2016

Practical Guides for Data Retrieval in Deep Web Crawling

Xu Sun
University of Windsor

Practical Guides for Data Retrieval in Deep Web Crawling

By

Xu Sun

A Thesis
Submitted to the Faculty of Graduate Studies
through the School of Computer Science
in Partial Fulfillment of the Requirements for
the Degree of Master of Science
at the University of Windsor

Windsor, Ontario, Canada

2016

Practical Guides for Data Retrieval in Deep Web Crawling

by

Xu Sun

APPROVED BY:

Dr. Huapeng Wu
Electrical and Computer Engineering

Dr. Dan Wu
School of Computer Science

Dr. Jessica Chen, Advisor
School of Computer Science

September 15, 2016

DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyones copyright nor violate any proprietary rights and that any ideas, techniques, quotations, 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 and have included copies of such copyright clearances to my appendix.

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 or Institution.

ABSTRACT

Deep web crawling refers to the process of collecting documents that have been organized into a data source and can only be retrieved via a search interface. This is often achieved by sending different queries to the search interface. Dealing with the difficulty in selecting suitable set of queries, this crawling process can be implemented with stepwise refinement: documents are retrieved step by step, while in each step, we adapt the query selection to our accumulated knowledge obtained from the documents downloaded in the previous steps. However, it takes much of our time and effort to download the documents and learn from the resulting sample in order to improve the query selection. Here we propose a cost-effective, data-driven method for stepping the adaptive crawling of the deep web. Through empirical study, we explore the criteria in setting the lengths of the steps to best balance the trade-off between the sample updating cost and the improved quality of the selected queries. Derived from four existing data sets typically used for deep web crawling, such criteria provide practical guidelines for cost-effective stepwise refinement in iterative document retrieval.

AKNOWLEDGEMENTS

I would like to express my gratitude to my supervisor Dr. Jessica Chen, for her valuable assistance and support during my master program.

I also would like to present my appreciation to my committee members, Dr. Dan Wu and Dr. Huapeng Wu. Thank them for their valuable comments and suggestions to this thesis.

Meanwhile, I would like to thank Dr. Jianguo Lu and Dr. Yan Wang for collecting the data sets and providing suggestions on my work.

Finally, I want to thanks to my parents and my friends who give me consistent help over the past master life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

Introduction

Deep web is usually associated with databases or file systems, and the data kept in the databases or file systems are retrieved using mechanisms different from those used for surface web. With surface web, data can be accessed directly through URLs. With deep web, on the other hand, data are guarded by a search interface. Crawling deep web [6][13][19][26][41] is the process of collecting hidden data by issuing queries through various interfaces such as HTML forms, web services, programmable web API, etc. Crawling deep web is helpful when we want to reuse the excavated data, to provide index to the data, or to build up an integrated environment for search such data.

Two major issues have been substantially addressed in the literature regarding deep web crawling. One is about learning and understanding the interface and the returned result so that query submission and data extraction can be automated [1][23][33]. Another issue, which is considered in this thesis, is about properly selecting queries to be submitted to the search interface so that the hidden data can be retrieved cost-effectively and efficiently.

The cost-effectiveness refers to maximizing the coverage of the retrieved documents in the dataset being considered with minimal cost. Different cost models have been adopted in previous work [20][30][39][42] reflecting different settings in the real world. Typically, they include either sending cost or receiving cost, or both, The sending cost is usually counted using a constant value for each query. It models the situation where document retrieval is charged by the website service provider on a per-request basis. The receiving cost is usually counted using a constant value for

each received link. With the assumption that the received links have more or less the same number of bytes, this provides a simplified model to estimate the total number of bytes downloaded for the request. It models the situation where the data transmission over the internet is charged by the service provider of the data plan on a per-usage basis. In this thesis, both sending and receiving cost are considered. To keep the model simple, the unit sending cost is defined as the ratio to the unit receiving cost.

The efficiency is concerned about reducing the time and memory space consumed for the related computation and document retrieval.

Regarding query selection and data extraction, there are several typical questions: (a) Where do we get the words to choose from? (b) What criteria do we use to select the queries? (c) How to find a proper set of queries satisfying the given criteria? A naive approach is to pick some words randomly from a dictionary. In the past, people have found it more cost-effective to choose the queries from the terms that frequently appear in the documents that are stored in the data source being considered. Core to this proposal is what criteria should be adopted in order to rank the queries for selection. Much of previous work was dedicated to proposing different solutions to such criteria with different cost models [30][42].

This query selection scheme naturally leads to the following two-step document retrieval approach:

- Initially, retrieve a set of sample documents by issuing some words randomly selected from a dictionary.
- A query pool is built by taking terms appeared in the previous set of sample documents. Then, a set of queries is selected and issued to the original database.

Clearly, this two-step approach can also be extended into a multi-step one by repeatedly refining the sample during the process of document retrieval, i.e. to repeatedly select queries, retrieve documents, update sample documents by downloading newly obtained documents to local database, until certain conditions are met, such as the limit of total cost is hit or the retrieval of the documents is sufficient enough.

Normally, this adaptive document retrieval process can be more cost-effective when we take more retrieving steps, because after each sample revision, we are in a better position to select better queries. However, this requires multiple updates of the sample, and it usually takes much of our time and efforts to download and analyze new documents in order to improve the query selection on each step.

Some practical questions arise at this point: What is the optimal number of steps? If each step may have different length, how to determine the optimal length of each step?

In this thesis, we introduce the cost of each round of sample revision into the problem domain, in order to quantitatively measure the trade-off between sending more queries at a time and sending queries more often. This revision cost is defined as the ratio to the unit sending/receiving cost, to keep the cost model simple. Note that: The query here usually refers to the "phrase" or single keyword in our study.

We use *cost limit* to define the length of a step. In a simple situation where there is no receiving cost, this cost limit is reduced to the *number of queries* to select on a step. Based on our cost model, we propose a method to determine, via empirical study, the optimal lengths of the steps in terms of cost limit. The soundness of this method is based on three hypotheses. For each hypothesis, we argue why it should hold, together with some sample supporting data obtained from real datasets.

Through experiments, we have observed that such optimal lengths vary according to different sizes of the sample dataset. Thus, we repeatedly apply the above method to obtain the *sample-condition relationship* i.e. the optimal cost limits according to different sample sizes.

We have applied the above method to four different datasets used for deep web crawling. The resulting sample-condition graphs demonstrate similar values and their distribution.

Such obtained sample-condition relationship provides practical guides on the adaptive document retrieval: We embed previously obtained sample-condition relationship into the future crawling of the other datasets. Using such knowledge, the document retrieval process is then able to dynamically determine, for each round of iteration,

the best length for the next step of iteration. With such knowledge, the document retrieving process will outperform the sample-based and the ordinary incremental approaches proposed in the literature.

The datasets considered for deep web crawling are normally classified into text-based documents [10][38][39] and image-based ones [2][19][28]. In the present work, we only consider the data sources that contain plain text documents. Textual data sources are usually associated with a keyword-based query interface, so the queries are simply words or phrases, also called terms here. Given that many query interfaces nowadays offer the *advanced search* feature to allow users to send multiple-keyword queries, recent work on deep web crawling has seen the consideration of both single-keyword [4][30][42] and multiple-keyword queries [32][41]. For simplicity, only simple-keyword queries are considered in the present thesis. We leave it open for feature work to extend the thesis work into the multiple-keyword setting.

The rest of this thesis is organized as follows: In chapter II, we review the previous research work about deep web crawling. Chapter III introduces the preliminaries of our method. In chapter IV, we present the new cost-effective document retrieval method. In chapter V, we show the experiment results using four datasets. We conclude our studies in chapter VI.

CHAPTER II

Related Work

In this chapter, we summarize the related work of our research with more details. Section 1 reviews several techniques to analyze and select appropriate queries from sample. In section 2, we introduce some other document retrieving strategies in deep web crawling.

1 Query Selection Problem

One important problem of deep web crawling is how to generate a set of queries cost-effectively and efficiently to harvest more documents, i.e determine which queries should be submitted to a search interface. A simple solution is selecting some random words from a dictionary. But it fails to consider that a large number of unnecessary queries with many duplicates would be issued at the same time. Therefore, many approaches [4][12][30][34][38][39] had been proposed to address the query selection problem in deep web crawling.

In [4][37], the author study the problem of siphoning hidden data behind simple search interfaces by issuing keyword-based queries. They suggest that terms presented in a document collection would generate a set of appropriate queries and these queries would lead to a high coverage(over 90% is obtained) for most of their sites considered. Thus, instead of blindly sending queries, they sample HTML forms to select the higher frequency keywords from a potential keyword list which is expected to lead to a high coverage in original datasets. In addition, they consider the performance of stop words in their experiments, which tend to be extremely cost-effective in some of the

document collections even though it sometimes turn out to be useless.

Their algorithm consists of two phases: The first phase is to iterate and build the list of candidate keywords based on their frequency. Once new keywords was appended into this list, the algorithm modifies the frequency rank of existing keywords. In phase 2, they select queries with the highest frequency firstly from the term list in phase 1. They iteratively carry out this process until the coverage in original DB is no more increased.

1.1 Greedy Algorithm

In [30], the author proposed an efficient sampling-based method to retrieve the most of entire database in a text data source with low overlapping rate. At the beginning, they downloaded a set of documents from original DB as the initial sample. From this sample, a set of appropriate queries was acquired. The query selection process in sample could be viewed as a set-covering problem which is NP-hard. So they apply greedy algorithm in experiments to retrieve more documents with a lower cost on selecting each query. This process is iterated until the selected query set can cover all documents in sample. It is estimated that those queries which can return most documents in sample are cost-effective to return most of data in original DB as well; The sample size(the number of documents in sample) do not need to be very large in order to get an accurate enough prediction of the total database. The framework of this approach is showing below:

1. Create a sample D by initially selecting a fixed number of documents from original DB.

2. Analyse the documents and terms in sample D, and build the query pool QP.

3. Select a set of queries Q from QP based on the greedy algorithm.

4. Issue query set Q to original DB to retrieve documents.

In this paper, the framework aims to select the most cost-effective query. The work is carried out within the context that the bottleneck to deep web crawling is the

number of documents crawled, not the queries issued. So their algorithm focus on minimizing the overlapping rate. However, in real deep web crawler, sending queries to original DB may also account for a part of total resources.

1.2 Weighted Greedy Algorithm

In [30], the authors address the query selection problem by cast it as a set-covering problem. Then they suggest greedy algorithm to generate an optimal query set in practice. Therefore, each document is of equal importance in their experiments. However, in real deep web crawling scenario, not every document is the same. That is because a large document is more likely to be returned than the small documents which only contain few words.

The initial solution to set covering problem for deep web crawling [30] was improved [38] with the observation that the document-term relationship in this application domain follows the power law instead of uniform distribution

In [38], Wang et al. further improve the straightforward greedy algorithm by introducing weights into the greedy strategy. Similar to [30], this paper reports the work on generating an appropriate set of queries so that they can cover all the documents in sample D with less overlapping rate. In simple greedy algorithm [30], the unit cost for sending one query q is set to $1/df$, where df is the document frequency of q. The queries are selected one by one. On each step, they select the query with maximum new/df, where new is the number of new documents that could be retuned by q. Unlike [30], in weighted greedy algorithm, the authors incorporate weights into each query selection process, where the weight of a document is the inverse of its term frequency. The idea behind the weighted greedy method is to select the query with higher requirement degree as early as possible. The work process of this query selection strategy is illustrated by the following example.

Example 1: *qw is the weight of a query. It is the sum of the document weights for the query. In table 1, the minimum df/qw value on this step is* 1.72*, which is for query* q_1*. So* q_1 *is added into the appropriate query set.*

document ID	q_1	q_2	q_3	q_4	q_5
d_1	0.5	0	0	0	0.5
d_2	0	0.5	0.5	0	0
d_3	1	0	0	0	0
d_4	0.33	0.33	0	0	0.33
d_5	0	0.33	0.33	0.33	0
d_6	0.5	0	0	0.5	0
d_7	0	0.5	0	0	0.5
d_8	0	0	0	0	1
d_9	0	0.33	0.33	0	0.33
df	4	5	3	2	5
qw	2.33	1.99	1.16	0.83	2.66
df/qw	1.72	2.51	2.59	2.41	1.88

TABLE 1: Example: weighted greedy algorithm

1.3 TS-IDS Algorithm

In [39], Wang et al. further modified their weighted greedy algorithm to $TS-IDS$ algorithm in order to have a better performance in query selection. The authors define IDS (the inverse of document size) as a measurement of the documents weight, and TS (term size), as the number of documents it can cover, i.e. the document frequency. Further, they have assumed that the importance of a document is proportional to the minimal term size in it. The reason is that: a term with higher TS will bring about large number of duplicates because more documents could be returned by issuing it to search interface; On the other hand, small terms are usually with less redundancy in crawling process. Based on their experiments, the $TS-IDS$ method decreases the redundancy rate compared with greedy algorithm [30] and IDS algorithm [38] on a variety of datasets.

Example 2: *In the previous example, the unit weight of a document* $d_i = \frac{1}{df} \times min(TS)$*, and the weight of each query is the sum of unit documents weight. The*

query with maximum μ/df will be selected. Here, $\mu = min(TS)$. Therefore, in table 2, q_5 should be appended to the selected query set.

document ID	q_1	q_2	q_3	q_4	q_5
d_1	2	0	0	0	2
d_2	0	1.5	1.5	0	0
d_3	4	0	0	0	0
d_4	1.33	1.33	0	0	1.33
d_5	0	0.66	0.66	0.66	0
d_6	1	0	0	1	0
d_7	0	2.5	0	0	2.5
d_8	0	0	0	0	5
d_9	0	1	1	0	1
df	4	5	3	2	5
μ	8.33	5.99	2.16	1.66	11.83
μ/df	2.08	1.20	0.72	0.83	2.37

TABLE 2: Example: $TS - IDS$ algorithm

1.4 Incremental Queries

In great majority of incremental deep web crawling approaches, appropriate queries or HTML forms [17][2] [26][27] are usually selected from documents or web pages that have been downloaded in previous steps. The number of documents in sample increases as more queries were sent to original database. This process is iterated until some terminate conditions were satisfied.

Ntoulas et al. [42] proposed an incremental method on textual database to generate a near-optimal solution by analyzing and examining the documents returned from previous steps. The basic idea of this method is that we can approximately get a good prediction on how many pages or documents will be returned by every word on each step. The frame work of their approach is the following:

1. Select a term as initial query q_1 and send it to web site.
2. Acquire result index pages and download new documents from internet.
3. Predict the next 'best' query, then send it to site for retrieving and downloading new pages.
4. Iterate this process until some termination criterion is met.

Query selection strategy

In order to generate the optimal query on each iteration, the author estimate the next potential query q_i by:

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

where P means the fraction of pages that q_i can return. $P_{new}(q_i)$ is the approximate fraction of new documents that will be returned by q_i:

$$P_{new}(q_i) = P(q_1 \vee \cdots q_{i-1} \vee q_i) - P(q_1 \vee \cdots \vee q_{i-1})$$

$Cost(q_i)$ represents the total cost of issuing q_i to web site, and it can be separated into three aspects: submitting a query q_i, retrieving result pages and downloading new documents.

Additionally, the authors have reported on their experiments by three policies, namely random, generic-frequency and adaptive on four real hidden websites. The results show that their adaptive method is able to retrieve and download most of the documents in deep websites by sending the least number of queries compared with the other two policies.

2 Document Retrieving Strategies

2.1 Automatic form filling

In Google's deep web crawling [32], the authors select good candidates after processing and examining input values from HTML forms [24][31][15]. The technique they

developed aims a large coverage in deep web contents, rather than focusing on specific web sites. Their surfacing method has already been incorporated into Google search engine, and their results can drive more than a thousand queries per second today.

Form filling

In this paper, a HTML form is defined within a form tag just like figure 2.1. There

```
<form action='http://books.com/find' method='get'>
  <$input type='hidden' name='src' value='hp'>
  Keywords: <input type='text' name='keywords'>
  Province: <select name='province'> <option value='Any'/>
    <option value='ON'/> ... </select>
  Sort By: <select name='sort'> <option value='relevant'/>
    <option value='price'/> ... </select>
  <input type='submit' name='s' value='go'>
</form>
```

FIGURE 1: HTML form in Google deep web crawling

are several input value in the form, but only select menus (selection inputs) and text boxes (presentation inputs) were considered here. In their work, selection inputs were usually assigned a wild card value that matches some records in their database. The presentation inputs is often treated as sort order or HTML layout. Therefore, the problem of query selection can be seen as the problem of selecting an appropriate form contents in *SQL*:

select $*$ **from** *DB* **where** *selection inputs* **ordered by** *presentation inputs*

Also, in their assumptions, the value of text boxes could be a null string, but for select menus, they assigned a default value as a wild card value. Figure 2 shows what is selection input, presentation input and text box in amazon.ca.

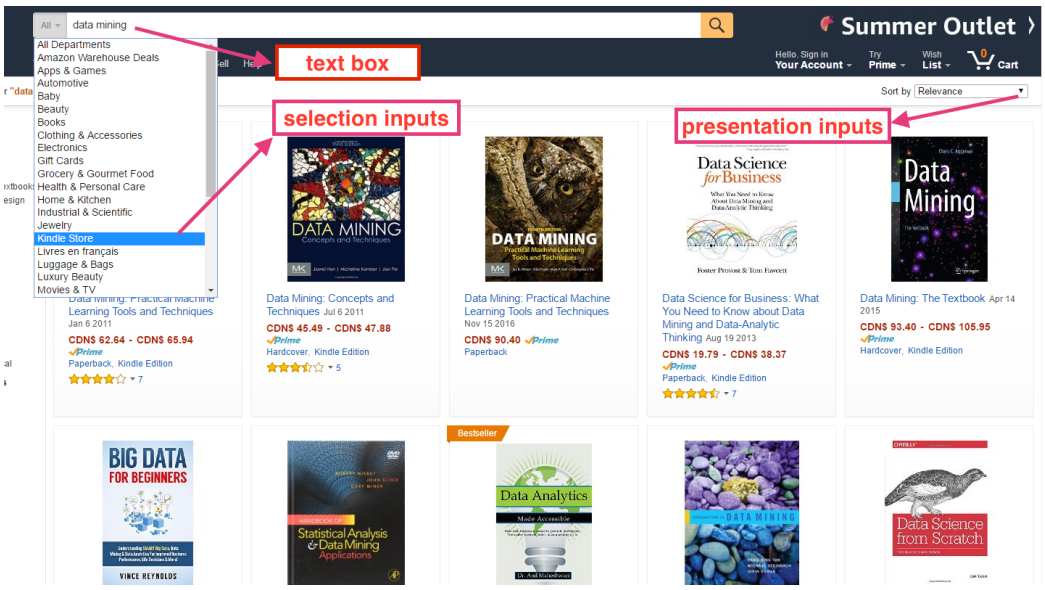


FIGURE 2: Text box, selection inputs and presentation inputs

Query selection strategy

The purpose of that paper is to receive a good coverage on each web site by issuing a small number of queries (form submissions). For this reason, they introduce TF-IDF [35] to select terms from a page after examining the words most relevant to its contents. In their deep web crawler, TF (term frequency) is a measurement for the importance of the term on that corresponding web page. IDF (inverse of document frequency) measures the importance of the word among all possible pages. That is to say, TF-IDF can balance the word's significance on the particular page with its general significance by $tf\text{-}idf(word, pages) = tf(word, page) \times idf(word)$.

In their experiments, initially, they take top 50 words with the largest tf-idf value into query pool. Only the top 25 words were added into query pool in next steps. Furthermore, some useless words with high term frequency (over 80%) or only appeared on one page will be removed from the query pool. Then they submit the remaining queries in query pool to original database and a new sample will be downloaded. This process is iterated until some termination condition is met.

2.2 Web Database Retrieval

As we all know, the majority of web databases are highly dynamic in practical applications. So a problem has to be addressed about how to maintain the consistency between the local database and the integrated web database. In [20], the author proposed an incremental crawling method to update their local database by several queries generated by their incremental harvest model. That is to say, their work can be divided into two phases: the first one is how to build the incremental harvest model; Second one is selecting the appropriate queries. Furthermore, they define three measurements to evaluate the performance of their crawler in their experiments, which are TRC (Total Coverage Rate), ICR (Incremental Coverage Rate) and EFF (Efficiency).

Incremental harvest model

In their paper, Huang et al. use $WDB(t_i)$ to represent all the records in web database at time t_i. So the incremental records between the neighbouring samples, such as t_i and t_{i+1} can be calculated by:

$$\Delta WDB(t_{i+1}) = WDB(t_{i+1}) - WDB(t_i)$$

Next, they build the training sample automatically through these incremental records, and the incremental harvest model was obtained.

Query selection

The query selection problem is transferred to Set Covering Problem by keeping records and queries in a document-term matrix. This is similar to [30]. Firstly, the initial query q_i was selected randomly from their initial query list (pool), then some new records will be retrieved and downloaded to local database by submitting q_i to the web database. Afterwards, an optimal query is iteratively chosen on each step by their incremental harvest model. This process is terminated when no more new records were returned within recent 10 queries.

2.3 Domain Specific Crawling

From previous work, we know that the only way to crawling deep web data is by sending some appropriate queries to search interface [3][22][36]. Moreover, some strategies about query selection problem have already been introduced. Most of them usually focused on receiving the largest coverage in original database, i.e. breadth-first. Actually, there are also some cost-effective deep web crawling methods for specific topics [25], i.e. depth-first.

In [9], a Domain specific Hidden Web Crawler (AKSHR) was proposed. Their framework can be divided into four phases: Firstly, there is a search interface crawling which can retrieve and store domain-specific search interface by carrying out domain-specific-assisted approach automatically; The second phase is a domain-specific interface mapping. The authors use the Domain Specific Interface Mapper (DSIM) [8] to detect the semantic distance between the components of different web interfaces in the same deep web site. The third phase consists of an automatic form filling which can generate queries by collecting labels and corresponding values from returned pages; The last phase is page analysis, which distinguishes whether the result pages returned by queries on last step is right or wrong. Additionally, they use three measurements ($Precision$, $Recall$, and $F-measure$) to evaluate the performance of their crawling results in experiments.

In [21], Jiang et al. provide a deep web crawler framework using machine learning techniques. Because some existing crawling methods ignored the experience gained from previous queries, in this paper, they analyze and conclude the environment information on each given step. Furthermore, the crawler acquires this information and selects the next query by reward calculation (Q-value) based on the last sample.

Moreover, in [40], the author further proposed a topic specific deep web crawling method by applying weka as their classification arithmetic to analyze the semantic webs. They use $HarvestRatio$ as the measurement to evaluate the performance of their experiment results.

Generally speaking, the framework of deep web domain specific crawling can be

describe by the following figure 3

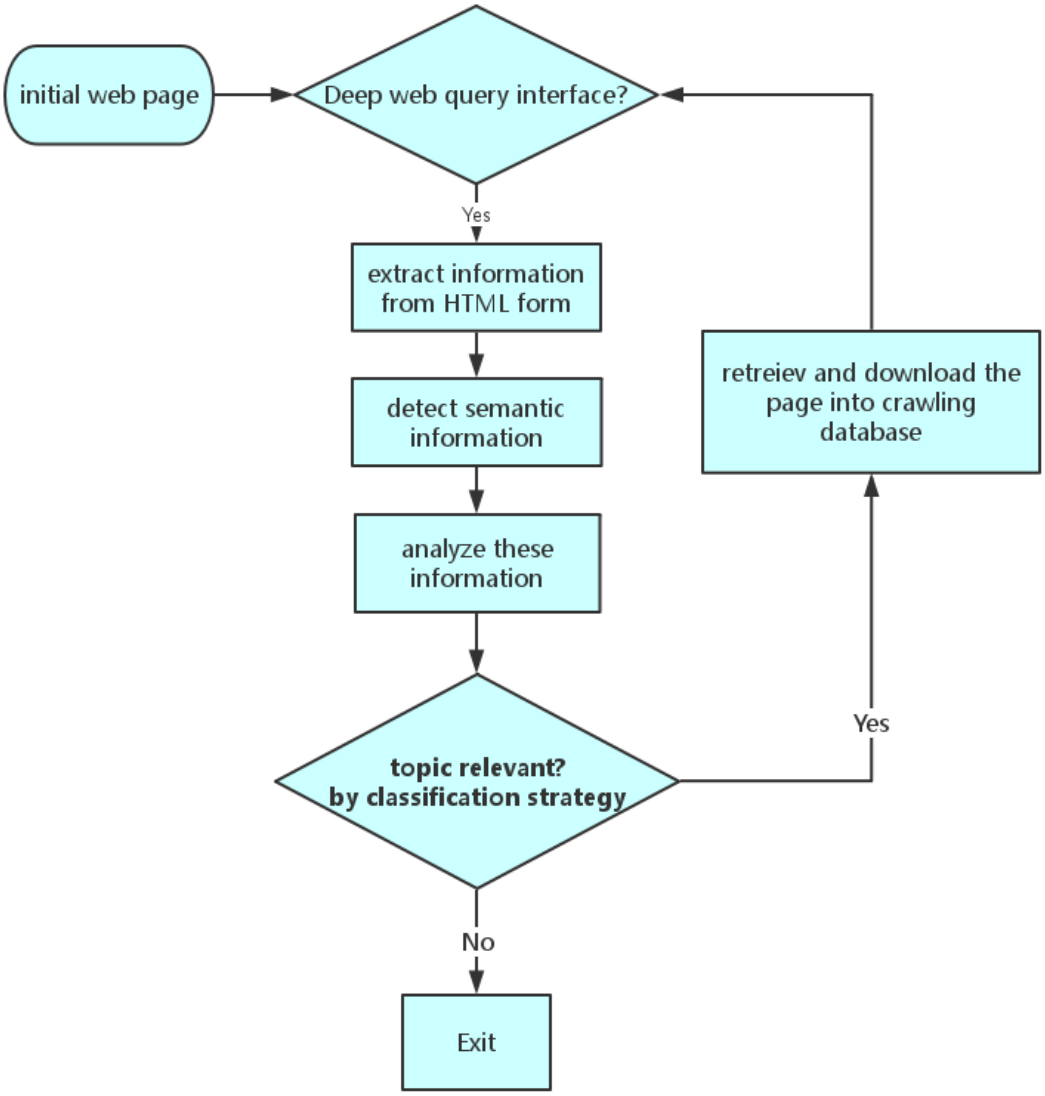


FIGURE 3: Work flow of topic focused deep web crawling

3 Summary

In this chapter, we have reviewed some research work which are relevant to deep web crawling. The present thesis is mainly based on the original sampling based [30] and

incremental approaches [42]. In these two papers, the authors have presented algorithms and results that are very helpful in our work. On the other hand, their studies may be more reasonable if they take both the sending cost and the downloading cost into consideration. In our thesis, we would like to add some parameters and criteria on their work. We will show the corresponding results in different situations.

CHAPTER III

Background

In this chapter, we discuss the preliminaries of our work in two sections. In Section 1, we will show the framework and relationship between one-step sample-based and incremental approach. In Section 2, we introduce how to transfer the query selection problem into set-covering problem.

1 Framework of crawling approaches

In this thesis, in order to concentrate on analyzing the performance of one-step sampling approach [30] and incremental approach[42] under different criteria, we build our own deep web crawler. In addition, we present the framework of these two approaches in this section to show a rough work process of them.

1.1 One-step Sampling Approach

The framework of one-step sampling approach which is similar to [30], and the work flow diagram is showing in figure 4. Firstly, we randomly collect some documents as the initial sample D, and query pool QP is created by selecting some terms from sample D. Then a query set Q with some appropriate queries are generated and saved based on our query selection method. After that the corresponding result documents which can be covered by Q will be retrieved from original datasets DB.

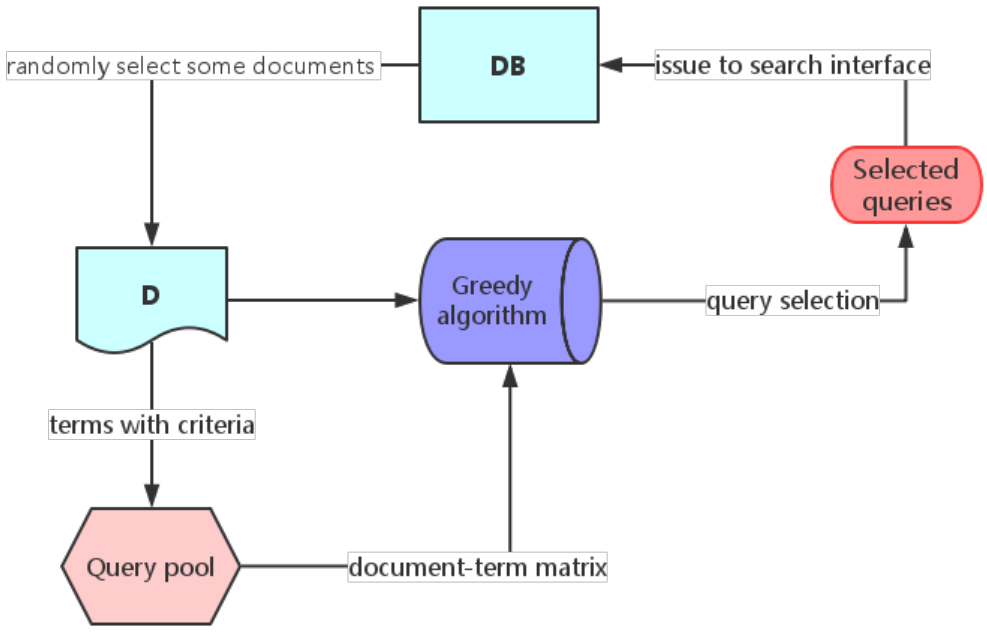


FIGURE 4: Flow diagram of one-step sampling based approach

1.2 Incremental Approach

With incremental strategies [42], queries are selected step by step, and new documents are returned and saved as more queries were generated and issued. We follow this same line of approach here. On first step: We randomly select some documents from original DB as initial sample D_1, which could be the same sample with one-step sampling approach. Then, a set of optimal queries Q_1 will be generated by greedy algorithm based on the sample D_1. After that, we issue Q_1 to original database DB, and a new sample D_2 will be returned and downloaded to our local database. On next steps: The process similar to the previous step is repeated until some termination condition is met.

In our studies, the stop criteria could be any cost constraint. Specially, one-step sampling approach can be seen as one part (first step) of incremental one. However, the final results of these two approaches should be different in practice. The reason is that: the total resources used for crawler are usually limited. We have to split all the

resources into pieces in incremental approach. But for one-step sampling approach, we do not need to do this.

2 Set-covering Model

In our study, the query selection problem is to select a set of appropriate queries which can cost-effectively retrieve as many new documents as possible with the least cost. In previous paper [30], it is found out that the query selection problem can be seen as the set-covering problem (SCP) [5][11][16][29] which is NP-hard [14]. So the greedy algorithm has been used as a popular solution for this problem.

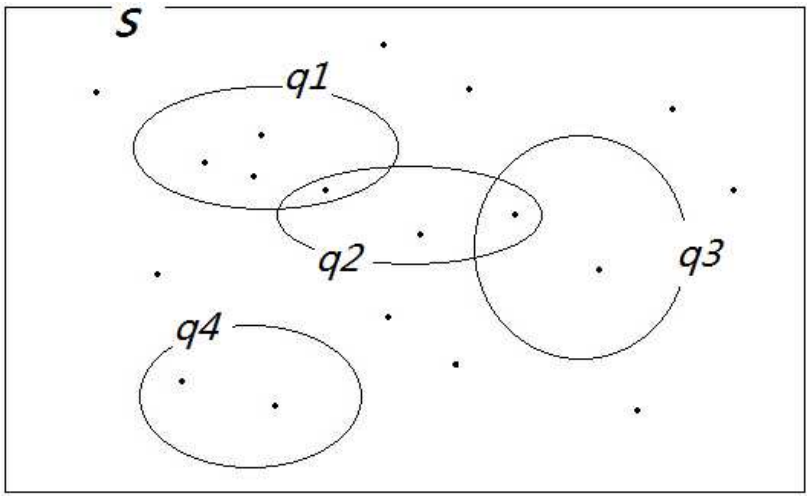


FIGURE 5: Set covering problem

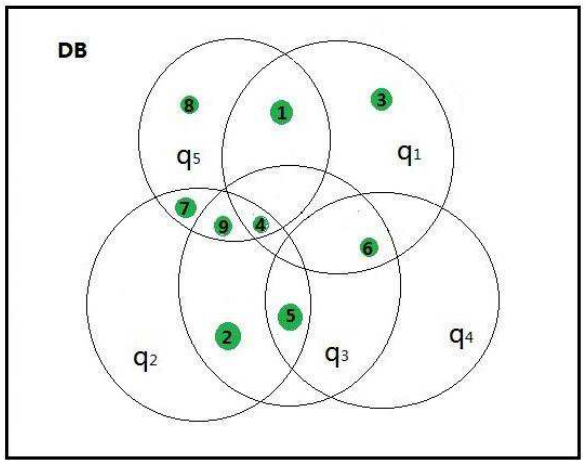
In figure 5, suppose S is the total database with all documents (point) in it. If a point is in a circle (q), we indicate that this document can be covered or returned by this query q. Therefore, the query selection problem in dataset S is to find a set of q (circle) which can cover more documents with a minimal cost. I.e. Set-covering problem

2.1 Document-term Matrix

Definition 1: ***(Document-term Matrix)*** *Given a set of documents* $D = \{d_1, d_2, \cdots, d_m\}$ *and a set of terms* $QP = \{q_1, q_2, \cdots, q_n\}$*, their relationship can be represented by*

the document-term matrix $A = (a_{ij})$ *where* $a_{ij} = 1$ *if the document* d_i *contains the term* q_j*, i.e.* d_i *can be covered by* q_j*; otherwise* $a_{ij} = 0$*.*

Example 3: *A deep web data source* $DB = \{d_1, d_2, \cdots, d_9\}$ *is shown in figure 6 below, and query pool* $QP = \{q_1, \cdots, q_5\}$*). Each query is contained in at least one document. The corresponding document-term matrix is given on the right. One possible solution to this set covering problem is* $Q = \{q_1, q_3, q_5\}$*.*



$\Longrightarrow$

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

FIGURE 6: Example: Document-term matrix

2.2 Hit Rate and Overlapping Rate

From document-term matrix, we can define hit rate [30] to reflect the cost-effectiveness of the optimal query set Q in terms of its coverage.

Definition 2: ***(Hit Rate, HR)*** *Given an* $m \times n$ *document-terms matrix* $A = (a_{ij})$ *which comes from a datasets* $DB = \{d_1, \cdots, d_m\}$ *and a query pool* $QP = \{q_1, \cdots, q_n\}$*. A set of queries* Q *(*$Q \subseteq QP$*) is represented by an n-binary vector* $X = [X_1, \cdots, X_n]$*,* $X_j = 1$ *if* q_j *was selected to* Q*, otherwise,* $X_j = 0$*. Let* $Y = [y_1, \cdots, y_m]$ *be a binary m-vector and* $y_i = 1$ *if* $\sum_{j=1}^{n} a_{ij} \times X_j \geq 1$*, otherwise ,* $y_i = 0$*. The hit rate of* Q *in DB, denoted by HR(DB,Q), is represented by the ratio between the number of unique*

documents (u) collected by query set Q and the size of datasets DB.

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

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

Additionally, we define overlapping rate [30] to represent the retrieving cost of all matched documents by queries in Q.

Definition 3: ***(Overlapping Rate, OR)*** *Given $Q = \{q_1, \cdots, q_k\}$, the overlapping rate of Q can be calculated by the ratio between the sum of the number of covered documents by each query in Q (v) and the number of of unique documents (u). In terms of document term matrix, this is defined as follows:*

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

2.3 Query Selection Method

Definition 4: ***(Query Selection)*** *Given an $m \times n$ binary matrix $A = (a_{ij})$, let $Cost(retrieve) = [c_1, c_2, \cdots, c_n]$ be a non-negative n-vector and each $Cost(retrieve_j) = \sum_{i=1}^{m} a_{ij}$ represents the retrieval cost of the query q_j. The total cost of query q_j is $Cost(q_j) = \beta + \sum_{i=1}^{m} a_{ij}$ where β is the ratio of unit cost between sending a query and receiving a document. Hence, the query selection problem is to search for a binary n-vector $X = [X_1, X_2, \cdots, X_n]$ that satisfies the objective function.*

$$\min \sum_{j=1}^{n} \left(Cost(q_j) \times X_j \right)$$

Subject to

$$\sum_{j=1}^{n} a_{ij} \times X_j \geq 1 \qquad (1 \leq i \leq m)$$

$$X_j \in 0, 1 \qquad (1 \leq j \leq n)$$

In the document-term matrix of a sample D, each column represents a potential query in query pool QP and each row represents a document in sample D.

$Cost(retrieve_j)$ is the document frequency of term q_j, which is the number of documents that can be returned by q_j.

There are many query selection strategies in deep web crawling, and they can be divided into three directions [30]:

- **Minimize cost**

  We select the next query q_k which has the smallest cost: $Cost(q_k) = \min\{Cost(q_j) \mid 1 \leq j \leq n\}$ and $X_j = 0$(q_j has not been selected yet) on each step. The lowest total cost for optimal query set Q is $\sum_{j=1}^{n}(Cost(q_j) \times X_j)$, which approximates the smallest cost.

- **Maximum coverage**

  Another query selection strategy is to select the next term q_k to cover the largest number of rows (documents) which are not yet covered by previous columns (queries). Namely, if we select q_k as the next column (query), setting $X_k = 1$ to maximize $\sum_{i=1}^{m}((1 - y_i) \times a_{ik})$ where $y_i \in \{0, 1\}$ and $y_i = 1$ if d_i has been returned, otherwise, $y_i = 0$. The largest coverage is approximated by covering most new rows(documents) on each step.

- **New/Cost**

  In our studies, we combine the previous two strategies together

  $$\frac{new}{cost} = \frac{new\ returned\ documents\ by\ q_k}{Cost(q_k)}$$

  One of the issues here is how do we know the number of new documents that would be returned by q_k. Let $S(Q_{k-1})$ be the set of documents which have already been returned by previous query set $\{q_1, q_2, \cdots q_{k-1}\}$. We can calculate $S(Q_k) - S(Q_{k-1})$ by $\sum_{i=1}^{m}((1 - y_i) \times a_{ik}$ for every potential query in QP, i.e. the number of new documents that could be retrieved by q_k, where $y_i \in \{0, 1\}$ and $y_i = 1$ if d_i has been returned, otherwise, $y_i = 0$. Therefore, we have:

  $$\frac{new}{cost}(q_k) = \frac{\sum_{i=1}^{m}((1 - y_i) \times a_{ik})}{\beta + \sum_{i=1}^{m} a_{ik}} \tag{1}$$

  In algorithm 1, we show a specific query selection method.

Algorithm 1: Query selection algorithm

Input: an $m \times n$ matrix A $= (a_{ij})$, β

Output: optimal query set Q

$$Cost(q_j) = \beta + \sum_{i=1}^{m} a_{ij}$$

$$X_j = 0(1 \leq j \leq n)$$

$$y_i = 0(1 \leq i \leq n)$$

while $\sum_{i=1}^{m} y_i < m$ **do**

/* generate the query with maximum *new*/*cost* */

for *all potential queries* q_k *in* QP **do**

$$\frac{new}{cost}(q_k) = \frac{\sum_{i=1}^{m}((1 - y_i) \times a_{ik})}{\beta + \sum_{i=1}^{m} a_{ik}}$$

end

append q_k with maximum $\frac{new}{cost}(q_k)$ into optimal query set Q

$$X_k = 1$$

$$y_i = 1,\ a_{i,k} = 1\ for\ (1 \leq i \leq n)$$

end

return Q

Example 4: *Consider the document-term matrix in figure 6 as our original dataset DB, and* $\beta = 10$*. A random initial sample D is given in figure 7. When we apply algorithm 1 on DB:*

	q_1	q_2	q_3	q_4	q_5
d_1	1	0	0	0	1
d_2	0	1	1	0	0
d_4	1	1	0	0	1
d_5	0	1	1	1	0
d_6	1	0	1	1	0
d_9	0	1	1	0	1

$\Longrightarrow$

	q_1	q_2	q_3	q_4	q_5
d_1	1	0	0	0	1
d_2	0	1	1	0	0
d_4	1	1	0	0	1
d_5	0	1	1	1	0
d_6	1	0	1	1	0
d_9	0	1	1	0	1
df	3	4	4	2	3
$Cost(q_k)$	13	14	14	12	13
$new/cost$	0.23	0.29	0.29	0.17	0.23

$\Longrightarrow$

	q_1	q_2	q_3	q_4	q_5
d_1	1	0	0	0	1
d_2	0	1	1	0	0
d_4	1	1	0	0	1
d_5	0	1	1	1	0
d_6	1	0	1	1	0
d_9	0	1	1	0	1
df	3	4	4	2	3
$Cost(q_k)$	13	14	14	12	13
$new/cost$	0.15	0	0.07	0.08	0.08

FIGURE 7: Example: query selection

1. *q_2 was selected, which has the largest $\frac{new}{cost}(q_2) = 0.29$ in QP, and d_2, d_4, d_5, d_9 will be returned by q_2. Now X = [0,1,0,0,0], $y_2, y_4, y_5, y_9 = 1$, q_2 is appended*

into Q, $\sum_{i=1}^{6} y_i = 4 < 6$. *Note:* q_2 *and* q_3 *have the same* $\frac{new}{cost}$, *we randomly select one of them in this situation.*

2. q_1 *was selected, which has the largest* $\frac{new}{cost}(q_2) = 0.15$ *in* QP. *After this step,* X *= [1,1,0,0,0],* $\sum_{i=1}^{6} y_i = 6$, *so we append* q_1 *into* Q. *All documents in sample* D *are covered by* Q, *so the while loop in algorithm 1 is terminated.* $Q = \{q_2, q_1\}$ *is returned by our query selection algorithm. The HR(DB,Q) =* $\frac{8}{9}$ *= 0.89, OR(DB,Q) =* $\frac{4+5}{8}$ *= 1.125, because* d_4 *was returned two times by* Q.

CHAPTER IV

Method

In the previous chapter, we have shown some preliminary knowledge about deep web crawling. In this chapter, we present our method. In section 1, we focus on giving an overview of our method. In section 3, we propose three hypotheses, after which we explain how to generate a sample-condition relation diagram and the equal splitting strategy based on these three hypotheses.

1 Overview

In our studies, we aim to provide a guideline for users to choose a more cost-effective and efficient crawling strategy in crawling deep web. With this objective, we need to define some parameters to measure the performance of a crawler in different situations, such as the coverage, cost model and quality. After that, we will explain the work process of our method in section 1.4.

1.1 Performance Parameters

Coverage in Original Datasets

In our work, the coverage in original dataset DB refers to the ratio between the number of documents that contain at least one of the selected queries in Q and the total number of documents in DB, which is defined as hit rate (HR) in [30]. In our work, we name the coverage in DB as $HR(Q, DB)$, and it can be used to evaluate the efficiency of a query set Q. Similarly, the coverage in sample D can be represented by $HR(Q, D)$. Therefore, no matter in which kind of strategy, $HR(Q, DB)$ can be

calculated as following:

$$HR(Q, DB) = \frac{number\ of\ documents\ containing\ Q}{|DB|} \tag{1}$$

$|DB|$ means the number of documents in original database DB.

Cost Model

As pointed out in many papers on deep web crawler, the crawling cost comes from many sources, such as the network traffics caused by receiving documents from DB or the limitation on total number of queries that can be issued by the users daily.

In our work, for each query in Q, the unit receiving cost is represented by α. Unless explicitly mentioned, in the following, we will use numeric value 1 to denote the unit receiving cost. The cost of receiving 1000 documents, for example, will be 1000.

The unit cost of sending one query is represented by β, with integer values. When α is nonzero, it can be viewed as the ratio of the cost between sending one query and that of receiving one document. If $\alpha = 1$, $\beta = 1500$, and there are 500 documents in the data set containing term q, for example, the sending cost of q will be 1500, and the receiving cost of q will be 500. The total cost of q will be 1500 + 500 = 2000. The cost model for each query is:

$$Cost(q) = \beta + \alpha \times number\ of\ documents\ containing\ q \tag{2}$$

where α is usually fixed as 1 in this thesis.

Quality

Let Q be a set of queries, and D is a data set. The quality of a query q w.r.t. (Q, D) is measured by the per unit cost return of the number of documents that are obtained by sending query q to D and that are not obtained by sending any other queries in Q to D. Note that the cost is calculated for both sending and receiving.

During the query selection process, ideally, we determine the quality of a term with respect to (Q, DB) where Q is the set of queries that have been used to build

the current sample, and DB is the original data set. However, during the crawling process, we do not have complete information about the original data set. Thus, we approximate this quality by measuring the quality of q w.r.t. (Q, S) where Q is the set of queries already selected by the query selection process during the current round of iteration, and S is the sample data set. The quality formula of sample S is given below:

$$Qual(q, Q, S) = \frac{number\ of\ new\ documents}{c} \tag{3}$$

A higher value of $Qual(q, Q, S)$ means this q can harvest more new documents with less resources.

1.2 Revising Cost (k-value)

In the adaptive query selection process, the document retrieval is realized by iteratively selecting queries followed by downloading the corresponding documents. This process is adaptive in the sense that after each iteration of document downloading, the newly retrieved documents are merged into the sample data set for update before processing the next round of query selection.

We assume that the queries selected from the updated sample possess better quality, compared to those selected from the sample without update. There are two sources that may lead to such quality improvement.

- We have improved knowledge, obtained from the previously downloaded documents from the same data source, about the terms used in the original data set. This information is used to enrich the sample for query selection.

- During the query selection process, we determine the quality of a term based on its quality measured in the sample data set, in order to approximate the same measurement in the original data set. Such estimation is generally better in the updated data set.

This adaptive retrieval process requires multiple updates of the sample. Updating sample data set is costly in terms of the amount of time needed for document down-

loading, and the computational resource consumed to construct the updated matrix, to generate the query pool, and to select queries.

We use an integer number k to represent the cost of one round of sample update. When α is nonzero, it can be viewed as the ratio between the cost of one round of sample update and that of receiving one document. If $k = 1000$, for example, and there are in total 5 iterations of sending and receiving in the crawling process, the total cost for revising the sample data sets will be $1000 * 5 = 5000$.

Following the stepwise refinement approach, the problem here is how many steps should we take and how to divide the iteration steps, in order to maximize the overall result of document retrieval within limited cost budget, or minimize the cost to reach the quality document retrieval.

1.3 Stop Criteria

In deep web crawling, the queries are usually selected from the query pool sequentially [30][38][39][42]. There are basically three criteria to terminate the selection:

1. All documents in original datasets are retrieved by Q, or $HR(Q, Original)$ is high enough, to determinate the crawling process.

2. The crawling process has to terminate due to cost constraint.

3. The set of queries already selected has met the condition for the current round of iteration

Here we study the best conditions for the third one. Such a condition, called refinement condition, may include, for example:

- the total number of queries already selected

- the hit rate of the set of queries already selected

In our setting, this refinement condition is a cost limit in the sense that: The query selection process for the current round should terminate if the cost of the queries already selected has hit this cost limit. Here, the cost of a query includes

both the cost of sending this query, and the cost of retrieving all the documents that could be retrieved by this query. The cost of a set of query is the total sum of the cost of each query in this set.

1.4 Work Process

Through empirical study, we explore the best cost limit for the refinement condition which determines the length of the steps. Such cost limit varies with different sample sizes: when the sample is large, the cost limit will be larger, leaving the refinement condition loose. In this situation, the iteration is in favour of the sample-based crawling.

Actually, the result of our study is a sample-condition relation (SCR) that provides the refinement condition based on the sample size. This sample-condition relation provides practical guidelines for cost-effective stepwise refinement in iterative document retrieval. For example, when sample size is $|D|$, users can find the corresponding optimal cost condition is $SCR(|D|)$ by checking our sample-condition relation diagram.

Given a real deep web crawler, with the initial sample D. Using SCR, we can find that the optimal cost condition is $SCR(|D|)$. Suppose the total cost budget is $costT$, HR_{target} refers to the target of hit rate in original database. A flow diagram for users is given in figure 8. Note that when the first query is selected, Q is empty. The frame work of our method is explained here:

1. Set $SCR(|D|)$ as the cost limit on first round by checking our sample-condition relation diagram.

2. Apply greedy algorithm on D and QP, then select one optimal query q, calculate $Cost(q)$ by formula 1.1.

3. Let $Cost(Q) = Cost(Q) + Cost(q)$, and check whether $Cost(Q) < SCR(|D|)$ and $HR(Q, Original) < HR_{target}$ and $SCR(|D|) < costT$.

4. If the answer is yes, add q into Q, go back to step 2 and select the next query.

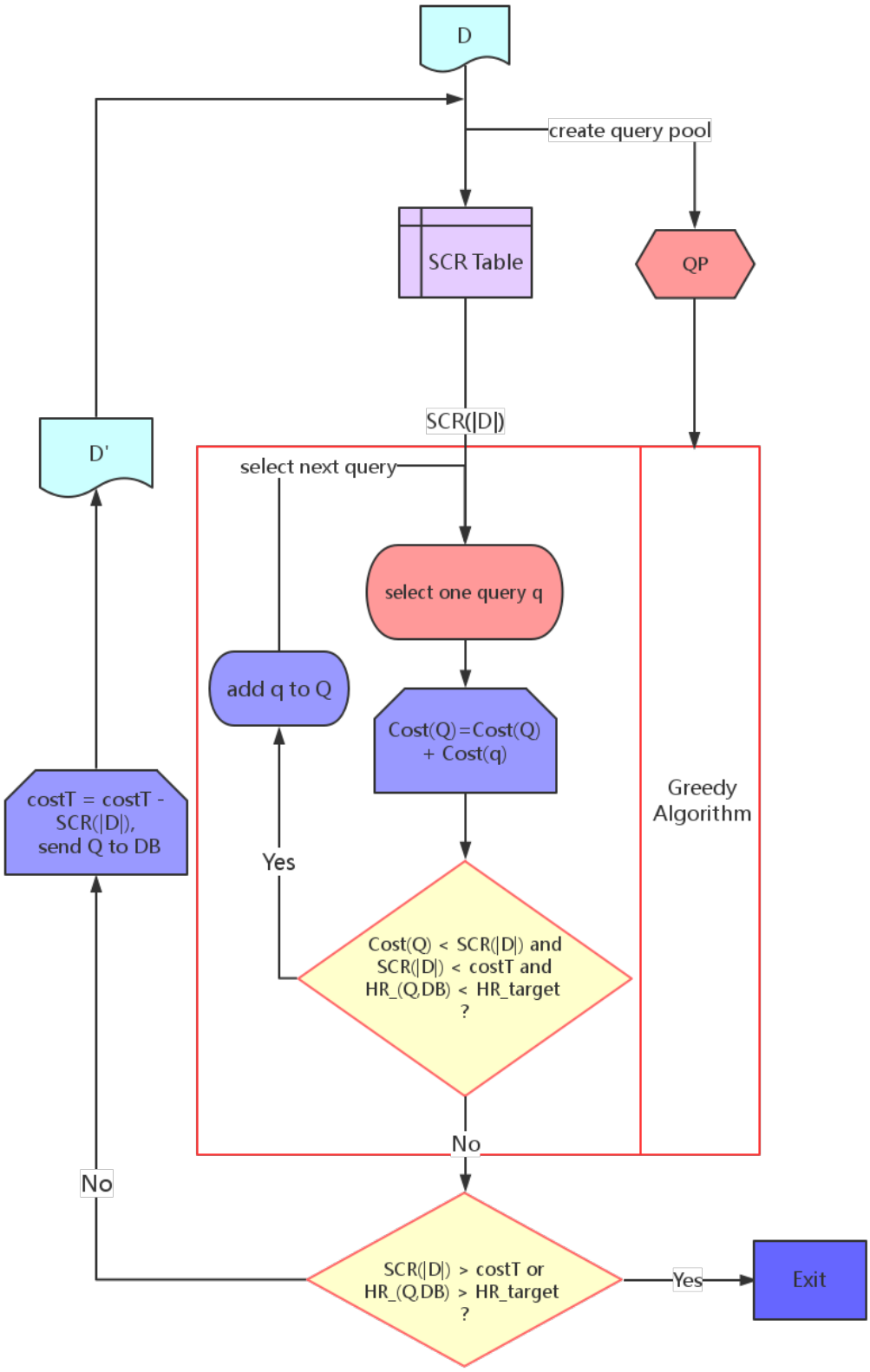


FIGURE 8: Flow diagram of work process

Otherwise, set the total cost budget $costT$ to $costT - SCR(|D|)$. And issue all queries in Q to DB. New documents will be retrieved and downloaded to local database to construct new sample D'. Check the sample-condition relation diagram, obtain a new cost limit $SCR(|D'|)$ for D'. After that, proceed to the next round by following step 2 again.

Special Cases

In normal cases, we set α to 1, so β can be treated as the ratio between the unit cost of sending a query and receiving a document. There are also some special cases we have to consider.

1. $\alpha = 0$

In this situation, the cost limit is uniquely determined by the number of queries. So the refinement condition expresses the constraint on the total number of queries to select as the length of the step.

Example 5: *Suppose that there is only sending cost and no receiving cost ($\alpha = 0$). The sample-condition relation is reduced into the sample-querynumber relation.*

- *Suppose that the sample size is initially 20. According to the sample-querynumber relation in 9, we should select 5 queries for the first round.*
- *Suppose that the first two stop criteria in section 1.3 are not met, and that after sending the first 5 queries, the sample size is updated into 100. By sample-querynumber relation, we should select 11 queries for the next round.*
- *Suppose that the first two stop criteria in 1.3 remain unsatisfied. We send these 11 queries in the previous step to DB, and the sample size is updated into 300. According to 9, we should select 14 queries for the next round.*

This process is iterated until one of those three terminate conditions in section 1.3.

2. $\beta = 0$

In this situation, we do not need to consider the sending cost of each query in Q. This is similar to what was introduced in Lu et al. [30]. In their work, they apply

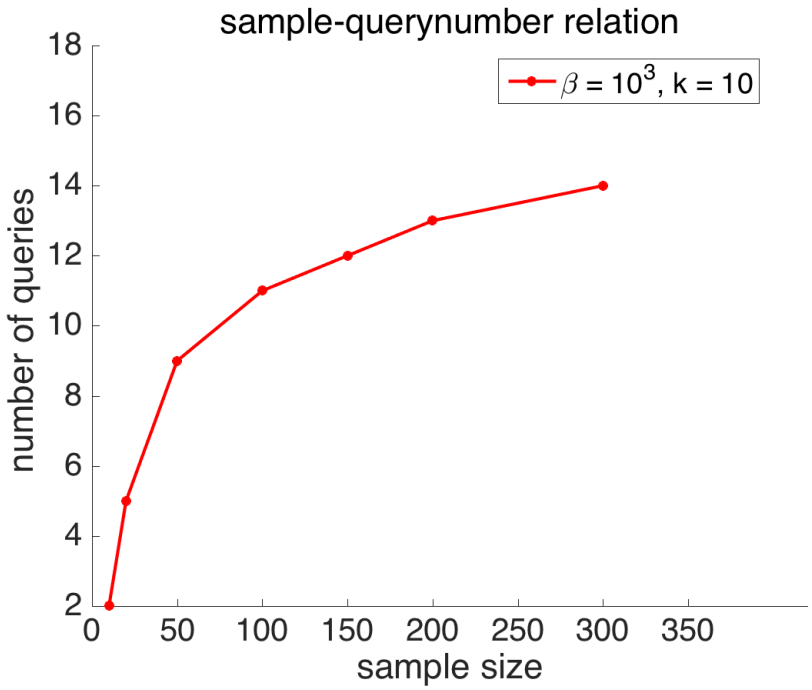


FIGURE 9: Sample-querynumber relation, $\alpha = 0$

the cost of retrieving URLs as the total cost in their studies. We will to provide more detail in chapter V to this situation.

3. $k \to 0$

When $k \to 0$, the updating cost is almost ignorable. In this situation, the suggested refinement condition is the tightest: terminating the query selection soon. To the extremist, we could send the queries one by one, updating the sample after each query.

4. $k \to \infty$

When $k \to \infty$, on the other hand, the update cost is too high that we should almost always go with the sample-based approach to send out all the queries at once. The refinement condition in this situation is quite loose so the query selection process mostly terminates with the first two conditions.

More precisely, a sample-condition relation on newsgroup data set with different k value is given in 10, where $\beta = 1000$.

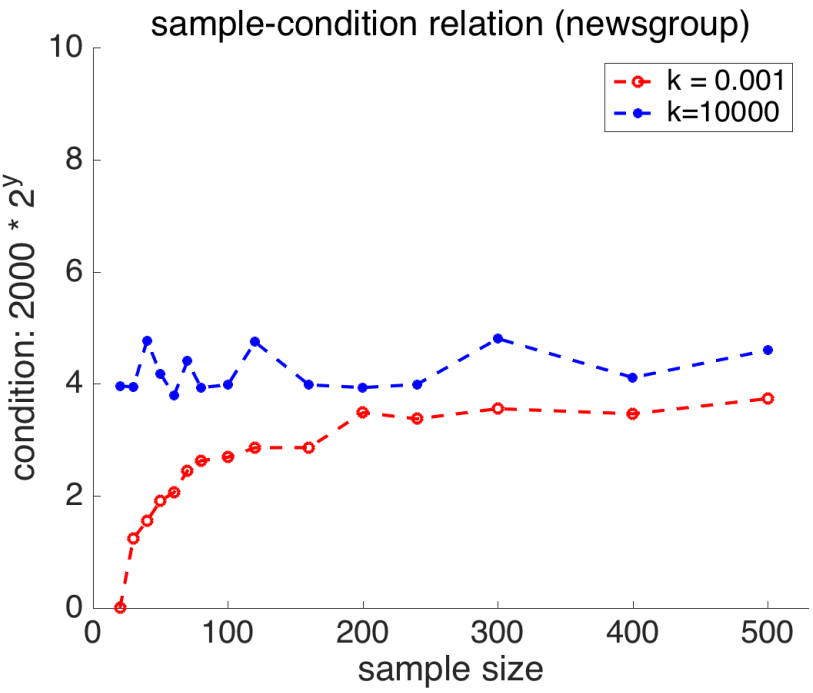


FIGURE 10: Sample-condition relation, $k = 0.001$ and $k = 10^4$

2 Problem Formalization

In our study, based on our cost model, we propose a method to determine the near optimal lengths of the steps in terms of cost limit. On the other hand, we further observe that optimal lengths vary according to different sizes of the sample dataset.

Given an initial sample S_0 and total cost limit L. The aim of our study is to determine a positive integer n and positive integers set $\{x_1, x_2, \cdots, x_n\}$ to maximize

$$\sum_{i=1}^{n} q_i$$

subject to

$$\sum_{i=1}^{n} c_i < L$$

where x_i represents the number of queries sent on the i^{th} step, i.e. the length of this step. q_i represents the quality obtained by sending x_i queries on the i^{th} step. It can be determined on S_0 and $\{x_1, x_2, \cdots, x_i\}$. c_i represents the cost spent on the i^{th} step, including the sending and receiving cost for the x_i queries. It also can be determined

on S_0 and $\{x_1, x_2, \cdots, x_i\}$. n is the total number of steps. Given cost limit L, n can only have finite values. For each value n, variable x_i (for $1 \leq i \leq n$) can only take finite values. Thus, this is a combinatorial optimization problem.

3 Theoretical Basis

The sample-condition relation is derived from the four data sets used for deep web crawling. We need to determine, for a given sample size, the best refinement condition (cost limit).

Let the sample size be fixed. In order to find the best condition, we study the condition-quality relation with the following two strategies:

- **single step strategy**
- **equal splitting strategy**

With the single step strategy, we use the given cost limit c to select queries to send to the original data set all at once. The quality of these queries is recorded as the quality of this condition c with single step strategy.

With the equal splitting strategy, we use 50% of the cost limit c to select the first part of the queries. This is followed by sample update with cost k. Then the second part of the queries are selected with the remaining cost $50\% \times c - k$. The quality of the queries from these two parts is then recorded as the quality of this condition c with the equal splitting strategy.

We use $Q(S_1, c)$ to denote the set of queries generated by greedy algorithm from sample S_1 under the cost limit c. With our context, the revising cost for updating sample is fixed as k. In addition, we define $Qual_1(|S_1|, c)$ as the quality of single step strategy by sending $Q(S_1, c)$.

$$Qual_1(S_1, c) = \frac{the\ number\ of\ new\ documents\ covered\ by\ Q(S_1, c)}{c}$$

Similarly, the quality of equal splitting strategy is define as:

$$Qual_2(S_1, c) = Qual_1(S_1, \frac{1}{2}c) + Qual_1(S_2, \frac{1}{2}c - k)$$

where S_2 is the updated sample after issuing $Q(S_1, \frac{1}{2}c)$ to DB.

3.1 Hypothesis I and Hypothesis II

We have the following two hypotheses on the condition-quality relationship. Let c and c_1 be two cost limits. Note that the cost k for updating sample data set is included in the total cost limit c.

Hypothesis 1: *If single step strategy yields better quality than equal splitting strategy w.r.t. (S_1, c), i.e. $Qual_1(S_1, c) > Qual_2(S_1, c)$, then it would generally also yield better quality than equal splitting strategy w.r.t. (S_1, c_1), if $c_1 < c$.*

The reason is this: In equal splitting strategy, k is fixed, and the crawler have to split the cost limit c into two parts. When cost limit is c_1, the available resources for two parts are $\frac{1}{2}c_1$ and $\frac{1}{2}c_1 - k$ respectively, which are both less than $\frac{1}{2}c$ and $\frac{1}{2}c - k$ because of $c_1 < c$. Therefore, the revising cost k plays a more important role in c_1 than in c. That is to say, compared to c, there are more resource deduction to do query selection for equal splitting than for single step when cost limit is c_1. With less queries generated, less new documents will be returned. When cost limit is c, $Qual_1$ is smaller than $Qual_2$, so given a smaller cost limit c_1, the quality of equal splitting strategy would generally not be better than single step strategy as well.

Hypothesis 2: *If single step strategy cannot yield better quality than equal splitting strategy w.r.t. (S_1, c), i.e. $Qual_1(S_1, c) < Qual_2(S_1, c)$, then generally it would not yield better quality than equal splitting strategy either w.r.t. (S_1, c_1), if $c_1 > c$.*

With a larger value of cost limit c_1, the revised sample S_2 can make a more accurate prediction to original DB than c, and the cost used in downloading new sample S_2 accounts for a smaller proportion in c_1 compared with c. I.e. k has less impacts on the results of equal splitting strategy. Consequently, for any $c_1 > c$, if $Qual_2(S_1, c) > Qual_1(S_1, c)$, we have $Qual_2(S_1, c_1) > Qual_1(S_1, c_1)$.

Based on these two hypotheses, we compare the condition-quality relations with the two strategies, and search for the maximum cost limit c with which the single

step strategy yields better quality than equal splitting strategy: when cost limit is larger than c, we should split the process into multiple steps.

We look for such refinement condition for a given sample size in this way:

- Use binary search to find a range $[a_1, a_2]$ such that it contains the refinement condition and $(a_2 - a_1 < \Delta)$;
- Use linear interpolation or average to find the refinement condition in range $[a_1, a_2]$.

The condition-quality relations with single step and equal splitting strategy on newsgroup datasets is given below, where sample size is 200.

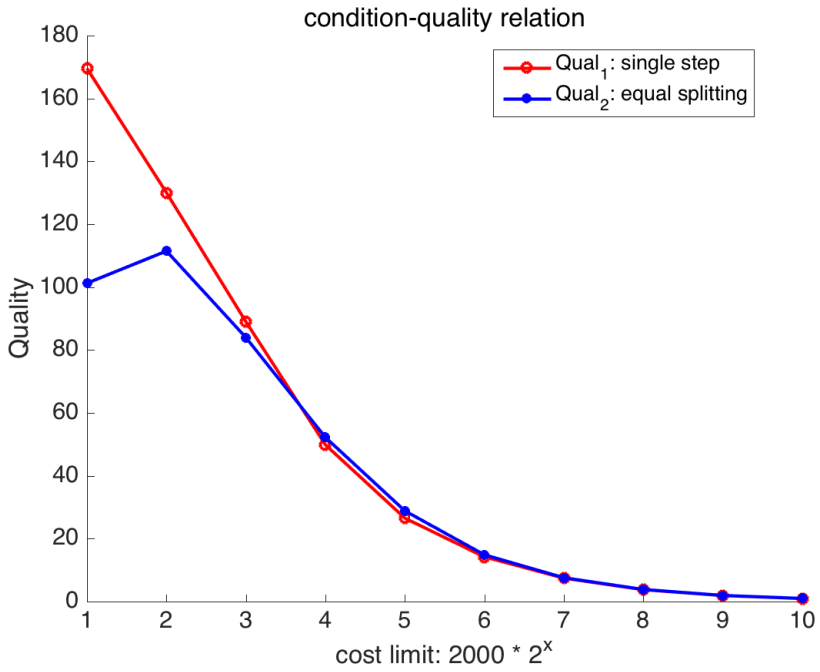


FIGURE 11: $Qual_1$ and $Qual_2$, sample size $= 200$, $\beta = 10^3$, $k = 10$

Example 6: *In the above graph, red is from single step strategy and blue is from equal splitting strategy. The maximum cost limit c for single step strategy to be better is around 15000.*

Suppose that we start from range $[1, 2^8]$. *The splitting strategy is better for* 2^8, *so we continue to consider range* $[1, 2^7] \cdots$. *Eventually, we locate the range* $[2^3, 2^4]$. *So*

the cost limit we are looking for is $2000 * (2^4 - 2^3)/2$.

Furthermore, in this condition-quality relation diagram, when $c_1 < 15000$*, single step strategy always yields better quality; when* $c_1 > 15000$*, equal splitting strategy always yields better quality. To be more precise, we have listed some corresponding data in Table 3 about this diagram.*

cost limit	$Qual_1$**: single step strategy**	$Qual_2$**: equal splitting strategy**
2^1	**169.5650**	89.3000
2^2	**130.0387**	106.1250
2^3	**88.9537**	82.9750
2^4	49.8272	**53.4937**
2^5	26.4997	**28.4844**
2^6	14.1763	**14.8453**
2^7	7.4150	**7.5172**
2^8	3.7420	**3.7961**
2^9	1.8745	**1.9115**
2^{10}	0.9385	**0.9785**

TABLE 3: $Qual_1$ and $Qual_2$, sample size is 200 in Newsgroup

3.2 Hypothesis III

We define the refinement condition c for a given sample size to be the maximum cost limit for single step strategy to be better than multi-step strategies. Based on the following hypothesis III, we adopt the condition c obtained from the above method with equal splitting strategy.

Hypothesis 3: *Using other splitting strategy in the above method will yield similar result of cost limit c.*

In precious contents, we define $Qual_1$ and $Qual_2$ to measure the quality of single step and equal splitting strategies. It should be noted that there are many other strategies. Here are some examples of other splitting strategy:

1. Also splitting once, but with different percentage for the two parts, e.g. 10/90 splitting, which means the first part of query selection should terminate with cost limit $10\% \times c$, while the second part terminates with cost limit $90\% \times c - k$.

2. Splitting more than once, e.g. $\frac{1}{3}/\frac{1}{3}/\frac{1}{3}$, which means the first part of query selection should terminate with cost limit $\frac{1}{3} \times c$; the second part terminates with $\frac{1}{3} \times c - k$; and the third part terminates with $\frac{1}{3} \times c - k$.

10/90 Splitting Strategy

For the first situation, compared with equal splitting strategy, although 10/90 splitting can not generate a good query set on first step, it would have a more accurate prediction to DB by updated sample on second step. Therefore, if equal splitting have a better quality than single step with refinement condition c, usually 10/90 strategy would also outperform the quality of single step strategy as well. On the other hand, if single step strategy yields better quality than 50/50 strategy, it is usually because the revising cost k accounts for a large proportion of the cost budget in the crawling process. Even though 10/90 splitting strategy distribute 90% of its total resources to the second step, which enriches the second part of the sending and receiving to yield better result, such benefit is balanced by less improvement of the sample due to less budget distributed to the first part of sending and receiving. That is to say the quality of 10/90 splitting would not be much better than equal splitting strategy. Furthermore, we show an example below.

Example 7: *Figure 12 shows condition-quality relationships between single step, equal splitting and 10/90 splitting strategies on newsgroup data set, where* $\beta = 10^3, k = 10$. *From this diagram, we can find that, no matter in 10/90 or 50/50 splitting strategy, the refinement condition c are within the range of* $[2^3, 2^4]$ *and* $[2^4, 2^5]$, *respectively for* $|S_1| = 200$, $|S_1| = 400$.

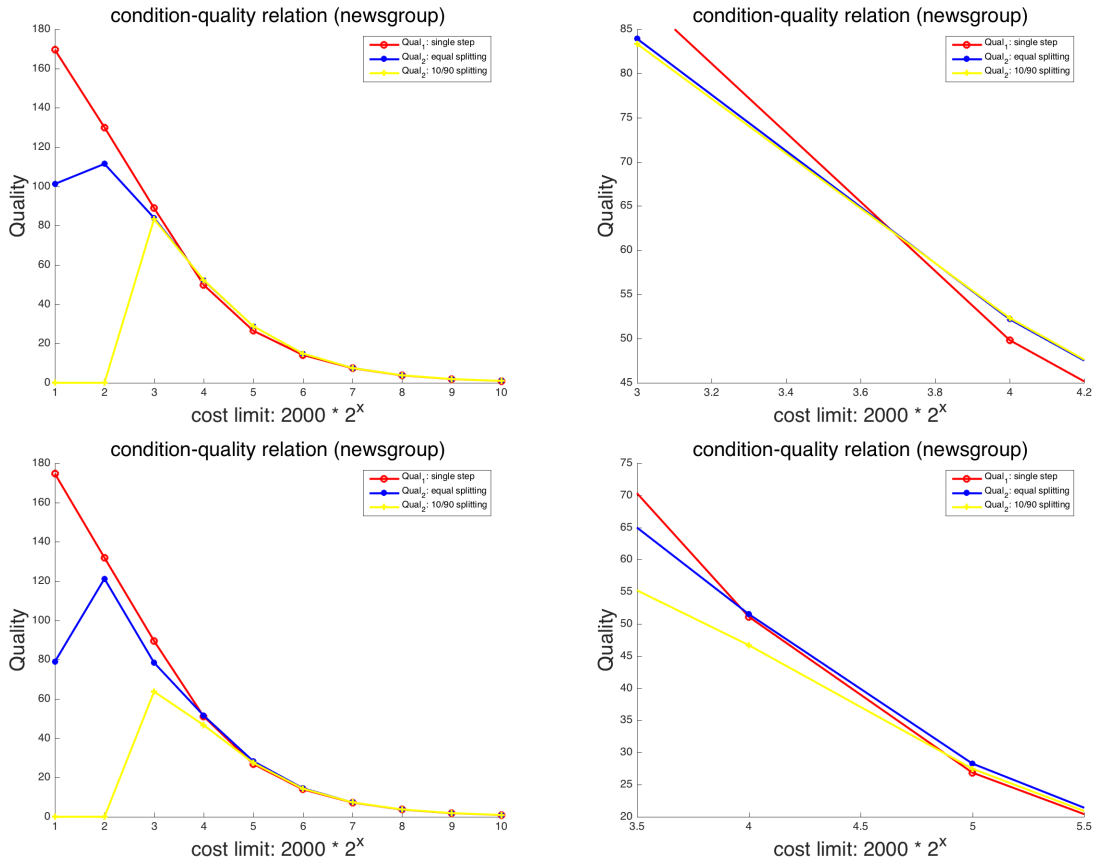


FIGURE 12: Single step, equal splitting and 10/90 splitting where sample size is 200 and 400

Multiple Steps Strategy

Similar to the definition of $Qual_1$ and $Qual_2$, we define $Qual_3$ as the quality for three-step:

$$Qual_3(S_1, c) = Qual_1(S_1, \frac{1}{3}c) + Qual_1(S_2', \frac{1}{3}c - k) + Qual_1(S_3, \frac{1}{3}c - k)$$

The reason why we apply multiple-step strategy is to select better query and obtain a more accurate prediction on the original database. When $Qual_1(S_1, c) > Qual_2(S_1, c)$, i.e. the quality of single step is better than two-step, even though we can receive a more accurate prediction on second step, the growth in quality cannot cover the cost which was spent in updating the sample. In other words, a large amount of resources were wasted in modifying the sample. Therefore, when we take three-step strategy into consideration, the available resources which could be distributed on each step is less than equal splitting strategy. In addition, the actual cost used in crawling work is only $\frac{1}{3}c - k$ on each step, which was even less than equal splitting strategy. Hence, normally if $Qual_1(S_1, c) > Qual_2(S_1, c)$, $Qual_1(S_1, c)$ would also be better than $Prod_3(S_1, c)$ as well. Analogously, if $Qual_1(S_1, c) < Qual_2(S_1, c)$, the improvement of quality can cover the impact of revising cost on second step. Thus it usually can cover the impact of revision cost in next steps. Hence, in this case, the quality of three-step strategy would in general also be better than single step. Further evidence of these arguments is given in Figure 13.

Example 8: *Figure 13 shows the condition-quality relation of one-step, two-step and three-step strategies which is under newsgroup data set, where* $\beta = 10^3$*,* $k = 10$*. Take all these three strategies into consideration, it suggests that* c *is in range* $[2^3, 2^4]$ *for all the three strategies when* $S_1 = 200$*, and when sample size is 400, the conclusion remains the same.*

We concentrate on only a part of splitting strategies in our studies. Other strategy may yield better result sometimes. For example, if 25/25/25/25 is the final split of total cost budget according to our equal splitting strategy, in some situations, it is possible that 30/30/40 strategy can generate a better result, but we do not check it

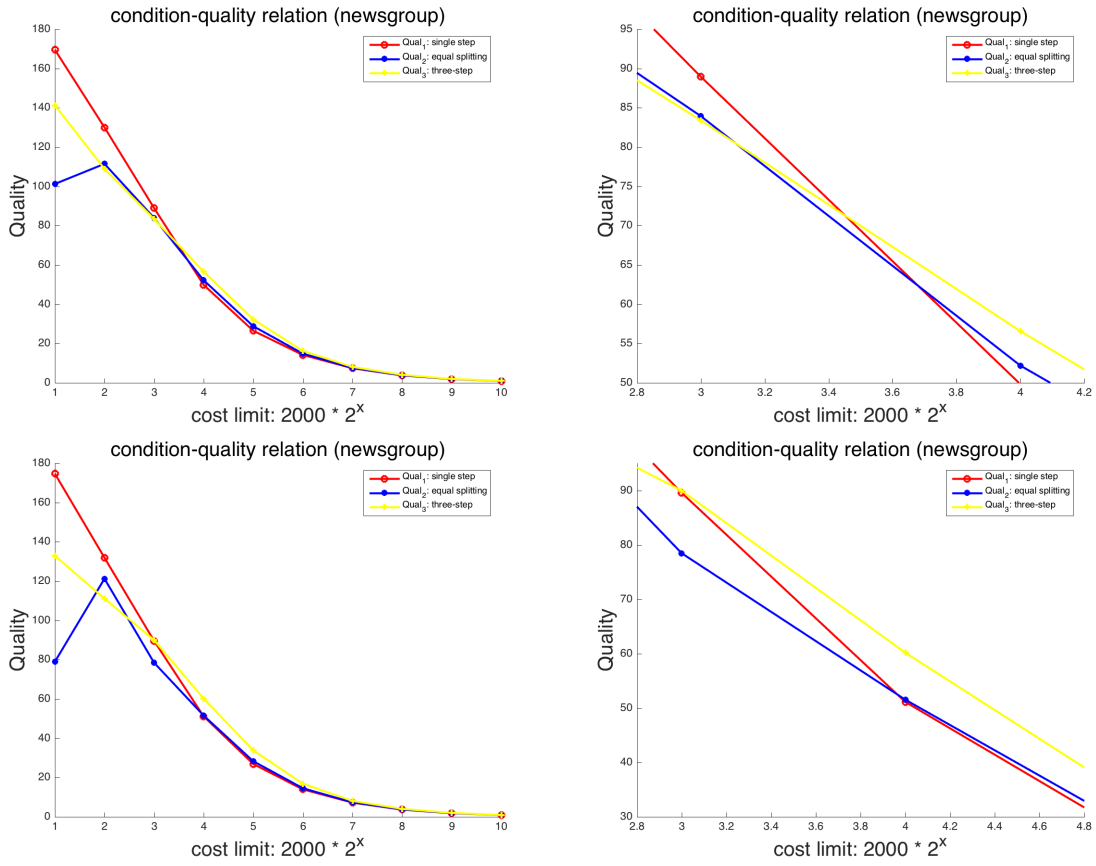


FIGURE 13: $Qual_1(S_1, c)$, $Qual_2(S_1, c)$ and $Qual_3(S_1, c)$ where sample size is 200 and 400

in our experiments. Therefore, we have to point out that some limitations of this study exists. It is hard for us to try all splitting strategies in our experiments due to the limitation of computational resources.

4 Summary

In this chapter, we define some parameters to measure the coverage and quality of different strategies. The purpose of our study is to provide a guideline for deep web crawler based on our sample-condition relation diagram. Our method is cost-effective and data-driven in general.

CHAPTER V

Experiments

In this chapter, we first explain in section 1, the datasets and the environment setting for the experiments. Then we present the sample-condition relation in different datasets. Moreover, we illustrate the impacts of some parameters on our results in section 2. Finally, we discuss some issues related to our experiment in section 3.

1 Experiment Environment and Datasets

We write our Java code about query selection based on Lucene [18] package. In order to show the consistency of our results, we carry out our experiments on four famous datasets in deep web crawling: newsgroup, Wiki, Gov2 and Reuters. Given the fixed value of unit sending cost β and revising cost k, we aim to provide a more cost-effective crawling strategy for users by the way of checking our sample-condition relation diagram. Our experiment is carried out on a web server with 24 processors. The original datasets are quite large compared to the computation capability we have. For example, it takes too much time to build a document-term matrix when we try to select queries by applying adaptive method. Thus we randomly select 10,000 documents from each of these four datasets as our original database.

2 Impact of Parameters

2.1 k-value

In our experiments, we use k to represent the revising cost for updating sample in adaptive crawling method. The reason is that the adaptive method have to take a large amount of CPU time and space to retrieve and download new documents from DB. After that, we also need to anylyze new sample and do query selection on it. Therefore, in order to examine how much k affects the sample-condition relation diagram, we set β to a fixed value 10^3, and provide the SCR diagram with four different k values in the following figure 14.

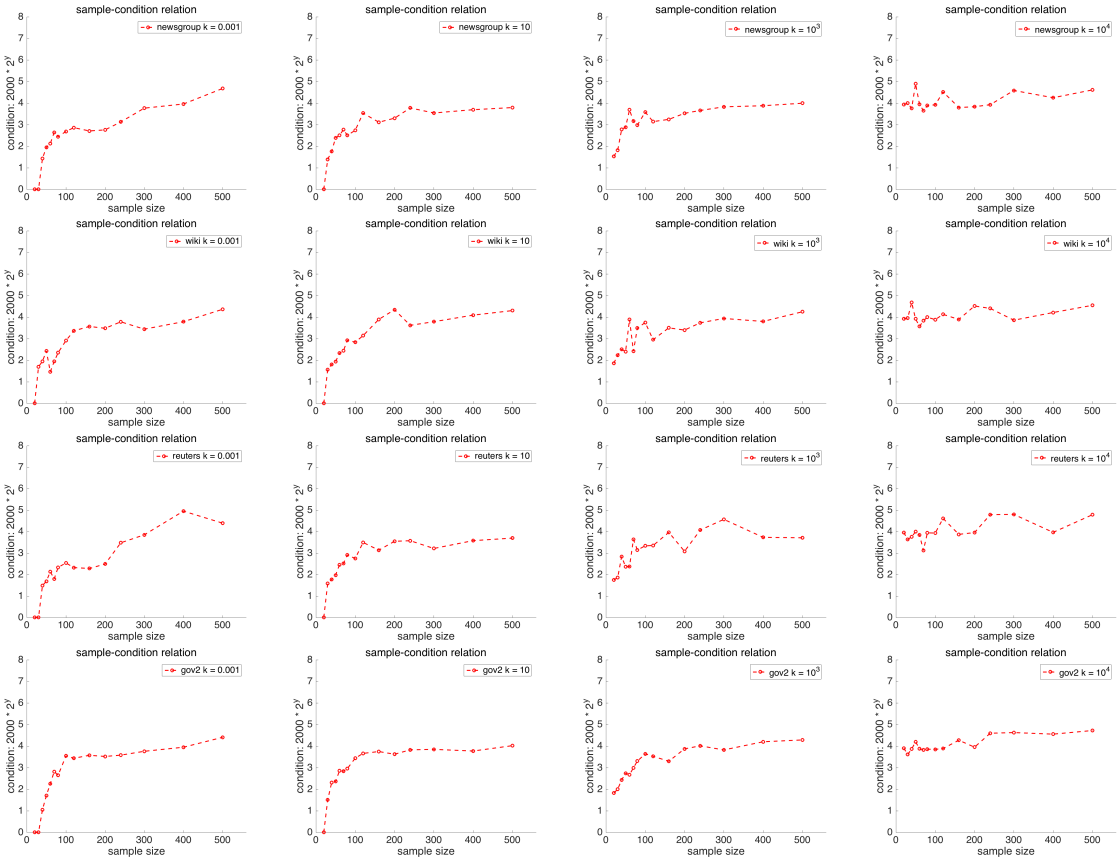


FIGURE 14: $k = 10^{-1}$, $k = 10$, $k = 10^3$, $k = 10^4$ in newsgroup, wiki, reuters and gov2. β is set to 10^3

The results from figure 14 show that the condition has a log-like growth with the

increase of sample size. In addition, when k is increasing, the base of this log function is increasing as well. That is to say, the cost limit is getting into a fixed value which increases slowly as sample size increases. This result is in support of our previous analysis. It demonstrates that when k increases its value, i.e. when it becomes more and more expensive to perform the sample update, the cost limit will become higher and higher, which turns more and more in favor of sample-based approach.

2.2 Unit Sending Cost

Recall that β refers to the ratio between unit sending and unit receiving cost in general. When the unit receiving cost and the unit revising cost are fixed, a higher value of β means the sending cost plays a more important role than revising cost,This is very similar to the situation when α and β are fixed and k decreases its value. We show our SCR diagram with three different β values below.

In the SCR diagrams in figure 15, when $\beta = 10^2$, for example, the relationship between condition and sample size is similar to the SCR diagram where $k = 10^4$ in figure 14.

A consolidated view on the impacts of k and β on sample-condition relation is given in figure 16 and figure 17.

3 Other Factors

In our work, we first randomly retrieve some documents from original DB as initial D. Then the QP is built by choosing some terms from this initial sample. After that, we select appropriate queries on each step. There are some other factors that affect our analysis on the step size of the increments in the querying process.

3.1 Initial Sample

About the method to generate initial sample D, we should be aware that we do not have global knowledge about the original datasets in practice. The only way

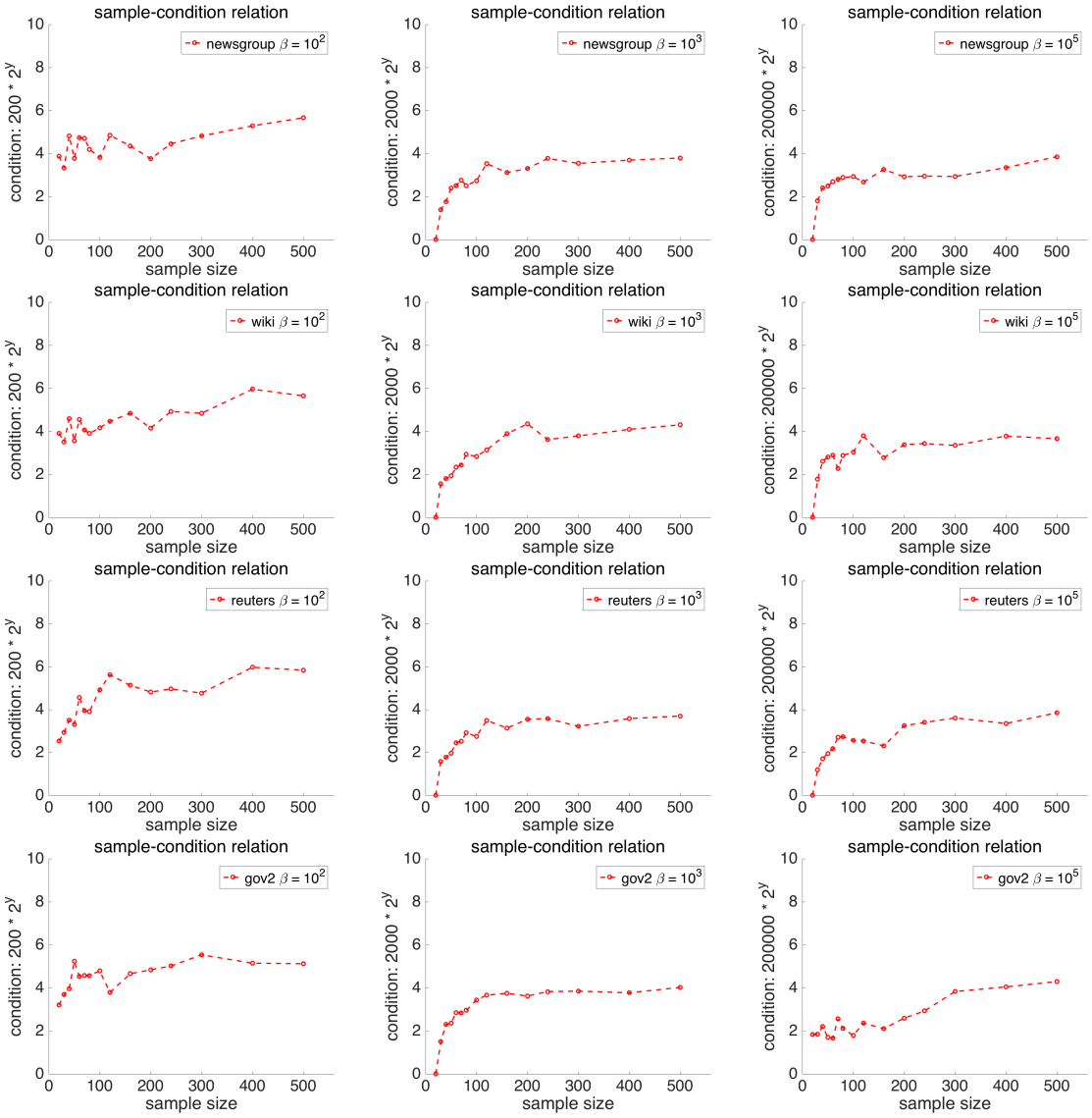


FIGURE 15: $\beta = 10^2$, $\beta = 10^3$, $\beta = 10^5$ in newsgroup, wiki, reuters gov2. k is set to 10

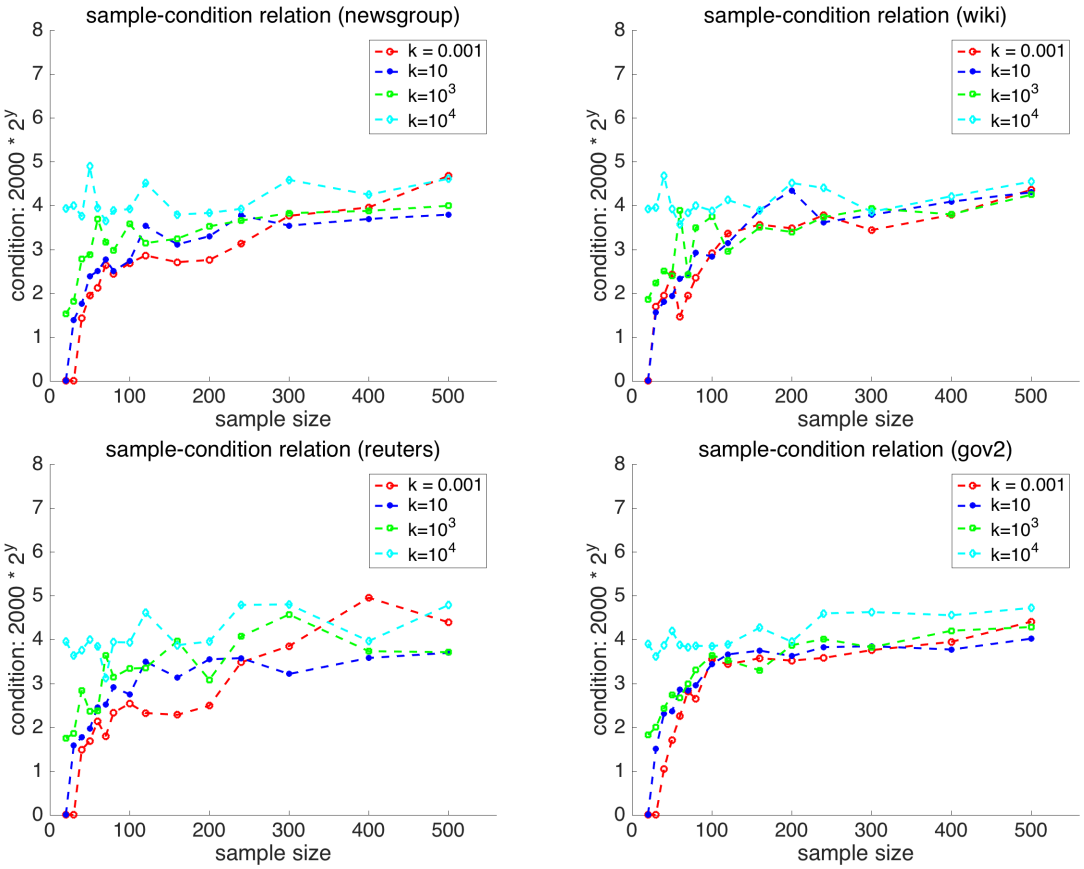


FIGURE 16: Sample-condition relation in different $k - value$, $\beta = 10^3$

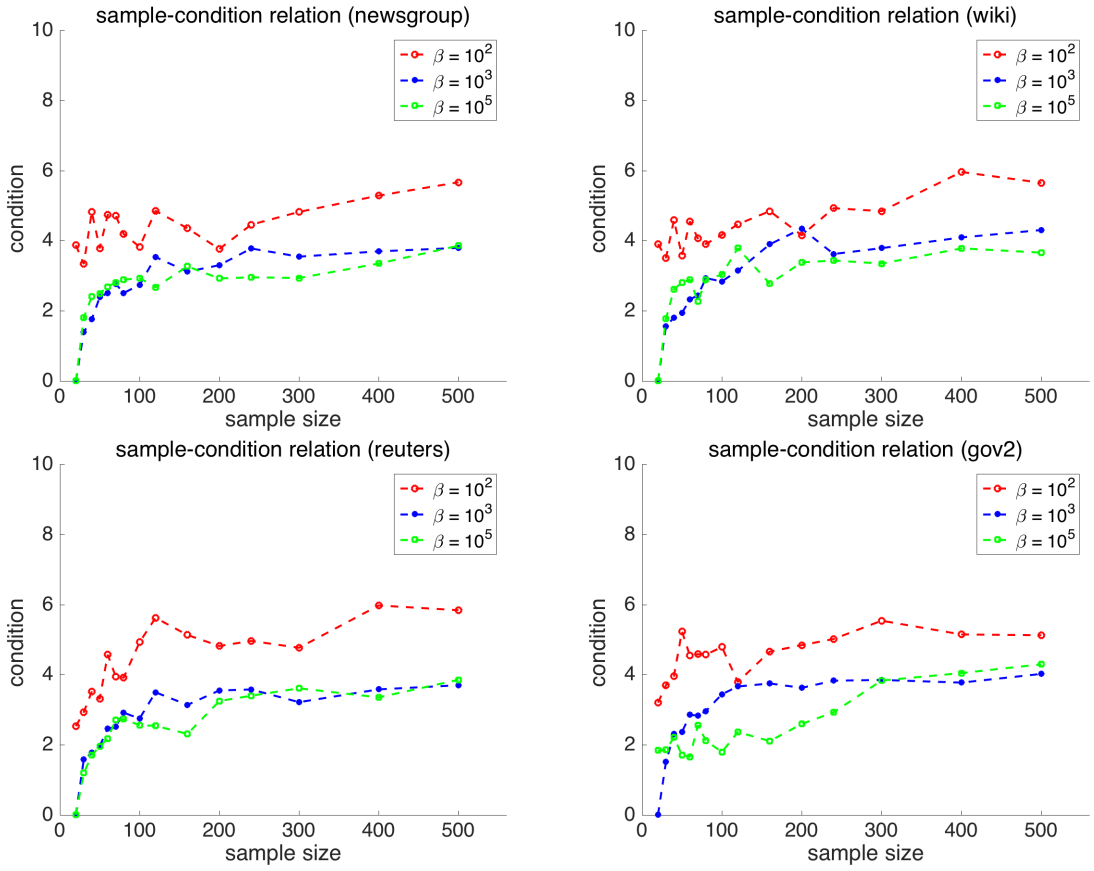


FIGURE 17: Sample-condition relation in different β value, $k = 10$

to retrieve documents is by issuing some queries to search interface. Usually, some queries are randomly selected to set up the initial sample. However, the random sampling of documents by sending random queries to a corpus is a challenging task [7][10]. In our context though the result between random queries and random sample has little difference [3]. Thus, in our experiments, we acquire our initial sample by randomly select some document IDs from our datasets.

Another issue regarding initial sample is the sample size. In our work, we run our program with 15 different values of sample size to generate a sample-condition relation diagram, and the maximum sample size in our experiments is 500. We do not include sample size larger than 500 because when sample size is larger than 300, the y-axis (condition) increases only very slightly with the growth of sample size. This can be seen clearly from figure 14

3.2 Create Query Pool

In order to do query selection, we have to build a corresponding query pool on each sample. In our experiments, similar to the other researches [30][38], we only select the terms whose document frequency (df) is in a range of 2-15% of the sample size as well. There are two reasons for this choice: If we include the terms with too low df into QP, it is probable that we will generate too many queries in query selection process. These queries will waste a large amount of resources in terms of sending cost. On the other side, if too high frequency terms are selected and issued to the original database, these queries will cause too many duplicates, which will seriously reduce the efficiency of deep web crawler in this scenario.

3.3 Query Selection Algorithm

In this subsection, we show the dependency of our method on the query selection strategy. The method we presented so far is based on the greedy strategy for query selection, with which the queries are ranked for selection according to new/df, where *new* denotes the number new documents this query can retrieve, and *df* is the docu-

ment frequency of the query. This ranking is consistent to the way quality of queries are defined in our method.

If our method is applied with a query selection strategy whose query ranking strategy is not consistent to what we use in our method, then the results might exhibit different characteristics. As an example, we re-examine our method with the *weighted greedy algorithm*. The ranking strategy in weighted greedy algorithm is particularly designed for single-step approach: it is in favor of *rare terms* especially at the beginning in order to gain better overall quality of the entire set of queries selected. When this is used in our method, those rare terms are selected to determine the cost limit, which may lead to lower cost limits.

Figure 18 shows the related SCR diagrams when k takes four increasing values. Compared with the corresponding SCR diagrams obtained with greedy algorithm, we can see that the curves do not have too much differences when k increases. Figure 19 shows the related SCR diagrams when β takes three increasing values. The results are, on the other hand, similar to those obtained with greedy algorithm.

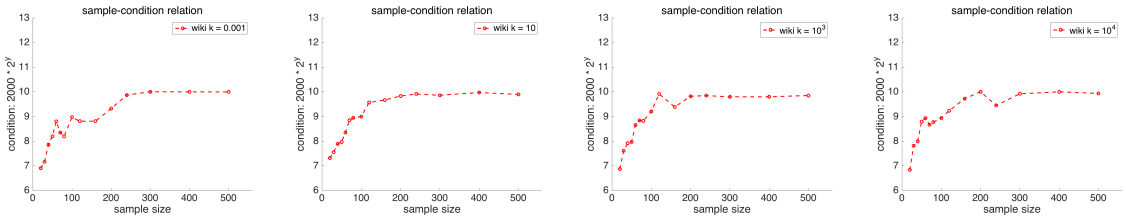


FIGURE 18: Weighted greedy algorithm when $k = 10^{-1}$, $k = 10$, $k = 10^3$, $k = 10^4$ in wiki, β is set to 10^3

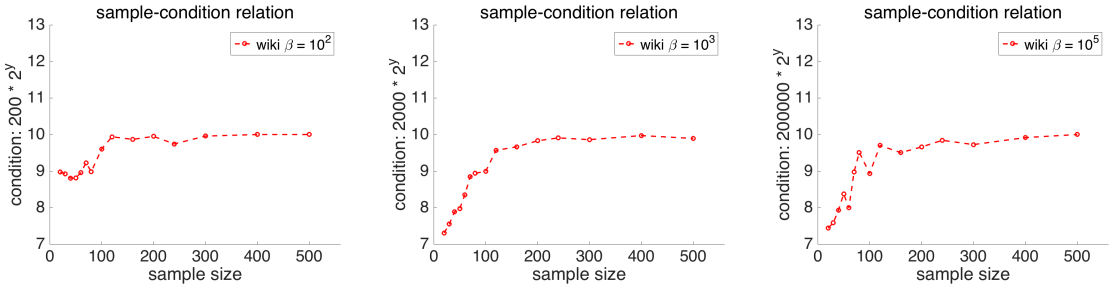


FIGURE 19: Weighted greedy algorithm when $\beta = 10^2$, $\beta = 10^3$, $\beta = 10^5$ in wiki, k is set to 10

CHAPTER VI

Conclusion

In our thesis, we focus on providing a guideline for users to choose a more cost-effective crawling strategy in crawling deep web. For this purpose, we introduce some parameters to measure the performance in different situations. In our method, for the cost-effectiveness, when the given cost budget is sufficient enough, we minimize the cost to reach desired retrieving result. On the other hand, when the given cost budget is limited, we maximize the retrieving result within this cost limit. Our practical guides for the stepwise strategy is learnt from the empirical study which was carried out on four existing data sets. We have also discussed the impacts of revising cost (k) and unit sending cost (β) on the sample condition relation by our experiment results.

REFERENCES

[1] Álvarez, M., Pan, A., Raposo, J., Bellas, F., and Cacheda, F. (2008). Extracting lists of data records from semi-structured web pages. *Data & Knowledge Engineering*, 64(2):491–509.

[2] Álvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F., and Carneiro, V. (2007). Crawling the content hidden behind web forms. In *International Conference on Computational Science and Its Applications*, pages 322–333. Springer.

[3] Bar-Yossef, Z. and Gurevich, M. (2008). Random sampling from a search engine's index. *Journal of the ACM (JACM)*, 55(5):24.

[4] Barbosa, L. and Freire, J. (2004). Siphoning hidden-web data through keyword-based interfaces. In *SBBD*, pages 309–321.

[5] Beasley, J. E. and Chu, P. C. (1996). A genetic algorithm for the set covering problem. *European Journal of Operational Research*, 94(2):392–404.

[6] Bergman, M. K. (2001). White paper: the deep web: surfacing hidden value. *Journal of electronic publishing*, 7(1).

[7] Bharat, K. and Henzinger, M. R. (1998). Improved algorithms for topic distillation in a hyperlinked environment. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, pages 104–111. ACM.

[8] Bhatia, K. K. and Sharma, A. (2008). A framework for domain-specific interface mapper (dsim). *IJCSNS*, 8(12):56.

[9] Bhatia, K. K., Sharma, A., and Madaan, R. (2010). Akshr: A novel framework for a domain-specific hidden web crawler. In *Parallel Distributed and Grid Computing (PDGC), 2010 1st International Conference on*, pages 307–312. IEEE.

[10] Callan, J. and Connell, M. (2001). Query-based sampling of text databases. *ACM Transactions on Information Systems (TOIS)*, 19(2):97–130.

[11] Caprara, A., Fischetti, M., and Toth, P. (1999). A heuristic method for the set covering problem. *Operations research*, 47(5):730–743.

[12] Caverlee, J., Liu, L., and Buttler, D. (2004). Probe, cluster, and discover: Focused extraction of qa-pagelets from the deep web. In *Data Engineering, 2004. Proceedings. 20th International Conference on*, pages 103–114. IEEE.

[13] Chang, C.-H., Kayed, M., Girgis, M. R., and Shaalan, K. F. (2006). A survey of web information extraction systems. *IEEE transactions on knowledge and data engineering*, 18(10):1411–1428.

[14] Cormen, T. H. (2009). *Introduction to algorithms*. MIT press.

[15] Doan, A., Domingos, P., and Halevy, A. Y. (2001). Reconciling schemas of disparate data sources: A machine-learning approach. In *ACM Sigmod Record*, volume 30, pages 509–520. ACM.

[16] Feo, T. A. and Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, 6(2):109–133.

[17] Furche, T., Gottlob, G., Grasso, G., Guo, X., Orsi, G., and Schallhart, C. (2013). The ontological key: automatically understanding and integrating forms to access the deep web. *The VLDB Journal*, 22(5):615–640.

[18] Hatcher, E. and Gospodnetic, O. (2004). Lucene in action.

[19] He, B., Patel, M., Zhang, Z., and Chang, K. C.-C. (2007). Accessing the deep web. *Communications of the ACM*, 50(5):94–101.

[20] Huang, Q., Li, Q., Li, H., and Yan, Z. (2012). An approach to incremental deep web crawling based on incremental harvest model. *Procedia Engineering*, 29:1081–1087.

[21] Jiang, L., Wu, Z., Feng, Q., Liu, J., and Zheng, Q. (2010). Efficient deep web crawling using reinforcement learning. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pages 428–439. Springer.

[22] Khare, R., An, Y., and Song, I.-Y. (2010). Understanding deep web search interfaces: a survey. *ACM SIGMOD Record*, 39(1):33–40.

[23] Knoblock, C. A., Lerman, K., Minton, S., and Muslea, I. (2003). Accurately and reliably extracting data from the web: A machine learning approach. In *Intelligent exploration of the web*, pages 275–287. Springer.

[24] Kushmerick, N. (1997). *Wrapper induction for information extraction*. PhD thesis, University of Washington.

[25] Liakos, P., Ntoulas, A., Labrinidis, A., and Delis, A. (2016). Focused crawling for the hidden web. *World Wide Web*, 19(4):605–631.

[26] Liddle, S. W., Embley, D. W., Scott, D. T., and Yau, S. H. (2002). Extracting data behind web forms. In *International Conference on Conceptual Modeling*, pages 402–413. Springer.

[27] Liu, J., Wu, Z., Jiang, L., Zheng, Q., and Liu, X. (2009). Crawling deep web content through query forms. In *WEBIST*, pages 634–642.

[28] Liu, W., Meng, X., and Meng, W. (2010). Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447–460.

[29] Lorena, L. A. N. and Lopes, F. B. (1994). A surrogate heuristic for set covering problems. *European Journal of Operational Research*, 79(1):138–150.

[30] Lu, J., Wang, Y., Liang, J., Chen, J., and Liu, J. (2008). An approach to deep web crawling by sampling. In *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT'08. IEEE/WIC/ACM International Conference on*, volume 1, pages 718–724. IEEE.

[31] Madhavan, J., Bernstein, P. A., Doan, A., and Halevy, A. (2005). Corpus-based schema matching. In *21st International Conference on Data Engineering (ICDE'05)*, pages 57–68. IEEE.

[32] Madhavan, J., Ko, D., Kot, Ł., Ganapathy, V., Rasmussen, A., and Halevy, A. (2008). Google's deep web crawl. *Proceedings of the VLDB Endowment*, 1(2):1241–1252.

[33] Nelson, M. L., Smith, J. A., and Del Campo, I. G. (2006). Efficient, automatic web resource harvesting. In *Proceedings of the 8th annual ACM international workshop on Web information and data management*, pages 43–50. ACM.

[34] Raghavan, S. and Garcia-Molina, H. (2000). Crawling the hidden web.

[35] Salton, G. and McGill, M. J. (1986). Introduction to modern information retrieval.

[36] Shestakov, D., Bhowmick, S. S., and Lim, E.-P. (2005). Deque: querying the deep web. *Data & Knowledge Engineering*, 52(3):273–311.

[37] Vieira, K., Barbosa, L., Freire, J., and Silva, A. (2008). Siphon++: a hidden-webcrawler for keyword-based interfaces. In *Proceedings of the 17th ACM conference on Information and knowledge management*, pages 1361–1362. ACM.

[38] Wang, Y., Lu, J., and Chen, J. (2009). Crawling deep web using a new set covering algorithm. In *International Conference on Advanced Data Mining and Applications*, pages 326–337. Springer.

[39] Wang, Y., Lu, J., and Chen, J. (2014). Ts-ids algorithm for query selection in the deep web crawling. In *Asia-Pacific Web Conference*, pages 189–200. Springer.

[40] Wang, Y., Zuo, W., Peng, T., and He, F. (2008). Domain-specific deep web sources discovery. In *2008 Fourth International Conference on Natural Computation*, volume 5, pages 202–206. IEEE.

[41] Wu, P., Wen, J.-R., Liu, H., and Ma, W.-Y. (2006). Query selection techniques for efficient crawling of structured web sources. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 47–47. IEEE.

[42] Zerfos, P., Cho, J., and Ntoulas, A. (2005). Downloading textual hidden web content through keyword queries. In *Proceedings of the 5th ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL'05)*, pages 100–109. IEEE.

VITA AUCTORIS

NAME:	Xu Sun
PLACE OF BIRTH:	Taian, Shandong province, China
YEAR OF BIRTH:	1990
EDUCATION:	Shandong University, Software Engineering, Jinan, China, 2011
	University of Windsor, M.Sc in Computer Science, Windsor, Ontario, 2016