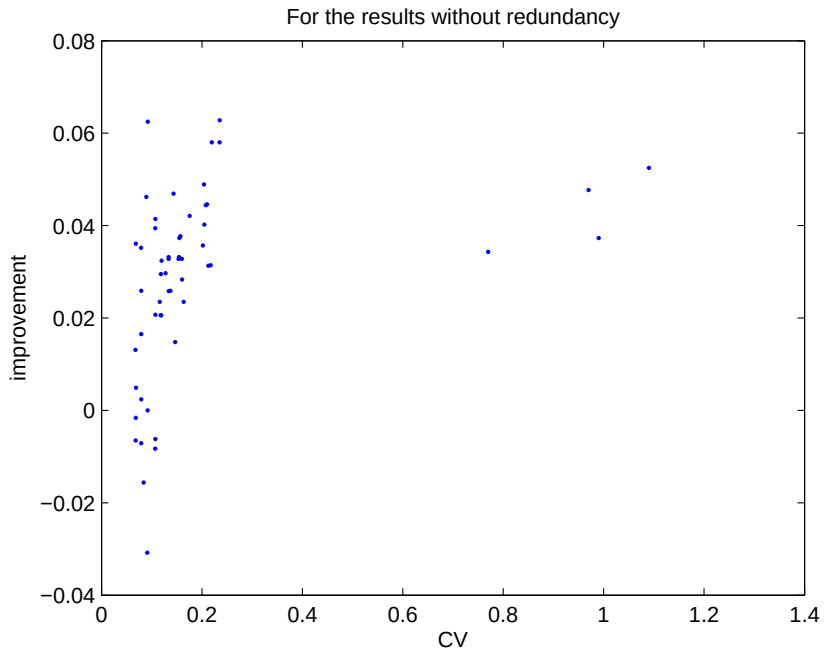

Figure A.3: The relationship between the CV of the document (element) degree and the average of the improvement of the weighted greedy method on our and the Beasley data without redundancy.

Appendix B

Detail of the example for Simple Good-Turing estimation

Table B.1 shows the complete intermediate calculation results of Example 4.

Table B.1: The full set of (f, n_f, Z_f, log10(f), log10(Z_f), f^*) tuples for the example of Section 2.4.2

f	n_f	Z_f	log10(f)	log10(Z_f)	f^*	df	n_f	Z_f	log10(f)	log10(Z_f)	f^*
1	1367	1367	0	3.13	0.45	63	2	2.00	1.80	0.30	62.30
2	314	314	0.30	2.49	1.62	64	1	1.00	1.81	0.00	63.30
3	170	170	0.47	2.23	2.44	65	1	1.00	1.81	0.00	64.30
4	83	83	0.60	1.91	3.41	66	1	1.00	1.82	0.00	65.30
5	79	79	0.69	1.89	4.39	67	3	2.00	1.83	0.30	66.30
6	63	63	0.77	1.79	5.38	69	2	1.00	1.84	0.00	68.30
7	35	35.00	0.85	1.54	6.37	71	1	0.67	1.85	-0.18	70.30
8	31	31.00	0.90	1.49	7.36	72	1	1.00	1.86	0.00	71.30
9	26	26.00	0.95	1.41	8.35	73	1	1.00	1.86	0.00	72.30
10	26	26.00	1.00	1.41	9.35	74	1	1.00	1.87	0.00	73.30
11	25	25.00	1.04	1.40	10.34	75	1	1.00	1.88	0.00	74.30
12	17	17.00	1.08	1.23	11.34	76	1	1.00	1.88	0.00	75.30
13	15	15.00	1.11	1.18	12.34	77	3	2.00	1.89	0.30	76.30
14	18	18.00	1.15	1.26	13.33	79	1	0.67	1.90	-0.18	78.30
15	3	3.00	1.18	0.48	14.33	80	1	0.67	1.90	-0.18	79.30

16	19	19.00	1.20	1.28	15.33	82	1	0.67	1.91	-0.18	81.30
17	16	16.00	1.23	1.20	16.33	83	1	1.00	1.92	0.00	82.30
18	12	12.00	1.26	1.08	17.32	84	1	0.67	1.92	-0.18	83.65
19	8	8.00	1.28	0.90	18.32	86	1	0.67	1.93	-0.18	85.30
20	4	4.00	1.30	0.60	19.32	87	3	2.00	1.94	0.30	86.30
21	10	10.00	1.32	1.00	20.32	89	1	0.40	1.95	-0.40	88.30
22	11	11.00	1.34	1.04	21.32	92	1	0.33	1.96	-0.48	91.30
23	5	5.00	1.36	0.70	22.32	95	1	0.33	1.98	-0.48	94.30
24	5	5.00	1.38	0.70	23.32	98	1	0.50	1.99	-0.30	97.30
25	10	10.00	1.40	1.00	24.32	99	1	0.40	2.00	-0.40	98.30
26	1	1.00	1.41	0.00	25.32	103	2	0.80	2.01	-0.10	102.30
27	7	7.00	1.43	0.85	26.31	104	1	1.00	2.02	0.00	103.30
28	5	5.00	1.45	0.70	27.31	105	1	1.00	2.02	0.00	104.30
29	8	8.00	1.46	0.90	28.31	106	1	1.00	2.03	0.00	105.30
30	3	3.00	1.48	0.48	29.31	107	1	1.00	2.03	0.00	106.30
31	6	6.00	1.49	0.78	30.31	108	1	1.00	2.03	0.00	107.30
32	1	1.00	1.51	0.00	31.31	109	1	0.40	2.04	-0.40	108.30
33	2	2.00	1.52	0.30	32.31	113	1	0.40	2.05	-0.40	112.30
34	5	5.00	1.53	0.70	33.31	114	1	1.00	2.06	0.00	113.30
35	2	2.00	1.54	0.30	34.31	115	1	0.67	2.06	-0.18	114.30
36	3	3.00	1.56	0.48	35.31	117	1	0.67	2.07	-0.18	116.30
37	6	6.00	1.57	0.78	36.31	118	1	0.33	2.07	-0.48	117.30
38	5	5.00	1.58	0.70	37.31	123	2	0.67	2.09	-0.18	122.30
39	2	2.00	1.59	0.30	38.31	124	1	0.50	2.09	-0.30	123.30
40	5	5.00	1.60	0.70	39.31	127	1	0.33	2.10	-0.48	126.30
41	2	1.33	1.61	0.12	40.31	130	1	0.40	2.11	-0.40	129.30
43	3	2.00	1.63	0.30	42.31	132	1	0.40	2.12	-0.40	131.30
44	3	3.00	1.64	0.48	43.31	135	1	0.20	2.13	-0.70	134.30
45	2	2.00	1.65	0.30	44.31	142	1	0.20	2.15	-0.70	141.30
46	1	1.00	1.66	0.00	45.31	145	1	0.50	2.16	-0.30	144.30
47	1	1.00	1.67	0.00	46.31	146	1	1.00	2.16	0.00	145.30
48	2	2.00	1.68	0.30	47.31	147	1	0.22	2.17	-0.65	146.30
49	1	0.50	1.69	-0.30	48.31	155	1	0.17	2.19	-0.78	154.30
52	2	1.00	1.72	0.00	51.30	159	1	0.11	2.20	-0.95	158.30

53	3	3.00	1.72	0.48	52.30	173	1	0.05	2.24	-1.29	172.30
54	3	3.00	1.73	0.48	53.30	198	1	0.04	2.30	-1.40	197.30
55	2	2.00	1.74	0.30	54.30	223	1	0.06	2.35	-1.19	222.30
56	1	1.00	1.75	0.00	55.30	229	1	0.20	2.36	-0.70	228.30
57	1	1.00	1.76	0.00	56.30	233	1	0.25	2.37	-0.60	232.30
58	1	1.00	1.76	0.00	57.30	237	1	0.07	2.37	-1.13	236.30
59	2	1.33	1.77	0.12	58.30	260	1	0.01	2.41	-1.94	259.30
61	3	2.00	1.79	0.30	60.30	412	1	0.006	2.61	-2.18	411.29
62	2	2.00	1.79	0.30	61.30						

Appendix C

Maximum likelihood estimation

In MLE applied to df estimation, firstly of all, we need to make an assumption and it is that any document contains any term in D independently. With the assumption, the sample df of a query q follows the *Binomial distribution* and its *probability mass function* (PMF) is defined as follows:

$$P(f(q)|N,p) = \binom{N}{f(q)} p^{f(q)}(1-p)^{N-f(q)} \tag{C.1}$$

where $f(q)$ is the sample df of the term q and $N = \sum_{q \in D} f(q)$ is the sum of the dfs of all terms in D, and p is the probability of q defined in equation 5.1.

Example 6 *To illustrate the concept of the PMF, here a simple artificial example is shown in Figure C.1. Suppose that a sample D is randomly collected from an original data source DB and it contains 100 documents ($m = 100$) and 500 terms ($n = 500$), and its total dfs are 1500(N=1500). For a term q in D, its probability in DB is 0.4% ($p = 0.004$) and then the PMF of its df in D is shown as following:*

$$P(f(q)|N = 1500, p = 0.004) = \binom{1500}{f(q)} (0.004)^{f(q)}(1-0.004)^{1500-f(q)}.$$

The shape of this PMF is shown in Figure C.1(a). This figure shows that, for q, it is most possible that its df is 5 or 6. If the parameter value is changed to say $p = 0.008$, a new PMF is obtained as

$$P(f(q)|N = 1500, p = 0.008) = \binom{1500}{f(q)} (0.008)^{f(q)}(1-0.008)^{1500-f(q)}.$$

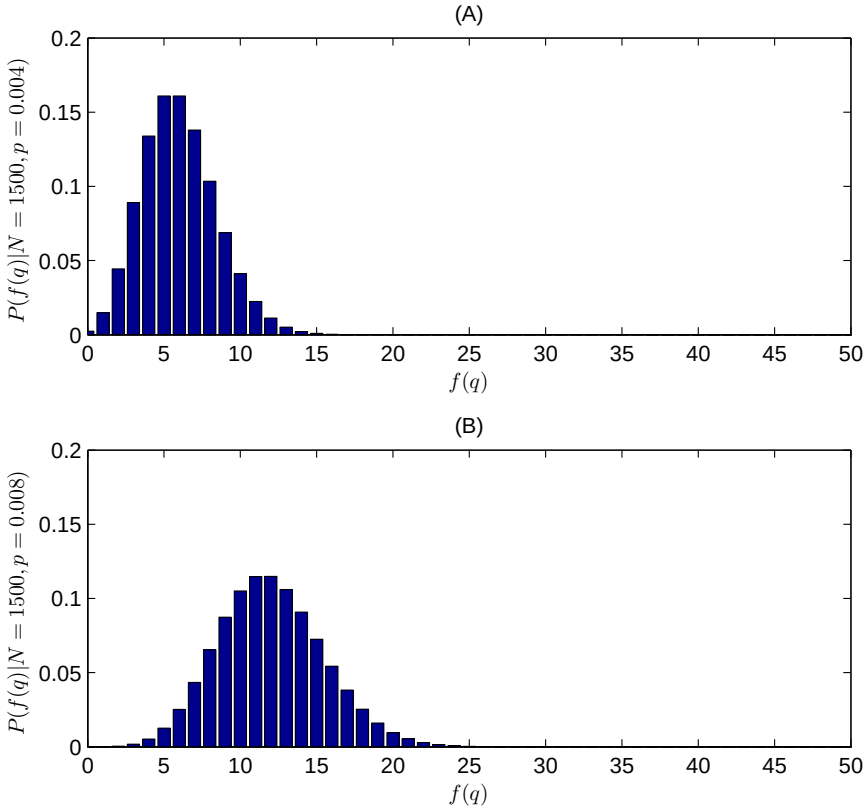

Figure C.1: The probability mass functions of the document frequency of query q with $N = 1500$ and $p = 0.004$(A) and $p = 0.008$(B)

Its shape is shown in Figure C.1(b).

Given a set of parameter values, the corresponding PMF will show that some data are more probable than other data. In the above example, the PMF with $p = 0.004$ and $N = 1500$, $f(q) = 5$ is more likely to occur than $f(q) = 2$(0.1608 vs 0.0444). In reality, however, the sample D is given first. Accordingly, we are faced with an inverse problem: given the sample D, with $f(q)$ of any term q and N, our purpose is to find its probability p in DB. To address the problem, the *likelihood function* is introduced by reversing the roles of $f(q)$ and p, and it is shown in equation C.2

$$L(p|N = c, f(q) = d) = P(f(q) = d|N = c, p) = \binom{N}{d} p^d (1-p)^{N-d}. \qquad \text{(C.2)}$$

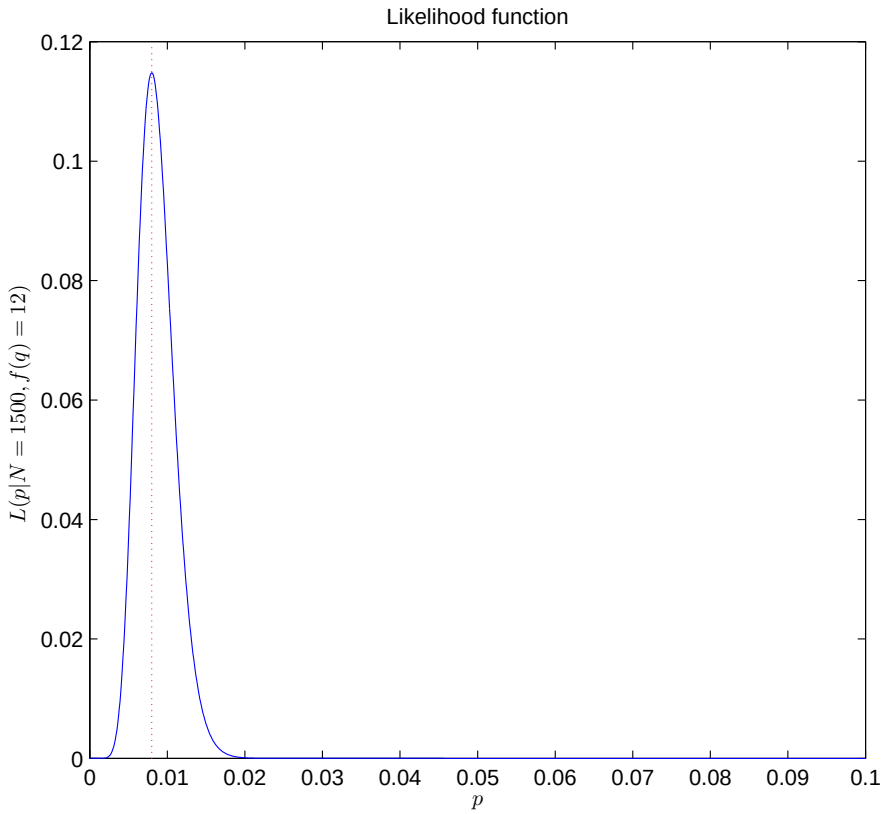

Figure C.2: The likelihood function $L(p|N, f(q))$ with $N = 1500$ and $f(q) = 12$ is based on the binomial distribution model described in the text.

Thus $L(p|N = c, f(q) = d)$ represents the likelihood of the parameter p of a term q. For example, the likelihood function for $f(q) = 12$ and $N = 1500$ is shown as following:

$$L(p|N = 1500, f(q) = 12) = \binom{N}{d} p^d (1-p)^{N-d} = \frac{1500!}{10!1488!} p^{12} (1-p)^{1488}.$$

and it is shown in Figure C.2.

Given the likelihood function and the sample D, we are in position to use the MLE to estimate the probability p of a query q in D. The MLE states that the desired probability distribution is the one that makes the observed data "most likely" to be produced. In our case, for a term q, the value for its p that makes the corresponding $L(p|N, f(q))$ maximum should be selected. For example,

in Figure C.2, $p = 0.008$ is selected by the MLE, which is the maximum value $L(p = p|N = 1500, f = 12) = 0.1148$. Simultaneously, the corresponding PMF is shown in C.1(B). According to the principle, setting p to 0.008 is most likely to have produced the term with $f(q) = 12$ while $N = 1500$. In a word, the object of the MLE is to seek the probability distribution that can produce the observed data most likely.

Finally, since the likelihood function $L(p|N, f(q))$ for binomial distribution is a convex function. Thus, the MLE estimates exist and are unique for $L(p|N, f(q))$ and it must satisfy the following partially differentiating equation known as *likelihood equation*:

$$\frac{\partial lnL(p|N, f(q))}{\partial p} = 0. \tag{C.3}$$

where $lnL(p|N, f(q))$ is log-likelihood function and both of the functions ($lnL(p|N, f(q))$ and $L(p|N, f(q))$) are monotonically related to each other. So, the same MLE estimate is obtained by maximizing either one. After solving equation C.3, finally we have the MLE estimate shown as follows:

$$\widehat{p} = \frac{f(q)}{N}. \tag{C.4}$$

According to equation C.4, we have the estimation function $\widehat{F}(q) = \widehat{p} \times \sum_{q' \in DB} F(q') = f(q) \times \frac{|DB|}{|D|}$, the estimated document frequency of any term can easily be calculated. The interesting readers can refer to [27] for more details of MLE.

Bibliography

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

[2] M.K.Bergman, "The deepweb: Surfacing hidden value," *The Journal of Electronic Publishing*, vol. 7, no. 1, 2001.

[3] "www.programmableweb.com."

[4] B.He, M.Patel, Z.Zhang, and K.C.Chang, "Accessing the deep web: A survey," *Communications of the ACM*, vol. 50, no. 5, pp. 94–101, May 2007.

[5] J. Madhavan, S. Cohen, X. Dong, A. Halevy, S. Jeffery, D. Ko, and C. Yu, "Web-scale data integration: You can afford to pay as you go," in *Proc. of CIDR*, 2007, pp. 342–350.

[6] C.H.Chang, M.Kayed, M.R.Girgis, and K.F.Shaalan, "A survey of web information extraction systems," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 10, pp. 1411–1428, Oct. 2006.

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

[8] J. Rennie and A. McCallum, "Using reinforcement learning to spider the web," in *Proc. of ICML*, 1999, pp. 335–343.

[9] S. Sizov, M. Biwer, J. Graupmann, S. Siersdorfer, M. Theobald, G. Weikum, and P. Zimmer, "The bingo! system for information portal generation and expert web search," in *Proc. of CIDR*, 2003.

[10] S.Chakrabarti, M.V.D.Berg, and B.Dom, "Focused crawling: a new approach to topic-specific web resource discovery," in *Proc. of WWW*, 1999.

[11] L.Barbosa and J.Freire, "An adaptive crawler for locating hidden-web entry points," in *Proc. of WWW*, 2007, pp. 441–450.

[12] M.Alvarez, A.Pan, J.Raposo, F.Bellas, and F.Cacheda, "Extracting lists of data records from semi-structured web pages," *Data Knowl Eng*, vol. 64, no. 2, pp. 491–509, 2008.

[13] C.A.Knoblock, K.Lerman, S.Minton, and I.Muslea, "Accurately and reliably extracting data from the web: a machine learning approach," *IEEE Data Engineering Bulletin*, vol. 23, no. 4, pp. 33–41, 2000.

[14] J.Lu and D.Li, "Estimating deep web data source size by capture-recapture method," *Informatoin Retrieval*, vol. 13, no. 1, pp. 70–95, 2010.

[15] S.Raghavan and H.G.Molina, "Crawling the hidden web," in *Proc. of the 27th international Conference on Very Large Data Bases (VLDB)*, 2001, pp. 129–138.

[16] L.Barbosa and J.Freire, "Siphoning hidden-web data through keyword-based interfaces," in *Proc. of SBBD*, 2004.

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

[18] J.Caverlee, L.Liu, and 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, pp. 103–114.

[19] S.W.Liddle, D.W.Embley, D.T.Scott, and S.H.Yau, "Extracting data behind web forms," in *Proc. of Advanced Conceptual Modeling Techniques*, 2002.

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

[21] J.E.Beasley and P.C.Chu, "Theory and methodology. a genetic algorithm for the set covering problem," *European Journal of Operational Research*, vol. 94, no. 392-404, 1996.

[22] T.A.Feo and M.G.Resende, "Greedy randomized adaptive search procedures," *Journal of Global Optimization*, pp. 109–133, 1995.

[23] L.W.Lorena and F.B.Lopes, "A surrogate heuristic for set covering problems," *European Journal of Operational Research*, vol. 1994, no. 79, pp. 138–150, 1994.

[24] A. Caprara, M. Fishetti, and P. Toth, "A heuristic method for the set covering problem," *Operations Research*, 1995.

[25] H. D. Mittelmann, *Recent Benchmarks of Optimization Software*, 2007.

[26] J. E. Beasley, "Or-library: Distributing test problems by electronic mail," *Journal of the Operational Research Society*, vol. 41, no. 11, pp. 1069–1072, 1990.

[27] I. J. Myung, "Tutorial on maximum likelihood estimation," *Jounral of Mathematical Psychology*, vol. 47, pp. 90–100, 2003.

[28] W.A.Gale and G.Sampson, "Good-turing frequency estimation without tears*," *Journal of Quantitative Linguistics*, 1995.

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

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

[31] A.Doan, P.Domingos, and A.Y.Halevy, "Reconciling schemas of disparate data sources: A machine-learning approach," in *Proc. of SIGMOD*, 2001.

[32] J.Madhavan, P.A.Bernstein, A.Doan, and A.Y.Halevy, "Corpus-based schema matching," in *Proc. of ICDE*, 2005.

[33] B. He, W. Meng, C. Yu, and Z. Wu, "Automatic integration of web search interfaces with wise-integrator," *VLDB Journal*, vol. 13, no. 3, pp. 255–273, 2004.

[34] N.Kushmerick, D.S.Weld, and R.Doorenbos, "Wrapper induction for information extraction," in *Proc. of IJCAI*, 97.

[35] M.L.Nelson, J.A.Smith, and I.G.D.Campo, "Efficient, automatic web resource harvesting," in *Proc. of RECOMB*, 2006, pp. 43–50.

[36] J.P.Lage, A.S.Silva, P.B.Golgher, and A.H.F.Laender, "Automatic generation of agents for collecting hidden web pages for data extraction," *Data Knowl Eng*, pp. 177–196, 2004.

[37] M.Diligenti, F.Coetzee, S.Lawrence, C.L.Giles, and M.Gori, "Focused crawling using context graphs," in *Proc. of VLDB*, 2000.

[38] K.Bharat and M.R.Henzinger, "Improved algorithms for topic distillation in a hyperlinked environment," in *Proc. of SIGIR*, 1998.

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

[40] K.Bharat and A.Border, "A technique for measuring the relative size and overlap of public web search engines," in *WWW*, 1998, pp. 379–388.

[41] "http://www.ncbi.nlm.nih.gov/pubmed/."

[42] "http://www.uspto.gov/."

[43] T.H.Cormen, C.E.Leiserson, R.L.Rivest, and C.Stein., *Introduction to Algorithms*, second edition ed. MIT Press and McGraw-Hill, 2001.

[44] J.E.Beasley, "An algorithm for set covering problems," *European Journal of Operational Research*, vol. 31, 1996.

[45] J.E.Beasley and K.Jornsten, "Enhancing an algorithm for set covering problems," *European Journal of Operational Research*, vol. 58, pp. 293–300, 1992.

[46] F.Balas and M.C.Carrera, "A dynamic subgradient-based branch-and-bound procedure for set covering," *Operations Research*, vol. 44, pp. 875–890, 1996.

[47] C. B. Moler, *Numerical Computing with MATLAB*. the Society for Indutrial and Applied Mathematics, 2004.

[48] A.Caprara, P.Toth, and M.Fishetti, "Algorithms for the set covering problem," *Annals of Operations Research*, vol. 98, pp. 353–371, 2000.

[49] E. Balas and A. Ho, "Set covering algorithms using cuting planes, heuristics, and subgradient optimization: a computtational study," *Math. Orogr. Studies*, vol. 1980, no. 12, pp. 36–60, 1980.

[50] P. Erdos, "On random graphs," *Publ. Math*, vol. 6, pp. 290–297, 1959.

[51] J. Aitchison and J. A. C. Brown, *The Lognormal Distribution*. Cambridge, England: Cambrige University Press., 1957.

[52] C. F. Gauss, "Theoria motvs corporvm coelestivm in sectionibvs conicis solem ambientivm," 1809.

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

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

[55] J.Lu, "Ranking bias in deep web size estimation using capture recapture method," *Journal of Data and Knowledge Engineering*, vol. 69, no. 8, pp. 866–879, 2010.

[56] G. J. Lidstone, "Note on the general case of the bayes-laplace formula for inductive or a posterior probabilities," *Transactions of the Faculty of Actuaries*, vol. 8, pp. 182–192, 1920.

[57] E. J. W, "Probability: the deductive and inductive problems," *Mind n.s*, vol. 41, pp. 409–423, 1932.

[58] H. Jeffreys, *Theory of Probability*, 2nd ed. Oxford:Clarendon Press, 1948.

[59] F. Jelinek and R. Mercer, "Probability distribution estimation from sparse data," *IBM Technical Disclosure Bulletin*, vol. 28, pp. 2591–2594, 1985.

[60] N. Hermann, "On structuring probabilistic dependences in stochastic language modeling," *Computer, Speech, and Language*, vol. 8, pp. 1–38, 1994.

[61] R. Kneser and N. Hermann, "Improved backing-off for m-gram language modeling," in *Proc. of IEEE 8th International Conference on Acoustics, Speech and Signal Processing*, vol. 1, 1995, pp. 181–184.

[62] S. F. Chen and J. Goodman, "An empirical study of smoothing techniques for language modeling," Harvard University, Technical Report TR-10-98, 1998.

[63] A. Nasdas, "Estimation of probabilities in the language model of the ibm speech recognition system," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. 32, no. 4, pp. 859–861, 1984.

[64] S.M.Katz, "Estimation of probabilities from sparse data for the language model component of a speech recognizer," *IEEE Transactions on Acoustics, Speech and Signal Processing*, vol. ASSP-35, no. 3, pp. 400–401, March 1987.

[65] X.Yan and X.G.Su, *Linear Regression Analysis: Theory and Computing*. World Scientific, 2010.

[66] B.B.Mandelbrot, *Fractal Geometry of Nature*. W.H.Freeman Press, 1988.

[67] M. E. J. Newman, "Power laws, pareto distributions and zipf's law," *Contemporary Phyiscs*, vol. 46, no. 323, 2005.

[68] R.Lambiotte, M.Ausloos, and M.Thelwall, "Word statistics in blogs and rss feeds: Towards empirical universal evidence," *Journal of Informetrics*, vol. 1, no. 4, pp. 277–286, 2007.

[69] R. Rousseau, "Lack of standardisation in informetric research. comments on "power laws of research output. evidence for journals of economics" by matthias sutter and martin g. kocher," *Scientometrics*, vol. 55, no. 2, pp. 317–327, 2002.

[70] R.R.Cancho and R.V.Solé, "Two regimes in the frequency of words and the origin of complex lexicons: Zipf's law revisited," *Journal of Quantitative Linguistics*, 2001.

[71] M.A.Montemurro, "Beyond the zipf–mandelbrot law in quantitative linguistics," *Journal of Physica A*, 2001.

[72] A. F. J. van Rann, "Two-step competition process leads to quasi power-law income distributions: Application to scientific publication and citation distributions," *Physica A*, vol. 298, pp. 530–536, 2001.

[73] R.Khare, Y.An, and I.Song, "Understanding deep web search interfaces: A survey," *ACM SIGMOD Record*, vol. 39, no. 1, pp. 33–40, March 2010.

[74] J.Lu, Y.Wang, J.liang, J.Chen, and J.Liu, "An approach to deep web crawling by sampling," in *Proc. of Web Intelligence*, 2008, pp. 718–724.

[75] V. Crescenzi, G. Mecca, and P. Meriakto, "Roadrunner: towards automatic data extraction from large web sites," in *Proc. of VLDB*, 2001.

[76] J.E.Beasley, "A lagrangian heuristic for set covering problems," *Naval Research Logistics*, vol. 37, 1990.

Vita Auctoris

Yan Wang was born in 1980 in Sichuan Province of China. He finished his Bachelor degree at the department of Computer Science in Beijing Information Technology Institute in 2003. He went on to study Computer Science at University of Windsor in Ontario-Canada and received his Master degree in 2007 and his PhD degree in Computer Science as well in Winter 2012.