4-1-2007

Downloading data from textual deep Web using clustering.

Xiaolei Yuan
University of Windsor

Downloading Data from Textual Deep Web Using Clustering

by

Xiaolei Yuan

A Thesis
Submitted to the Faculty of Graduate Studies
Through Computer Science
In Partial Fulfillment of the Requirements for
The Degree of Master of Science at the
University of Windsor

Windsor, Ontario, Canada

April, 2007

Your file Votre référence
ISBN: 978-0-494-34985-4
Our file Notre référence
ISBN: 978-0-494-34985-4

ABSTRACT

Deep web is the web that is dynamically generated from data sources such as databases or file systems. Crawling deep web is the process of collecting hidden data by issuing appropriate queries in order to download most of the data. Our main challenge is to select appropriate queries in order to obtain most of the data from a data source. A naive solution, which selects the queries that return most results, is problematic because 1) the results may not cover the data source, and more importantly, 2) the results suffer from high overlap, which makes the acquisition of new data items almost impossible after certain steps. The thesis experiments with four different algorithms to select the queries that minimize the overlap rate: 1) greedy algorithm based on set packing; 2) cluster-based algorithm to remove the queries that result in similar returns.

Keywords: deep web, hidden web, data discovery, data mining, clustering, information retrieval

DEDICATION

To my parents and all those who helped me through this journey

ACKNOWLEDGEMENTS

First of all, I really appreciate the great help of my supervisor, Professor Jianguo Lu, during my two and half years' master study in University of Windsor. Without his constant encouragement and valuable guidance, I could not make such success in my research field.

Secondly, I would also like to acknowledge my thesis committee members, Dr. Yuntong Wang, Dr. Christie Ezeife, and Dr. Dan Wu for their unwavering help.

Thirdly, I want to acknowledge my friends. Their wisdom guidance helped me solve many difficult problems during my research.

Finally, I am very grateful for my parents. During these two and half years' study, they gave me great confidence and solid support whenever I met any kind of problems. They always stand by my side and back me up no matter what happens. This is the most valuable asset in my life. I sincerely wish them have great happiness in their lives!

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Hidden textual data are accessible through HTML [19] forms or web services [36] interfaces. Web pages in the hidden web are dynamically generated according to users' requests. Bergman estimated in the "Hidden Web White Paper" [5] that the total size of the hidden web is 7,500 Terabytes compared with 19 Terabytes of data in the surface web, and that the total number of hidden web documents is estimated to be 550 billion compared with only 1 billion of that in the surface web. As shown in Figure 1 [5], the size of the deep web greatly exceeds that of the surface web. Furthermore, organizations such as the Census Bureau, Patents and Trademarks Office, and News media companies, whose information are of high-quality, are placing part of or all their public database online in the deep web [31], meaning their data are hidden behind search forms in large searchable databases [15]. This makes the extraction of deep web data necessary.

Crawling the hidden web [3][7][10][22][29][31][33][38] is the process of collecting hidden data by issuing queries through various search interfaces including HTML forms and web services [14].Our goal is to efficiently extract textual data from different data sources by selecting meaningful queries.

We have developed a deep-web crawler that is able to extract deep web data. Compared with previous approaches, our system is built with the aim to optimize the extraction efficiency (i.e. high hit rate and low overlap rate), especially when applied on data sources with limits on the number of results per query.

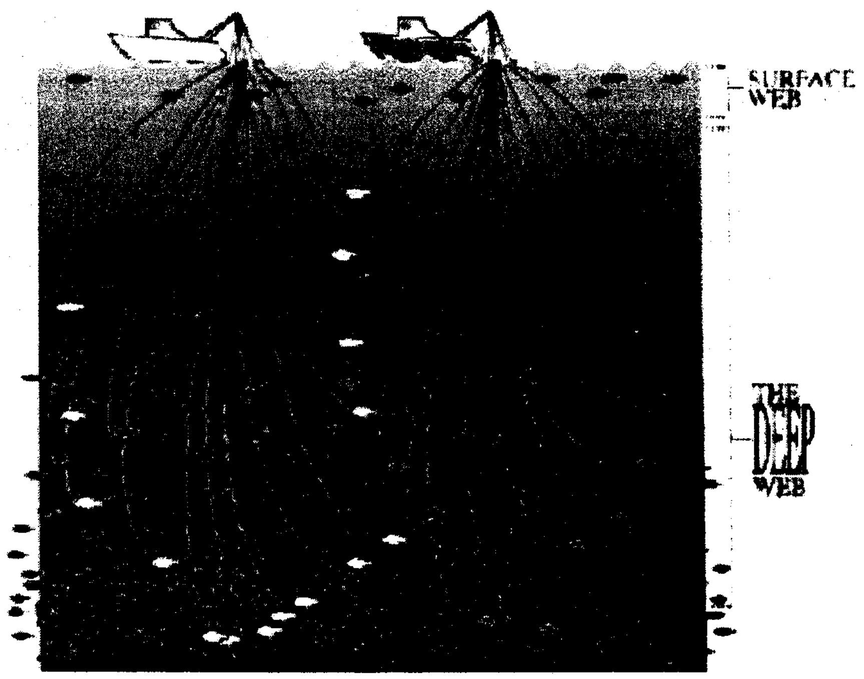


Figure 1: The surface web and the deep web [5].

1.1. Motivation

Nowadays, it is easy to search for anything web users are interested in by typing in one or a combination of keywords to query one or more data sources such as web search engines. However, studies show that a lot of web documents are "hidden" underneath different search forms. Users can only obtain data by issuing queries or keywords to search forms. Due to this ever-increasing size of the hidden web, most search engines are unable to index the entire collection of documents [5]. As a result, those documents are not displayed as search results to the user.

This thesis and its related researches and experiments were motivated while we attempt to collect WSDL (Web Service Definition Language) documents on the internet, so that the characteristics of the WSDL documents can be analyzed [42]. However the total number of unique results cannot go high during the collection stage because of the high overlaps of the results obtained from different search engines. This prompted us to choose appropriate keywords to be issued to the search engine in order to extract data efficiently.

Search engines themselves are considered to be large data sources. Users can input search keywords or queries to acquire web documents. However, most search engines, such as Google and Yahoo, limit the number of retrievable results to a certain number (Table 1), compared with millions of estimated results displayed on the search result page.

Search Engine	**Result limit per query**
AltaVista.com	1000
Amazon.com	4800
Ask.com	2000
Google.com	1000
Yahoo.com	1000

Table 1: Search limitation of popular web search engines.

When we search for a certain term *t* which results in a large number of matches, and when it is desirable to get all the data back instead of just the first *k* matches decided by the search engine, there is a need to develop a method to do so. This problem is not only applicable to various search engines, but also to most web forms where query forms are provided and there are hidden data that we wish to look at.

One naive solution to the problem is to query the data source with a dictionary that contains a large amount of keywords. Intuitively speaking, if the dictionary is large enough in terms of the number of words, and is broad enough in terms of the vocabulary coverage, then any hidden data source would be easily extracted and thus conquered by our hypothetical all-purpose dictionary.

However, this naive approach is not feasible because for the following reasons. First, speed is the concern. As we know, most data sources reside on the internet, and the network resource is limited. Constantly querying a data source by issuing many queries to the data source does not only require a lot of network resource but can also become a burden to the data source and thus preventing other users to visiting the data source. Second, a lot of keywords may result in zero returns, because the data source does not contain any documents related to that certain keyword. Third, as we will see in later chapters, if random queries are issued to a data source, there are many results that overlap with the results that over queries produce, meaning different result sets have common elements. We will also see that by carefully choosing the vocabulary that we use to query the search engine, we can solve the above three problems, and our data extraction process can become more efficient.

1.2. Related Work

Building a deep-web crawler has two main challenges. First, the crawler has to understand the web form. Raghavan et al. have proposed a model which learns the query forms [31]. Second, after understanding the query forms, the crawler has to be able to automatically query the form in order to get data back. Ntoulas et al. [29] have proposed an adaptive method that is able to collect and discover hidden-web data. The method chooses query to issue by calculating the efficiency of query candidates. The efficiency is obtained by dividing the number of new documents the query can return by the cost of issuing the query. This is a "greedy approach and tries to maximize the 'potential gain' in every step"[29]. However, their focus is on data sources that do not have a limitation on the number of returns per query. A more detailed comparison between the adaptive method and the clustering method is provided in section 6.4.3.

1.3. Contribution

We have set up a framework for the purpose of experimenting different methods of deep web textual extraction on different corpus. We have experimented four types of word selection methods with different variations. Those four types of methods are: random method, cluster method, set packing method, and set covering method (adaptive method proposed in [29]). We compare their effectiveness by measuring the overlap rate (*OR*) and the hit rate (*HR*) obtained by searching the data source using dictionaries generated from the four methods mentioned above. We also conducted experiments to compare how different data search engines' sorting policies and result limitations affect the performance of the word selection methods.

CHAPTER II

PROBLEM FORMALIZATION

In this section, we formally describe the problem that we are going to present solutions for in this thesis.

2.1. Textual Data Source

A number of different types of data sources with hidden web data exist on the internet. They can be categorized into a *textual database* or a *structured database* [29]. A textual database refers to a data source where the data are text documents that do not have a well-defined structure. When querying such a data source, the user can only input one or a combination of keywords. For example, the user can issue a query to the newsgroup data source we used in our experiments. The query contains one or more keywords, e.g., "*OpenGL simulation*". The feedback the user gets is the document(s) that contain these keywords. On the other hand, in a structured data source, user can specify attributes as well as keywords that are used to query the database, and the returning results are structured or semi-structured data, such as relational database tables or XML documents. We can consider a textual database as a special case of the structured database where there is only one attribute available for querying. Again using the previous example, user is able to query the data source in such way: "*I'd like to find documents which contain the* *word 'OpenGL' in the contents, and these documents should be in a category that contains the word 'simulation*'". Here in our paper, we only focus on extracting data from textual data sources.

Regardless the different types of data sources, the basic set of interactions between the user and the data source is the same. They can be described in three steps:

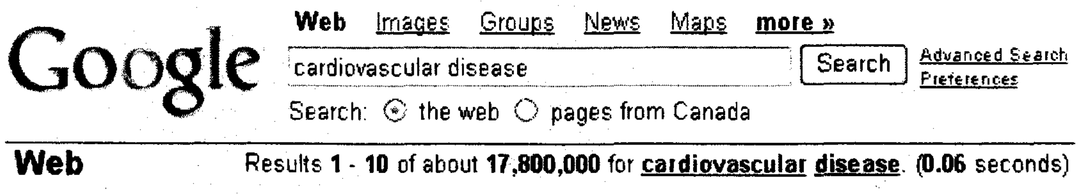
Query: the user thinks of one or a combination of keywords and issues that keyword to the data source engine.

Response: the data source engine returns a list of results corresponding to the keyword just entered. The results may be divided into several pages, and the user may only be allowed to look at the first few. Also, the results may be sorted by the data source in certain ways.

Result selection: the user chooses the results of his interest. User may "flip through" different pages of results by contacting the data source engine again.

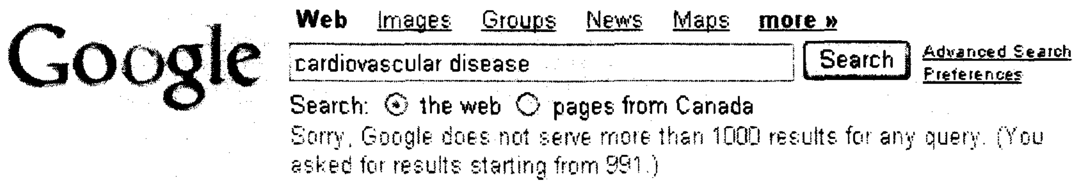
2.2. Problem Description

Our problem can be described as follows: first, we assume that there is a target space of data that the user is interested in. This space of data can be defined as the total results expected to be returned by the data source by issuing a certain query. For example, if the user is interested in finding information regarding "*cardiovascular disease*" on Google.ca, a query "cardiovascular disease" can be issued. The number of expected (estimated) results by issuing this query is 17,800,000, as shown in Figure 2a [17]. The user is able to request the first one-thousand results. However, the rest of the data cannot be retrieved because of the limitation imposed by Google. If we request results that have indexes larger than 1000, Google refuses to return any results. For example, after issuing a query, we have requested Google to return results starting from number 991. Without the imposed limitation, this query should cause the search engine to return results ranked from 991 to 1001. In reality, we get a message shown in Figure 2b [17], stating that Google "*does not serve more than 1000 results per query*" [17]. Similar scenarios can happen on other search engines or data sources such as Yahoo.com and Ask.com. Our goal is to extract as much resources as possible using partial or a fractional portion of information (e.g., first one thousand results) to efficiently discover the entire space.


Google
Web Images Groups News Maps more »
cardiovascular disease
Search
Advanced Search
Preferences
Search: the web pages from Canada
Web

Results **1** - **10** of about **17,800,000** for **cardiovascular disease**. (**0.06** seconds)

Figure 2a: Estimated results from Google for "cardiovascular disease" [17].


Google
Web Images Groups News Maps more »
cardiovascular disease
Search
Advanced Search
Preferences
Search: the web pages from Canada

Sorry, Google does not serve more than 1000 results for any query. (You asked for results starting from 991.)

Figure 2b: Google refuses to provide more than 1000 results pre query [17].

2.3. Problem Formalization

Our problem is formally described as follows: we assume that given a certain query q, there is a set of results $S(q)$ whose cardinality (i.e. the total number of results) is known.

The data source has a way of sorting its results. A limitation of search results k is imposed, and that only the first $k \leq |S|$ results can be obtained. Our goal is to harvest the rest of the results by issuing additional queries while minimizing the overlaps among result sets thus reducing the cost.

As shown in Figure 3 (Adopted from [29]), each circle corresponds to the set of results for a certain query denoted as $S(q_i)$. Each dot on the graph represents an item in the result. For example, if we issue query q_1 to the data source, we are expecting 5 returns. If we issue query q_2 to the data source, we will receive 4 items in the result set. $S(q_1)$ and $S(q_2)$ have one common item. We say that q_1 and q_2 overlap with each other. This thesis focuses on reducing the overlaps between result sets by choosing optimal queries.

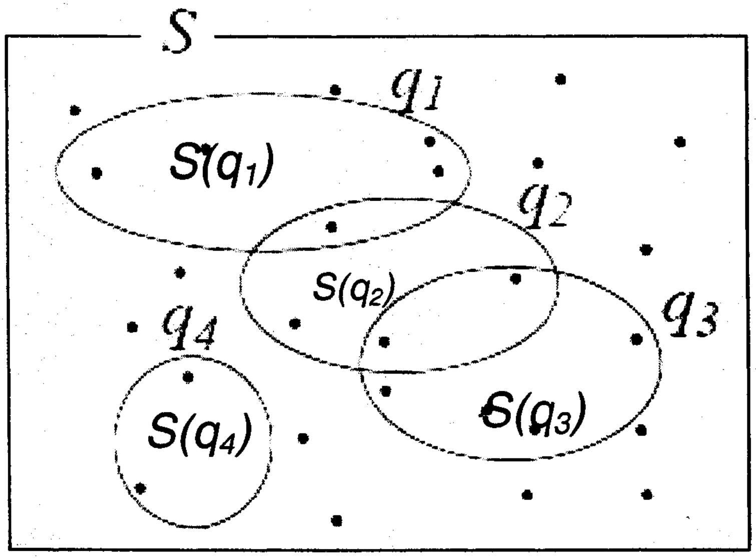


Figure 3: A set-formalization of the optimal query selection problem [29].

2.4. Evaluation Criteria

The goal of the extraction process is to harvest as much unique results as possible with less cost. In this section, we describe how we measure the amount of results collected and the cost respectively.

2.4.1. Hit Rate

Definition 1: (Hit Rate, HR) Given queries $Q = \{q_1, q_2, \ldots, q_i\}$. Let $S(q_j)$ denote the set of results of the query q_j. *n* is the number of data items in the data source. Hit rate is defined as ratio between the number of unique data items collected and the size of data set, i.e., hit rate at the *i*-th step is:

$$HR(i) = HR(\{q_1, \ldots, q_i\}) = \frac{|\bigcup_{j=1}^{i} S(q_j)|}{n}$$

The numerator of the right hand side of this equation denotes the number of accumulated unique results obtained up until the *i*-th step, which means the moment we obtain all results after issuing the *i*-th query to the data source. *n* is the size of the target dataset. In reality, this number is usually provided by the data source when we issue the "base query" to the database. In sum, Hit Rate (*HR*) is used to evaluate the *quantity* of discovery, i.e., how many unique results, out of all the expected results have been collected.

2.4.2. Overlap Rate

Definition 2: (Overlap Rate, OR) Given queries $Q = \{q_1, q_2, \dots, q_i\}$, the overlap rate of the search up to the *i*-th step is defined as the ratio between the total number of collected data and the number of unique data retrieved. i.e.,

$$OR(i) = OR(\{q_1, \dots, q_i\}) = \frac{\sum_{j=1}^{i} |S(q_j)|}{|\bigcup_{j=1}^{i} S(q_j)|}$$

The denominator of the right hand side of the equation represents the number of accumulated unique results obtained up to the *i*-th query, and the numerator represents the accumulated number of total results, including duplicates, up to the *i*-th query. This value measures the *quality* of extraction. The lower the *OR* value, the higher the quality of the extraction. Together with the number of queries issued (denoted as *Q*), they are considered the cost of the discovery.

In sum, we want to develop a method that is able to minimize the cost i.e., overlapping rate and the number of queries issued (*OR* and *Q*), while maximizing the fraction of documents obtained (*HR*).

CHAPTER III

THE DEEP WEB EXTRACTION PROCESS

In this chapter, we look at the deep web extraction process in detail. Each stage of the extraction process will be introduced.

3.1. Overview

The most important thing for a crawler to carry out efficient extraction of hidden-web data is to choose the right queries. As described earlier, a single extraction process can be divided into three steps. After these three steps are completed for the first time, we obtain the first batch of results, and we call it a *snapshot*. By analyzing the first batch of the result, we can gain an insight of the rest of the documents that are in the target space. In other words, if we obtain a snapshot of a picture, we can project the entire picture out by analyzing the snapshot (Figure 4).

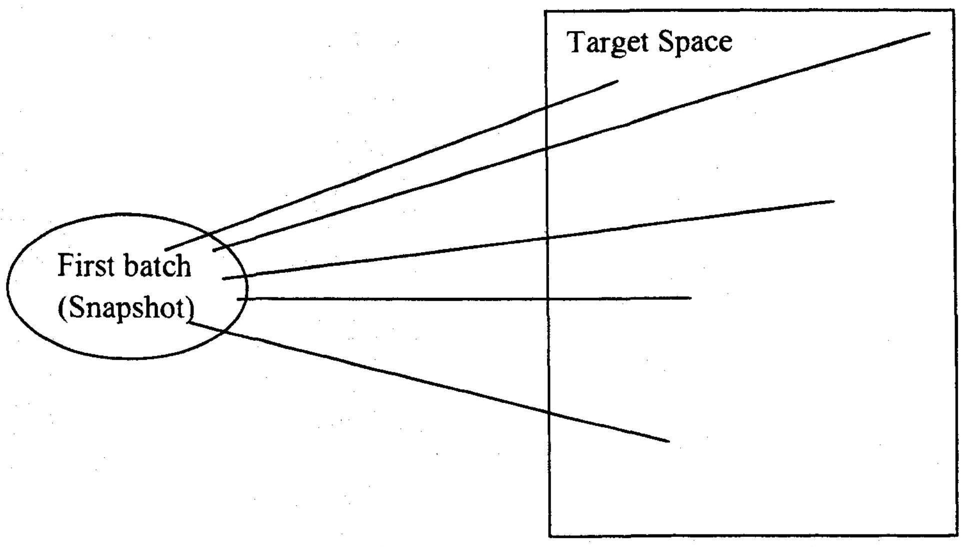


Figure 4: A snapshot of the entire space.

For example, the first 1000 results returned by Google when querying about "*cardiovascular disease*" are considered the snapshot of the entire 17,800,000 documents. These 1000 results are downloaded and analyzed locally to generate a dictionary we use to discover the rest of the big picture.

Information retrieval analysis is carried out on the first batch of search results. The process is shown in Figure 5. The documents in the snapshot results are first parsed to obtain a vocabulary set, which is a set of unique words that appear in the documents of snapshot results. Obvious stop words are excluded. We call these words the *Snapshot Vocabulary*. The size of the Snapshot Vocabulary can be very large, thus inefficient for further information retrieval analysis. Therefore, we need to downsize it to a desirable size. We call this process *Word Sampling*. After the performing the Word Sampling process, we get a *Sampled Vocabulary*. Finally, from the Sampled Vocabulary, a set of words are then chosen as keywords that will be issued to the search engine. The last part of this process is called *Word Selection*. This is also the core part of the extraction process. Techniques such as clustering and set packing methods are used to optimize the Word Selection.

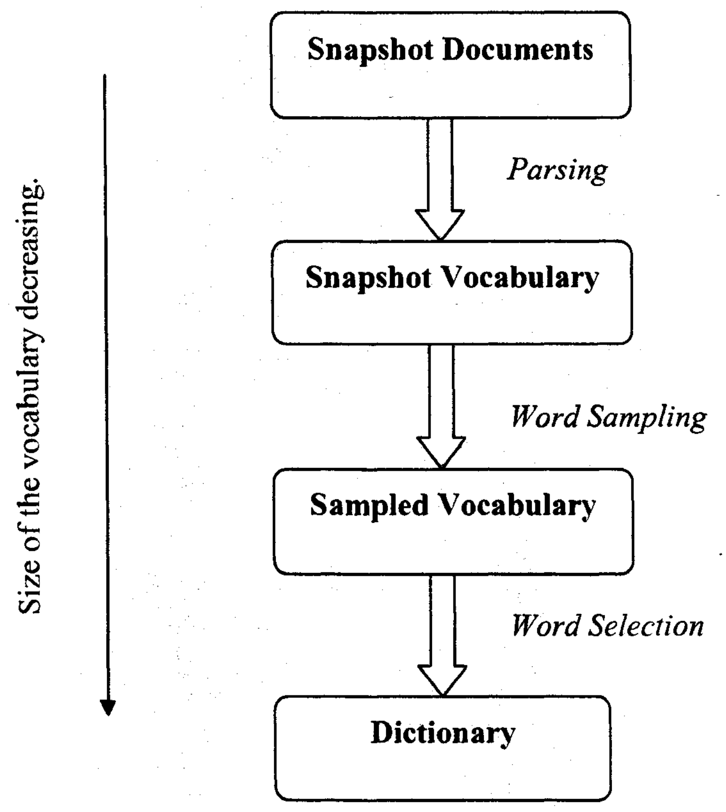


Figure 5: Refining the vocabulary to obtain dictionary.

Using the dictionary generated from the Word Selection stage, queries are issued to data source engines. Depending on the type of data source and restrictions that are applied on them, some data sources may only return the first *k* results of the entire result set *S(q)* for query *q*. Also, the first *k* results depend on the type of sorting policy that is either chosen by the user or the default option used by the data source. This process is shown in Figure 6.

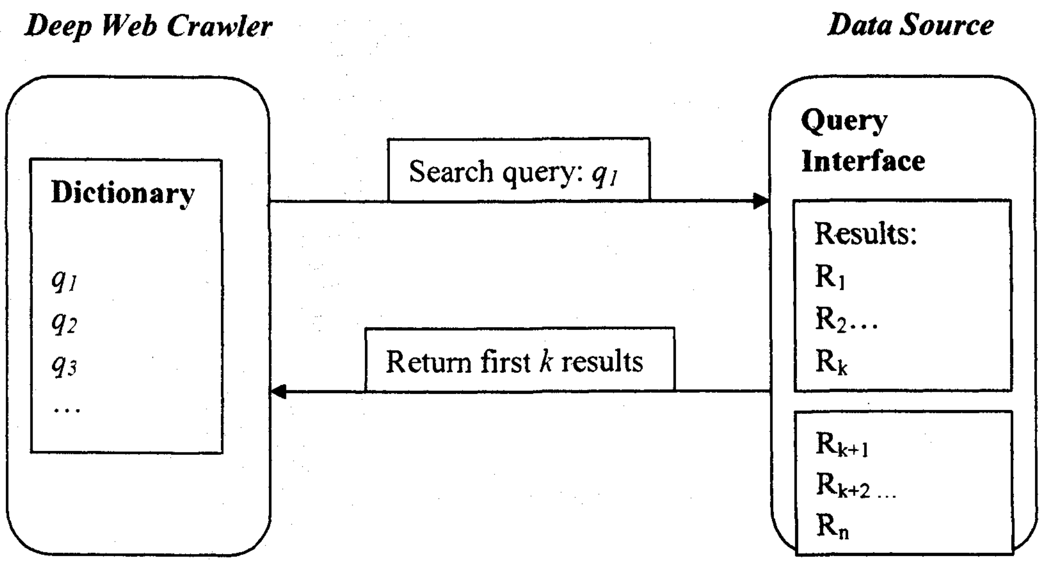


Figure 6: Interaction between crawler and data source.

We will discuss the entire extraction process in detail for the rest of this chapter.

3.2. Word Sampling

To select the queries to issue, we need to obtain a dictionary first. The dictionary is built from the first batch of search results.

Not every word in the first batch of search results should be included in the dictionary. For instance, a random first-batch result of newsgroup data source contains more than 36,000 unique words. Because of the limitation of the computer, we are only able to carry out the clustering process for a limited size of vocabulary. Moreover, it is unrealistic to assume that one can easily issue 36,000 queries to a search engine due to the limitation of network resources. Therefore, the first step is to choose a subset of terms that we are going to use. This is the first stage of query selection, and we have three options:

Random Words: The random words subset ensures the randomness. The subset we choose should have same or similar word frequency distribution as the original data. However, the drawback for this option is that not all words that we choose will guarantee us a good number of search results. According to *Zipf's Law* [16][43], the distribution of words sorted by their frequency (i.e., number of occurrences) is "very skewed" [2]. Therefore a set of stop words are taken out. However during our experimentations, we found that even after we removed the stop words, the distribution of words is still not even (Figure 7). We can see that about 75% of the words have very low frequencies. Therefore, by randomly polling words from the vocabulary, we still get a lot of queries with small number of returns, which is the reason why we need to consider the next option.

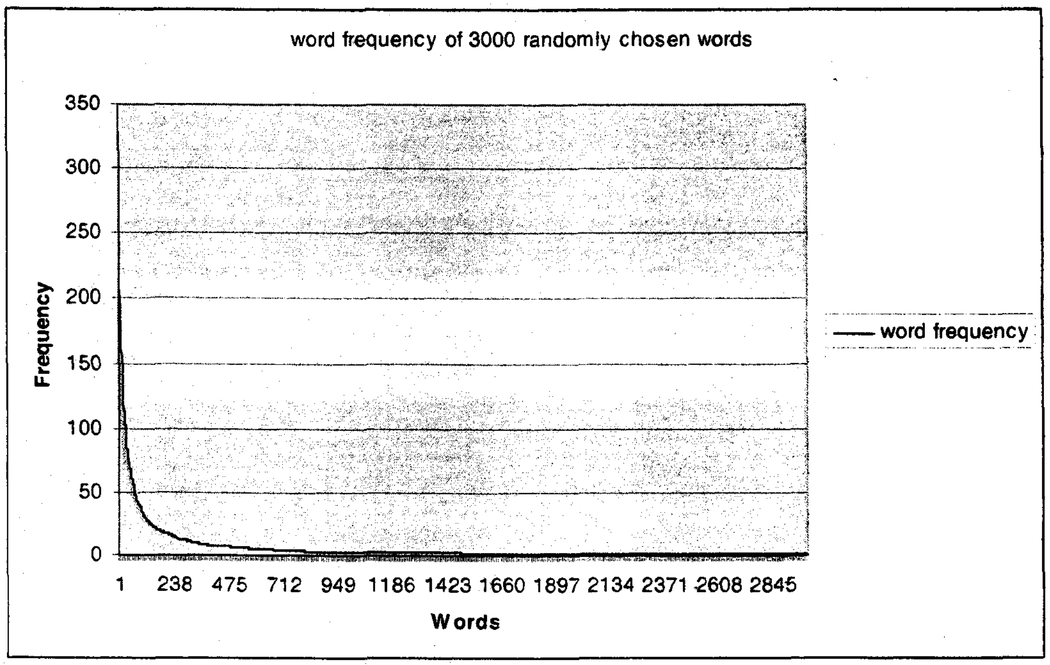


Figure 7: Word distribution sorted by term frequency.

Popular Words: This method of word selection guarantees that each word we choose will return a large number of results when the word is used as a query due to its popularity in the sampled data. The disadvantage of this option however, is that the overlapping rate will be inevitably high because of the popularity of words. Compared with the random selection method, the probability of two popular words having large overlaps is much higher than that of random words.

Optimal Words: As we can see from Figure 7, many words have either frequencies that are too high or frequencies that are too low. Those words are not suitable to be used as keywords in our extraction because of the following two reasons. First, high frequency words are likely to cause overlaps with other words, thus increase the *OR*. Second, even though words that have very low frequency are good for *OR*, they are not good for *HR*.

Low hit rate will not benefit our extraction process either. Therefore, we need to optimally sample words that are somewhere in the "middle" of the frequency table.

3.3. Word Selection

After a set of vocabulary with manageable size is obtained, we can start to further refine the vocabulary to produce the dictionary that will be used to query data sources. This stage is the core of the entire data extraction process, and can be done using different methods and techniques. Naive ways of doing word selection is to randomly choose *n* words from the vocabulary. Another simple method is called the *generic frequency* method [29] (most popular words method), in which most popular words are selected. As we will see in later chapters, this method ensures the *HR* of the extraction process, but high *OR* is also a drawback that comes with this method. In following chapters, we will describe how to utilize cluster method, set packing method random method and set covering method to carry out the word selection process.

3.4. Sorting Results

Different data source presents search results in different ways. For example, Google displays and sorts search results according to a page rank system. Other search engines are capable of sorting results by relevance, time modified, etc. Considering the fact the there is a limited number of results we can get per query, the sorting policies that a data source employs does play a role during the deep-web extraction process. There are three different types of results sorting policies simulated in our experiments. They are:

Random Order (No Sort, Unsorted, Unranked): Results are not sorted. Result items can appear in any random order.

Sort-by-Relevance: Results are sorted by their relevance to the query issued. The relevancy can be determined by the TF/IDF score [2] or other term-weighing methods.

Sort-by-Rank: Results are sorted according to a ranking system such as the page-rank system that Google uses. Each document is assigned a rank at time of indexing. The search results are sorted by their ranks. The higher the rank, the higher the result item positions in the result list, and thus more likely to be returned to the user.

CHAPTER IV

DATA EXTRACTION USING CLUSTERING

After downsizing the vocabulary to a manageable size during the word sampling stage, we can use cluster method to generate a dictionary to be used for extraction.

4.1. Clustering Overview

The method we are using is the bottom-up Hierarchical Agglomerative Clustering (H.A.C.), which is widely used in document clustering and information retrieval applications [32][37]. At the beginning of the process, each word in the dictionary is considered a distinct cluster. We then merge, at each step, two closest clusters together, until there are only a desired number of clusters left. When the clusters are obtained, the words in each cluster with the highest document frequency are then chosen as words in the dictionary. This is used because we want to ensure the Hit Rate.

In order to carry out the clustering process outlined above, there are several key elements we need to take into account. Different variations on these elements can have an impact on the performance of our method.

4.2. Distance Functions

In order to carry out the agglomerative clustering process, an N by N *proximity matrix* is computed to store distances between each pair of candidate terms, where N equals to the number of candidate terms.

Word distance plays a critical role in clustering method. There are different types of distance functions used to compute the proximity matrix. We experiment different distance functions and examine how they affect the clustering outcome. Many distance functions have been proposed over the years in the field of clustering. Here, we are considering three of them: Euclidean distance, Cosine angle distance, and Set-intersection distance.

Assuming that we have n documents, and let D_i denote the i-th document. Each term t_u is represented as a vector $\bar{t}_u = (t_{u,1}, t_{u,2}, \ldots, t_{u,n})$, where each element $t_{u,i}$ in $\bar{t}_u$ corresponds to the number of appearances of the term t_u in document *Di*. The size of the vector $\bar{t}_u$ equals to n, the number of total documents.

4.2.1. Euclidean distance

If we consider both terms to be vectors in an n-dimensional space, then the Euclidean distance is the distance between the two points.

Definition: Let $\vec{t}_u = (t_{u,1}, t_{u,2}, \ldots, t_{u,n})$ and $\vec{t}_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,n})$ be two vectors of document frequency values for terms t_u and t_v. The Euclidean distance $du(t_u, t_v)$ between term t_u and term t_v is defined as:

$$du(t_u, t_v) = \sqrt{\sum_{i=1}^{n} (\vec{t}_{u,i} - \vec{t}_{v,i})^2}$$

4.2.2. Cosine Angle Distance

The cosine of the angle between two vectors in n-dimensional space is calculated and normally used to determine inter-document similarity.

Definition: Let $\vec{t}_u = (t_{u,1}, t_{u,2}, \ldots, t_{u,n})$ and $\vec{t}_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,n})$ be two vectors of document frequency values for terms t_u and t_v. The Cosine angle distance

Definition: Let $\vec{t}_u = (t_{u,1}, t_{u,2}, \ldots, t_{u,n})$ and $\vec{t}_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,n})$ be two vectors

$dc(t_u, t_v)$ between term t_u and term t_v is defined as:

$$dc(t_u, t_v) = 1 - \frac{\vec{t}_u \cdot \vec{t}_v}{|\vec{t}_u| \times |\vec{t}_v|}$$

4.2.3. Set-intersection Distance

This distance measurement is derived from a classical information retrieval similarity measurement called the Hamming distance [2], which was originally used as a similarity measure between strings. In an n-dimensional binary Vector Space Model, the Hamming distance between two vectors is also the Manhattan [11] distance.

This distance function is selected to directly capture the document co-occurrence of terms. Intuitively speaking, words are similar if they tend to appear in the same set of documents together. In practice, the intersection of the two vectors v_1 and v_2 are taken. If both values of v_1 and v_2 in column i are positive, the column i of the resulting vector is 1. Otherwise, it is 0. We then sum up the columns in the resulting vector and normalize the result. This is the similarity between vectors v_1 and v_2. Their distance is computed as 1 – similarity.

Definition: Let $\vec{t}_u = (t_{u,1}, t_{u,2}, \ldots, t_{u,n})$ and $\vec{t}_v = (t_{v,1}, t_{v,2}, \ldots, t_{v,n})$ be two vectors of document frequency values for terms t_u and t_v. The Set-intersection distance $ds(t_u, t_v)$ between term s_u and term s_v is defined as:

$$ds(t_u, t_v) = 1 - \frac{|\vec{t}_u \wedge \vec{t}_v|}{\min(|\vec{t}_u|, |\vec{t}_v|)}$$

The numerator of the right hand side of the equation finds out how much overlapping there are between the two terms. This number is then divided by the minimum of the cardinality of the two. We use this number as the denominator because we want to capture the set relationships between the two terms locally. This point is elaborated further in section 4.4.

4.3. Clustering Process

After the proximity matrix is constructed, using either one of the three methods mentioned above, we are ready to carry out the agglomerative clustering process. The clustering algorithm is outlined as follows [8]:

```
Input: A set of terms $T$.
Output: A set of groups $G$ containing terms in $T$.
1: let each term $t$ be in a single group $\{t\}$
2: let $G$ be a set of groups
3: while $|G| > 1$ do
4:    choose $\Gamma, \Delta \in G$ where $d(\Gamma, \Delta)$ is the smallest
5:    remove $\Gamma$ and $\Delta$ from $G$
6:    let $\Phi = \Gamma \cup \Delta$
7:    insert $\Phi$ into $G$
8: end while
```

To carry out step 4, we need a measurement of distance between subsets Γ and Δ. Here, we have two options [20]:

Single-linkage clustering: merge two clusters with the smallest minimum pair-wise distance. In other words, the distance between two clusters is the distance of their most similar members.

Complete-linkage clustering: merge two clusters with smallest maximum pair-wise distance. In other words, the distance between two clusters is the distance of their most dissimilar members.

4.4. Clustering Scenario

This section focuses on illustrating how distance functions affect the clustering result thus affecting the keywords that we choose from the dictionary.

Assuming that in the set of sampled words, we have: $S = \{a, b, c, d\}$, and that there are 6 documents in the document space. $D = \{1, 2, 3, 4, 5, 6\}$. In Table 2, it shows which document contains which terms.

Document	***Content***
1	*a*
2	*a, b*
3	*a, c*
4	*c*
5	*c*
6	*d*

Table 2: Example documents and their contents

The term-document relationship can be translated into vector representations. The length of each vector equals to the number of total documents in the space. In this particular case, each vector has a size of 4. Each number in the vector represents if the term appears in the document. The document distribution of each term in a vector representation is described in Table 3. We also assume that we want to select two keywords out of the four as our final dictionary.

Term	Document Vector
a	<1, 1, 1, 0, 0, 0>
b	<0, 1, 0, 0, 0, 0>
c	<0, 0, 1, 1, 1, 0>
d	<0, 0, 0, 0, 0, 1>

Table 3: Document vectors of each term.

In this table, each vector represents how a particular term is distributed in the entire document space. For example, term *a* appears only in documents 1, 2, and 3.

Figure 8 gives a graphical representation of the overlapping relationships amongst those terms. Each dot accompanied with a number in the graph represents a document. The number indicates which document it represents. Each circle with *S*(*q*) in it represents a set of results for issuing *q*. For example, S(*a*) = { 1, 2, 3 }.

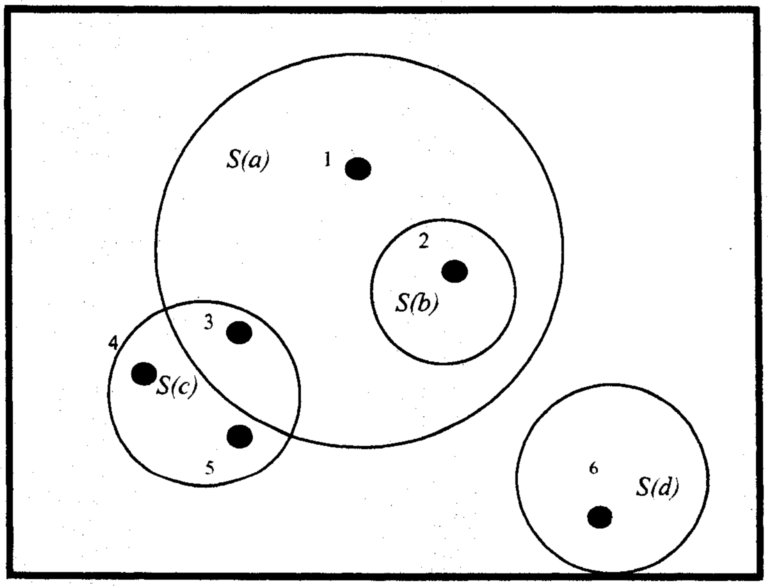


Figure 8: Set Representation of clustering scenario.

From the set representation figure above, we can easily see the overlapping relations between terms. We summarize that there are three types of relationships between terms.

Distinct: Two terms t_1, t_2 are *distinct* when $|S(t_1, t_2)| = 0$. For example, Terms *a* and *d* said to be *distinct* with each other. The two distinct terms should be furthest away from each other, and thus should have a distance of 1, which is the largest distance.

Intersect: Two terms t_1, t_2 are *intersect* when $|S(t_1, t_2)| > 0$. For example, terms *a* and *b* intersect with each other. When two terms intersect, they should have a distance between 0 and 1.

Eclipse: Term t_1 eclipses over term t_2 when t_1, t_2 intersect and $S(t_1, t_2) = S(t_2)$. Term *a* *eclipses over* term *b*. This is an inclusion relationship, which is important because when one term eclipses over another, and we issue both queries one after another to the search engine, we will only get more duplicates rather than new unique results. If we issue t_1 and t_2 sequentially to the search engine, then after getting $S(t_1)$, the entire set of $S(t_2)$ will become duplicates because $S(t_2) \subset S(t_1)$, and we will get $|S(t_2)|$ number of duplicates. If a lot of queries have this type of relationship, then we will have high *OR* thus decreases the quality of extraction process. Therefore, when one term eclipses over another, we should say that their distance is 0, which causes the cluster method to choose only the query that has a bigger number of results.

After calculating the distances between each pair of the terms, we get the distance table using the *set-intersection distance function*.

	a	*b*	*c*	*d*
a	0			
b	0	0		
c	0.67	1	0	
d	1	1	1	0

Table 4: Distance table

	d	*a*	*c*
a	1		
c	1	0.67	
b	1	0	1

Table 5: Compact representation of distance table.

From the distance table (Table 4), we can see that those terms that are distinct with each other have distance 1. Those terms that have eclipse relationship (*a* and *b*) have distance 0. Those terms that have intersection relationship (*a* and *c*) have distance between 0 and 1. Note that because the distance between two keywords are commutative (meaning $d(a, c) = d(c, a)$), and that the distance of a word to itself is always 0, therefore in practice, we do not need to construct a 4 x 4 table with every cell fixed. Instead, we just need to construct a 3 by 3 table with 5 cells fixed to cover all possible pair-wise combinations (Table 5). This way, it saves a lot of space while the dimension of the table (total number of documents in the space) gets really large.

Finally, by applying the Hierarchical Agglomerative Clustering algorithm outlined in section 4.3., we obtain the dendrogram of the four terms as follows:

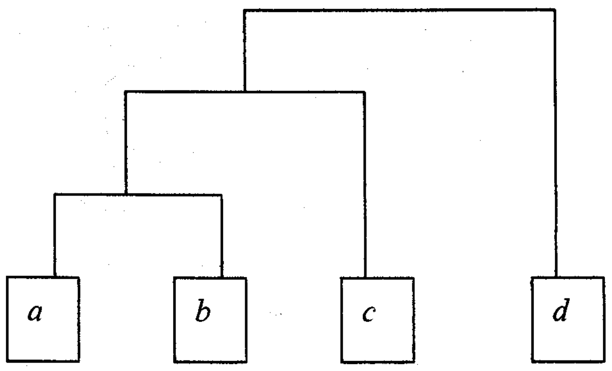


Figure 9: Dendrogram.

Now, we need to select two terms from the four terms. By traversing the dendrogram, *a*, *b* and *c* are put into one cluster while *d* is put into another. Recall that we use frequency to choose dictionary words from clusters. Both *a* and *c* have the same frequency 3. Let us first suppose that our crawler chooses *a* over *c*, then the final dictionary consists of queries *a* and *d*. Thus, if we issue queries *a* and *d* to the data source, documents 1, 2, 3, and 6 will be returned as results. We get an *HR* of 4/6 = 67%, and an *OR* of 4/4 = 1.

Now, if the crawler chooses *c* over *a*, then the final dictionary will comprise *c* and *d*, therefore after issuing them to the data source, we get documents 3, 4, 5, and 6. In this case, they have exactly the same *HR* and *OR* as the previous case.

CHAPTER V

DATA EXTRACTION USING SET PACKING

In this chapter, we are going to introduce the set packing method and the set covering method.

5.1. Set Packing Overview

In the set packing problem, we have a set of subsets $S = \{S_1, ..., S_m\}$ of the finite universal set $U = \{1, ..., n\}$. The goal is to find the largest number of mutually exclusive sets [34]. The complexity of the general maximum set packing problem has been proved to be NP-Complete as it was introduced as one of the famous *Karp's 21 NP-Complete problems* [21].

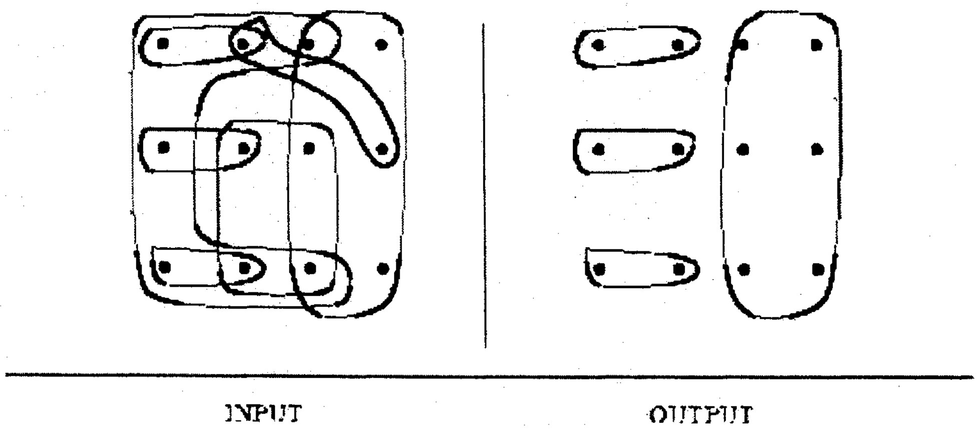


Figure 10: Set packing [34].

In our application, each subset in the set packing problem corresponds to the set of resulting documents for a particular query. The set packing version of our problem is to find queries which, after being issued to the data source, will return mutually exclusive sets of results. In other words, no overlapping is tolerated. We use a greedy algorithm outlined in the next section to select the list of dictionary words. In our extraction model, we can choose to relax this no-overlap condition so that the resulting sets of documents have a rate of overlaps less than a certain tolerance threshold.

5.2. The Set Packing Process

As stated in the previous section, by utilizing the set packing method, we aim to select the set of words such that the overlaps amongst them are controlled with a certain level. The algorithm of finding mutually exclusive words is outlined as follows:

```
Input: $C = \{t_1, t_2, ..., t_n\}$, the set of candidate words
Output: $D$, the set of selected words from $C$.
1. randomly choose $t_i$ from $C$
2. insert $t_i$ into $D$.
3. while $|D| <$ limit do
    4. choose $t_i \in C$ s.t. $S(t_i) \cap S(t_j) = \emptyset \; \forall t_j \in D$
    5. remove $t_i$ from $C$
    6. insert $t_i$ into $D$
7: end while
```

Through our experiments, we find that our snapshot set usually contains not enough mutually exclusive words. As a result, for data sources with result limits, the Hit Rate cannot go high because there are simply not enough words whose results can cover the entire data set. Thus, we want to relax the condition of mutual exclusiveness to increase the coverage. We define a tolerance Δ. We want Δ to be the maximum amount of overlap that can exist between any pair of words in *D*. Now, we have to have a way to describe the "overlap" relation between two words. We use the set-intersection distance introduced in Section 4.2.3 to evaluate how much overlapping there are between any two given words. Line 4 of the set packing algorithm then becomes:

```
4. choose $t_i \in C$ s.t. $ds(t_i, t_j) >= \Delta, \forall t_j \in D$
```

In practice, this relaxation of condition can improve the *HR* while sacrificing the value of *OR*. A detailed analysis using experimental data will be presented in Chapter 6.

5.2.1. Set Packing Scenario

In this section we look at an example of word selection using the set packing method. Here, we use same sample data set as we have used in section 4.4. Again, we want to choose 2 keywords as our final dictionary out of the four.

In the beginning, $C = \{a, b, c, d\}$, and $D = \emptyset$. The first step is to pick one term randomly. Suppose that we chose term *a*, and added it to *D*. Now, $C = \{b, c, d\}$, and $D = \{a\}$. Out of candidates *b*, *c*, and *d*, one has to be chosen and inserted into *D*. Let us first assume that our overlap condition holds strictly, meaning that words in *D* are not allowed to have any overlaps with each other. Now, according to Table 4 and Figure 8, only term *d* has no overlaps with term *a*. Both *b* and *c* overlap with term *a*. Therefore, our final dictionary $D = \{a, d\}$. If we issue the dictionary to the search engine, the results we are getting will have an *OR* of 4/4 = 1, and an *HR* of 4/6 = 67%.

5.3. The Set Covering Method

The set covering method is similar to the set packing method in terms of their definitions, however with a different approach.

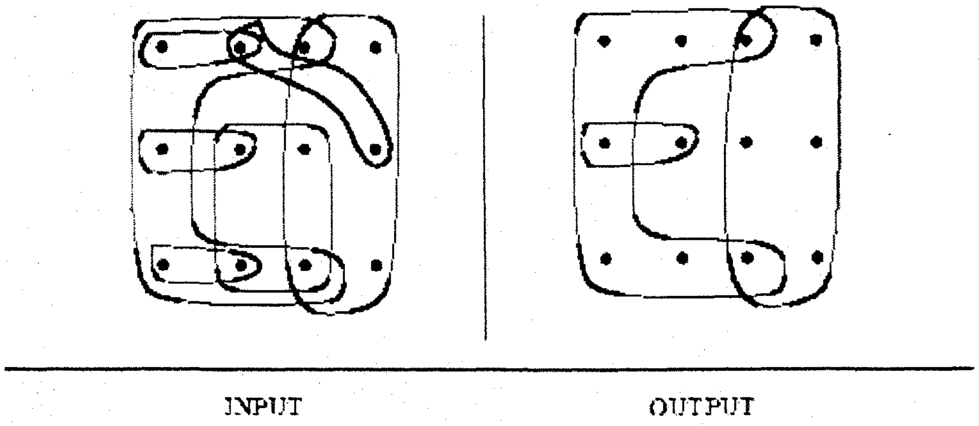


Figure 11: Set covering [34].

In a set covering problem, we again have a set of subsets $S = \{S_1,...,S_m\}$ of the universal set $U = \{1,...,n\}$. The goal is to find a full coverage of the total space while minimizing the number of sets used [11]. In our application, again the results from each query corresponds to a subset, and set covering aims on covering the maximum number of documents using minimum number of queries.

Authors in [29], who view the deep web extraction process as a set covering problem, have proposed the *adaptive* data extraction method to solve it. The adaptive data extraction method is a greedy approach that attempts to maximize the potential gain of the next keyword issued to the data set. In each iteration of the method, a new keyword is chosen based on its "Estimate Efficiency" among all keyword candidates. The estimate efficiency for each candidate keyword q_i is calculated using equation $\frac{Pnew(qi)}{Cost(qi)}$, in which the numerator stands for the amounts of *new* documents that can be returned for q_i, and the denominator is the cost of issuing q_i.

In the next chapter, we will compare the data extraction effectiveness among cluster method, set packing method and set covering method.

CHAPTER VI

EXPERIMENTS

In this chapter, details of the experiments performed will be described. Also, advantages and disadvantages of all methods compared will be discussed.

6.1. Overview

As we have discussed in previous chapters, there are three main stages that comprise the entire deep-web data extraction process. They are: word sampling stage, word selection stage, and results sorting stage. As outlined in Table 6, each stage has a number of options that we can choose to better optimize the over-all efficiency of the process. Also, every method used in each stage can have different parameter configured to better facilitate the extraction process. The different combinations of options for each method in each stage will cause different outcomes. We will discuss them in details in this chapter.

Word Sampling Methods	Word Selection Methods	Results Sorting
Most Popular Words	Random	Random
Random Words	Cluster	Sort by Relevance
Optimal Words	Set Packing	Sort by Rank

Table 6: Options/variations in each stage.

6.2. Data

We have chosen two different data sets to conduct our experiments. Both data sets are considered benchmarks in the field of machine learning and information retrieval [8][40].

The first one is the 20 Newsgroups data set (20NG for short) [35]. It contains 20,417 Usenet postings from 22 different newsgroup topics. In our experiments, we took a random subset of 3000 documents as the data source.

The other data set that we chose is a subset of the OHSUMED database. Our OHSUMED data set is a corpus that contains 56,984 abstracts from major medical journals. In our experiment, a subset of 3450 documents is randomly chosen.

Name of Dataset	**Total Number and size of Documents in target space**	**Number and size of Sampled Documents used for extraction**
20 Newsgroups (20NG)	20,417	3000
Ohsumed Medical Abstracts (OHSUMED)	56,984	3450

Table 7: Descriptions of data sets used.

6.3. Experiment Setup

In each of the methods outlined below, a dictionary is first generated, after which the orders in the dictionary are scrambled to ensure randomness. The size of the dictionary is controlled to be 150. As described in Section 2.5, we want to maximize the fraction of documents while minimizing both the overlapping rate and the number of queries issued. In our experiments, we keep the number of queries controlled, so that we need to compare only two variables: hit rate (*HR*) and overlapping rate (*OR*).

After the dictionary is constructed using one of the word selection methods, a sequential search is carried out on data source indexed using the Apache Lucene package [1]. The results returned from the data sources are constraint with a limit *k*. Recall that *k* is an artificial constraint of the maximum number of results a query is allowed to return. All excess results will be ignored. This is done to simulate the per-query result limits on major search engines.

Search results are then sorted by one of the sorting methods described in section 3.4. In our experiments, we mostly use No Sort policy to ensure a fair comparison between different methods. Other sorting policies will also be analyzed.

6.4. Comparison of Different Extraction Methods

In this section, we compare the performance difference of various data extraction methods.

6.4.1. Comparison of Word Sampling Methods

The first stage after parsing the original documents is to sample words down to a manageable size that is feasible for doing information retrieval process. This stage is called the word sampling stage. We compare three different methods of sampling words from the original vocabulary.

After parsing the 3000 documents from the newsgroup data set, there are 36,000 unique terms. Our goal at the word sampling stage is to sample 2000 words for further data analysis. Table 6 explains the type of data source, type of methods used for each stage and data source restriction.

We can see from Figure 12 that the optimal word sampling method is the best for choosing good quality terms. Good quality terms stand for those terms that will produce low overlap rates and high hit rates when issued to the data source. The most popular method produces slightly inferior results because of the potentially high overlaps between terms. Random sampling is the worst because it picks up a lot of terms with low coverage meaning the number of results is very small, which is the reason for the line for random method much shorter than the other two.

Data Source	**Word Sampling**	**Word Selection**	**Sorting Policy**	**k**
20 Newsgroup	Optimal, Random, Most Popular : 36,000 into 2000	Random	No sort	50

Table 8: Test environments for comparison of word sampling methods.

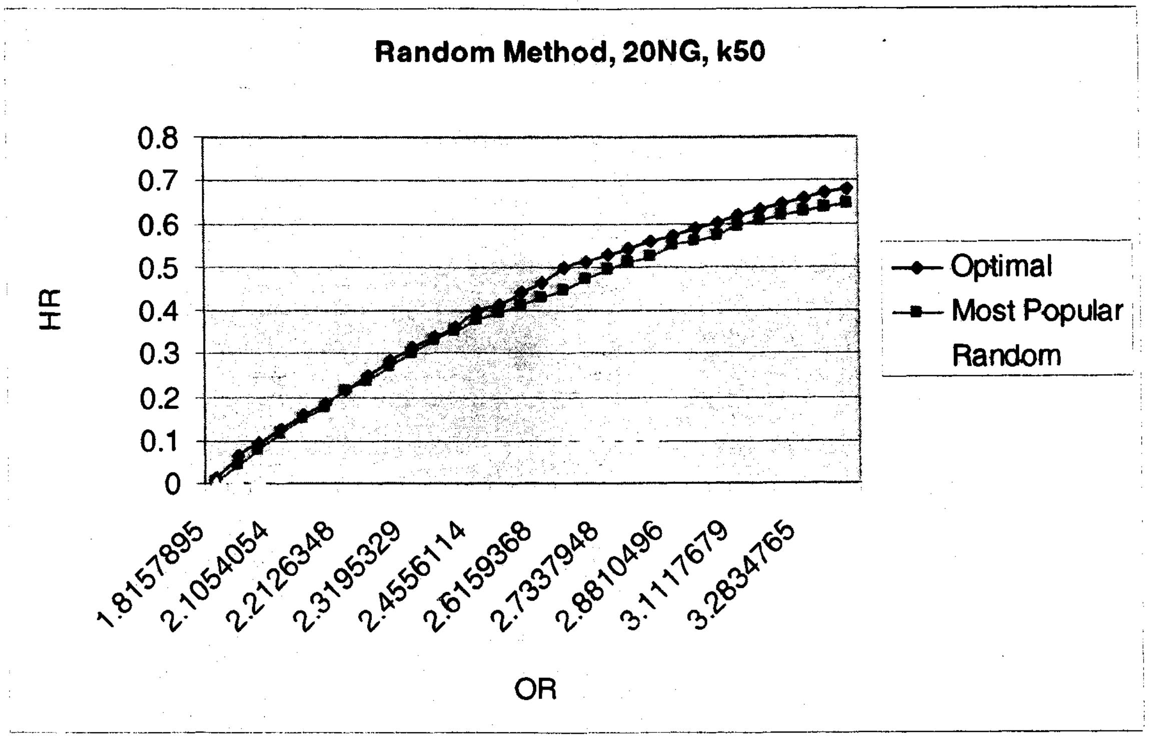


Figure 12: Comparison of word sampling methods.

6.4.2. Comparison of Cluster Distance Functions

As described in chapter 3, we have three options of distance functions to carry out our clustering process, and here we want to select the best one and to use it as the default distance function for further experimentation. The cluster method is performed using three different distance functions. The graph below shows how each of the three distance functions performs. This test is conducted in the following context:

Data Source	Word Sampling	Word Selection	Sorting Policy	k
20 Newsgroup	Optimal: 36,000 terms into 2000 terms	Cluster	No sort	50

Table 9: Test environment for distance function comparison.

Result in Figure 13 shows that the best scoring function is the set intersection distance function. We can see that all three functions eventually perform similarly when they reach the 80% to 85% *HR*. However, the set-intersection distance function performs consistently well throughout the extraction process. This is because that the set-intersection distance function best captures the set-overlap relationships between terms and the documents they are distributed in. The other two distance functions are more suitable for other types of information retrieval purposes, but not for our particular goal, which is to minimize the *OR* while maximizing the total percentage of returns. In future sections, we use set intersection distance function as the default function when carrying our clustering process, unless stated otherwise.

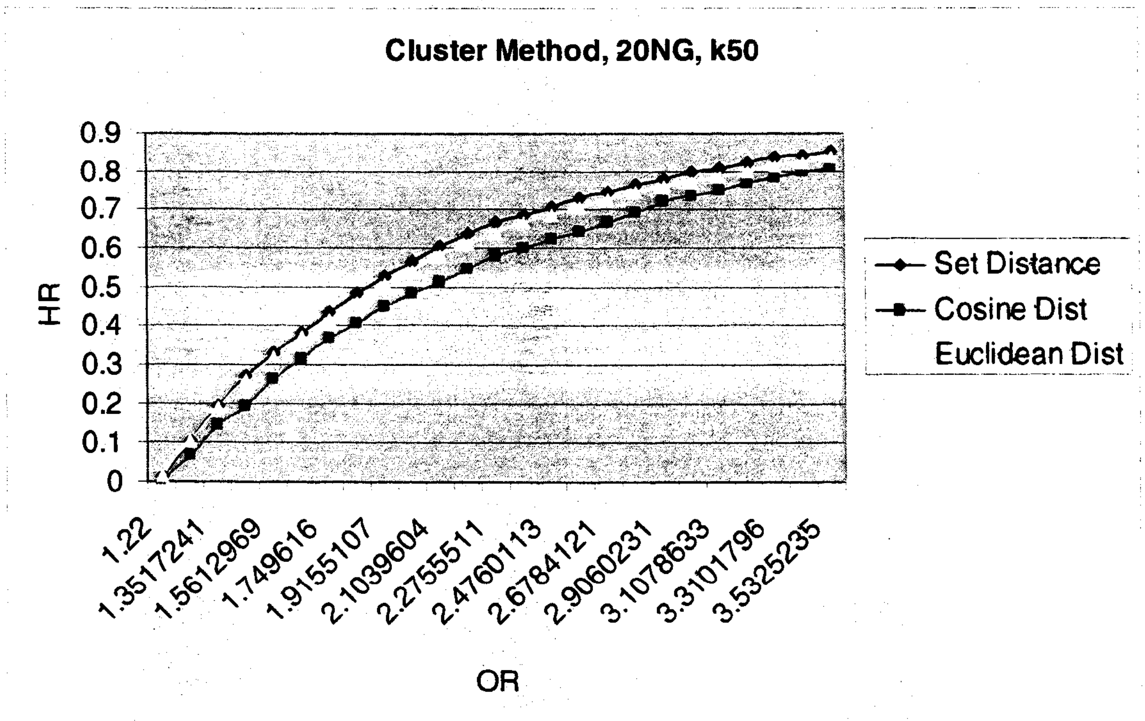


Figure 13: Comparison of distance functions.

6.4.3. Comparisons of Different Word Selection Methods

In this section, we compare different word selection methods on two data sets, i.e., *20 Newsgroups*(20NG) and *Ohsumed medical abstracts*(OHSUMED). The word selection methods are: random method, cluster method, set packing method, and set cover method (adaptive method in [29]).

For the 20NG data set, we collected 36,000 unique terms from the 3000 document randomly chosen from the 20,417 documents. We processed only 3000 documents because of the limitation of computer power – for each word we need to produce an inverse vector, which consists of all the documents where the word occurs. When the document set is large, the vector will become too large to process. For the OHSUMED data set, 3450 documents are randomly chosen to carry out experiments.

Optimal word sampling is then used to downsize the vocabulary from 36,000 to 2000. Specifically, only words with frequencies ranging from 20 to 500 are chosen, with the goal to eliminate those words with too high or too low frequencies. After the word sampling, we run three methods (cluster, set packing and random) on the 3000-document data set and compare their performances in terms of *OR* and *HR*.

From the 2000 sampled words, we select 150 keywords using 4 different methods. We do this because we want to eliminate the number of queries issued (*Q*) in our evaluation criteria (see section 2.4), so that we can better focus on the *OR* and *HR*. Dictionary keywords from different methods are listed in the Appendix section.

For the cluster method, we construct 150 clusters from the 2000 sampled words, and then we pick one representative from each cluster as a keyword. We used set-intersection distance as the distance function for it produces the best results according to section 6.4.2. The complete-linkage clustering method is applied, meaning that the distance between two clusters is computed as the distance between the most dissimilar members of those two. We used the clustering package from Lingpipe [25] to assist us during the process of clustering.

For the set packing method, we chose 0.6 as the Δ value, for Δ values that are less than 0.8 will all produce similar results based on optimally sampled words. For Δ greater than 0.8, not enough queries can be generated to meet the 150 query requirement.

The set covering method returns word with high document frequency, because it is based on an assumption that no limits are imposed on return results. However, we will take only a maximum of k (here k=50) number of returns, to mimic search engine that impose limits on return results.

To illustrate a concrete example of the search process, we take the 1st, 2nd, 50th, 100th, and 150th query for each method. For each query, we compare their: hits (estimated results), real unique results (always less than 50 because of the limitation assumption), accumulated results, and accumulated unique results. Table 10 is for cluster method. Table 11 is for set covering method. If a hits number is larger than 50, only the first 50 are included. Among those 50 results, there might be overlaps with previous search results. For example, the second word of the cluster method, "wrong", returns only 47 results out of the 50 we have chosen, because the other 3 results overlap with the results we get from the first query, "times". As we issue more queries, the number in the accumulated unique results column grow, however, the number in the accumulated results column go up even faster. This is inevitable because unique results are always less than or equal to accumulated results. As we can see in these two tables, *OR* starts to be low at the beginning, however, it climbs up as we issue more queries, meaning that the overlaps between the results increase.

Query Index	Query	Hits	Real Unique Returns	Accumulated Results	Accumulated Unique Results	HR	OR
1	times	187	50	50	50	0.016667	1
2	wrong	216	47	100	97	0.032333	1.030928
50	uk	136	23	3091	1565	0.521667	1.97508
100	send	156	50	6144	2219	0.739667	2.768815
150	day	244	15	9249	2554	0.851333	3.621378

Table 10: Step details, cluster method

Query Index	Query	Hits	Real Unique Returns	Accumulated Results	Accumulated Unique Results	HR	OR
1	larry	34	34	34	34	0.009855	1
2	hour	35	33	69	67	0.01942	1.029851
50	iii	40	23	2328	1046	0.303188	2.225621
100	love	114	11	4448	1666	0.482899	2.669868
150	craig	29	14	6792	2144	0.621449	3.16791

Table 11: Step details, set covering method

Figure 13 describes the relationship between *HR* and *OR* for various query selection methods. Each node in the diagram indicates an *OR/HR* pair. *OR* values and *HR* values increase as shown in the diagram as the crawler issue more queries to the data source. The more queries we issue, the more *HR* and *OR* values increase. A perfect solution to the extraction problem will always have an *OR* of 1, meaning that all returns we get from the search engine are unique (fresh). In this case, we will see a straight line going up from an *HR* of 0%to 100% at *OR* = 1. However in reality, result sets intersect with each other and this is the reason why all lines shown have smaller slopes. From Figure 13, we can see that the cluster method has the best performance in terms of both *OR* and *HR*. This type of behavior is expected because the cluster method analyzes the entire data set, and then produces the dictionary with keywords that have less overlaps as well as high returns. All three other methods produce inferior results than the cluster method.

Data Source	**Word Sampling**	**Word Selection**	**Sorting Policy**	**k**
20 Newsgroup	Optimal: 36,000 terms into 2000 terms	Cluster, Set Packing, Set Covering, Random	No Sort	50

Table 12: Test environment for four word selection methods.

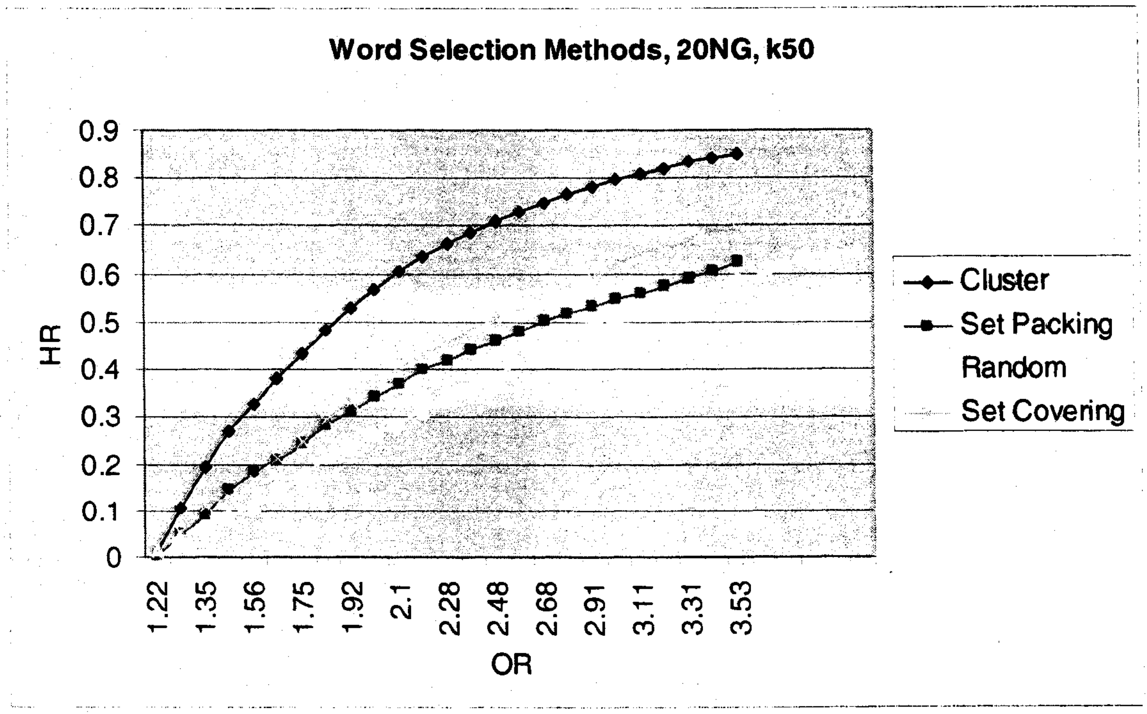


Figure 14: Comparison of 4 word selection methods (20NG).

Table 13 compares the four methods in detail by listing different *OR* values at different *HR* levels. We can see from this table that the cluster method outperforms all other methods at all different *HR* level. Also, the cluster method is able to extend the *HR* more than 75% which far exceeds the coverage of all three other methods.

As introduced in section 5.3, the nature of the set covering method (adaptive method) proposed in [29] is different from all other three. This method analyzes documents downloaded from data source and adaptively chooses the next query to issue to the data source. The set covering method chooses words that have the best *efficiency score* as the keywords for the final dictionary. The focus of the set covering method is to use minimal number of queries to extract most data. When there is a high level of overlaps, the algorithm is not efficient. Also, the set covering method assumes that each query can return unlimited number of result items, which is not practical in many cases.

	OR			
HR	*cluster*	*set covering*	*set packing*	*random*
15%	1.33	1.88	1.97	2.13
30%	1.51	1.99	2.3	2.3
45%	1.81	2.37	2.58	2.58
60%	2.1	2.75	3	3.02
75%	2.7	--	--	--

Table 13: Comparison of 4 word selection methods.

Table 14 illustrates the difference in the amount of real time it takes to perform each method. This data is collected from programs running on a computer with AMD Athlon 2000+ CPU, 1GB of memory, Windows XP with Java version 5.0. All methods are run to produce a keyword list of 150 words based on the 2000 optimally sampled words from 36,000 unique terms collected from the 3000 document subset of the 20NG data set. We can see that compared with the set covering method, cluster method and set packing method take a lot less time to produce the final result.

Set packing method and cluster method run fast because the term-frequency information is collected and stored in memory after the word sampling stage. Set covering method is slow because it has to search the data set every time it calculates the efficiency for candidate words. An "inefficient" model of set packing method could be constructed; however, it is out of the scope of this thesis.

Word Selection Method	Time
Random	100 milliseconds
Set Packing	1.8 seconds
Cluster	4 minutes
Set Covering	3 hours

Table 14: Time comparison of different word selection methods

A similar experiment has been conducted on the OHSUMED data set. In this experiment, a total of 3450 medical abstract documents have been randomly chosen from the 56,984 documents. All other variables of different methods are exactly the same as the one conducted on the newsgroup data set. However, in this data set, cluster method does not perform as well as the set covering method. This may be caused by the differences between the word-frequency distributions of the data sets. It is possible that the frequency range that we chose is not optimal for the cluster method to produce good results. Automatically choosing the frequency range is listed as one of the future works.

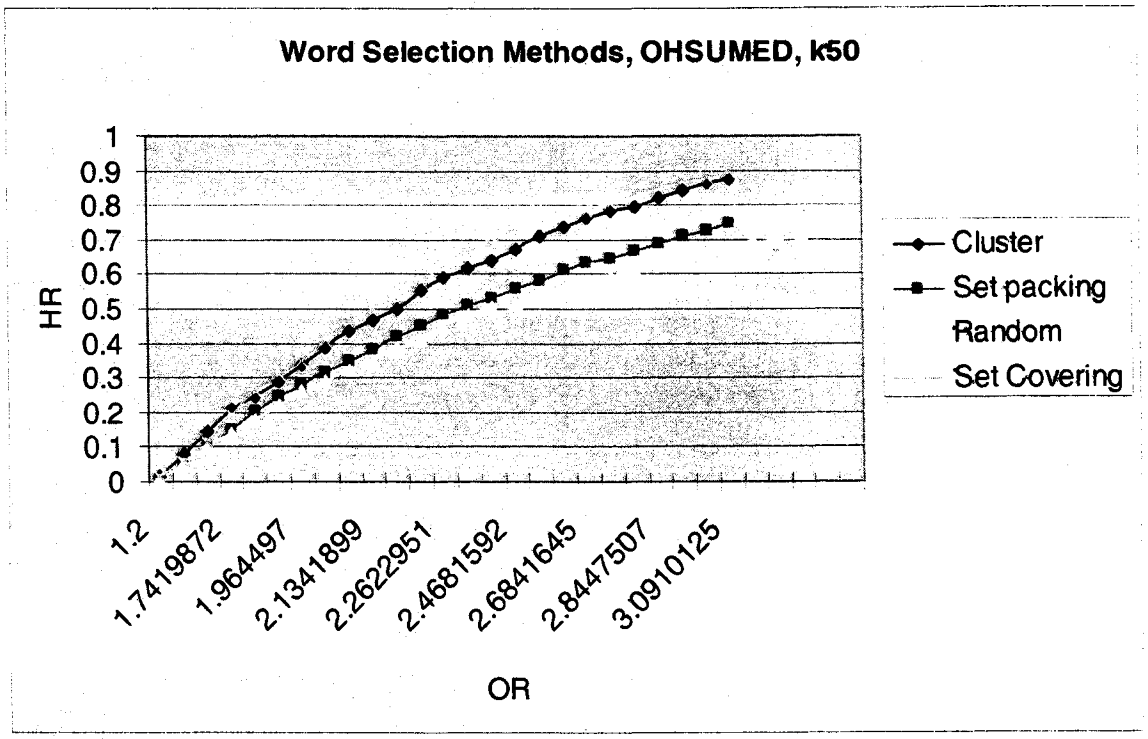


Figure 15: Comparison of 4 word selection methods (OHSUMED).

6.5. Comparison of Different Data Source Environments

Previous sections gave us a glance of how different data extraction methods behave. However, due to the different restrictions that different data sources impose, the performance of these methods can vary. In the next few sections, we discuss how restrictions of different data source environment affect data extraction methods.

6.5.1. Comparison of Different Result Sorting Policies

Data sources have different ways of sorting search results. Many data sources return only the top k results to the user. In this section we investigate how much impact different sorting policies have on our discovery process. Figures 15 and Figure 16 compare how different methods are affected by the sorting policies.

From Figure 15, we can see that the both sort-by-relevance policy and unsorted policy favor the cluster method, where the sort-by-rank policy will degrade the performance of the cluster method dramatically. In Figure 16, we can see that the set random method also benefits from the sort-by-relevance policy. Both the sort-by-rank policy and unsorted policy produce worse results. It is clear from both figures that the sort-by-relevance policy favors both word selection methods. This is because the sort-by-relevance policy provided by the data search engine sort results according to their TF/IDF scores [2], thus those documents that are "close to" the search query are returned. The closer they are to the search query, the further they are from other search queries, thus can reduce *OR*.

We also see inconsistent performances of different methods under the sort-by-rank policy. This is because of the particular sampled words set. In a sort-by-rank environment, the results for each query q issued depends on a ratio of $\frac{k}{k_1}$, where k is the maximum number of results permitted by the search engine and $k_1 = S(q)$, which is the expected number of results for q without the limit. For some dictionaries, this ratio might be large, while for some others this ratio is small, and thus affect the overall performance of the extraction method [27].

Data Source	Word Sampling	Word Selection	Sorting Policy	k
20 Newsgroup	Optimal: 36,000 terms into 2000 terms	Cluster, Random	Ranked Relevance No Sort	50

Table 15: Test environment for result sorting policies.

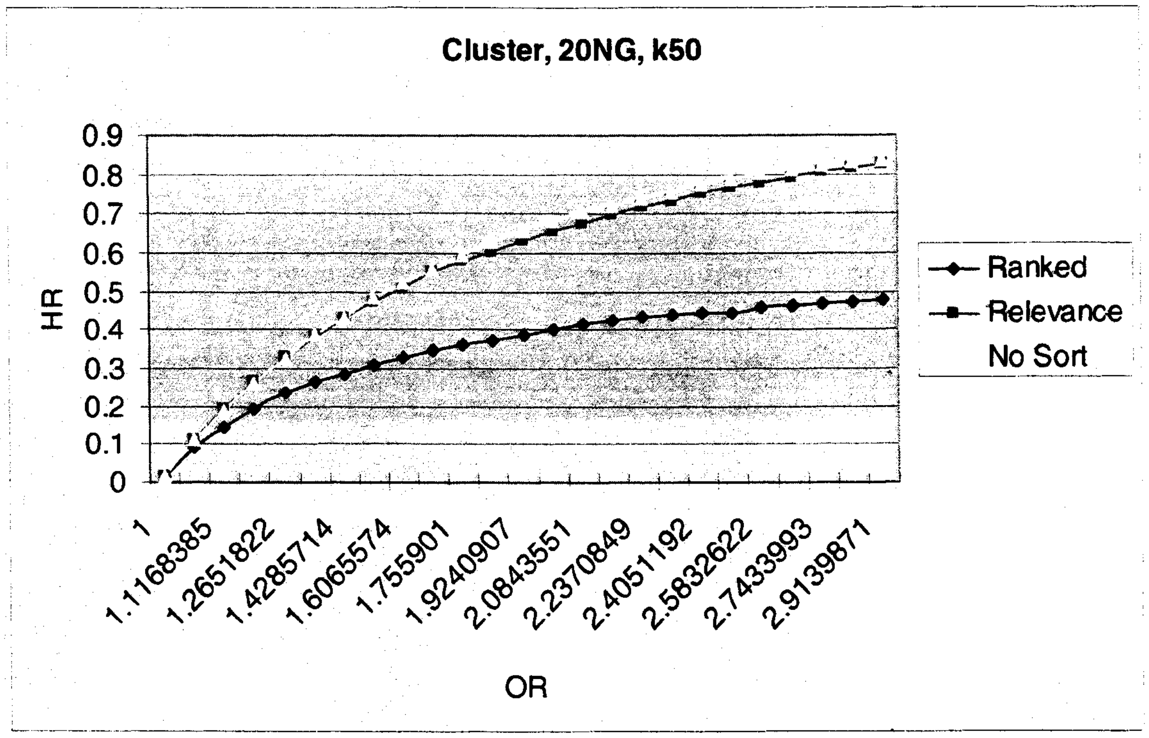


Figure 16: Comparison of sorting policies (Cluster Method).

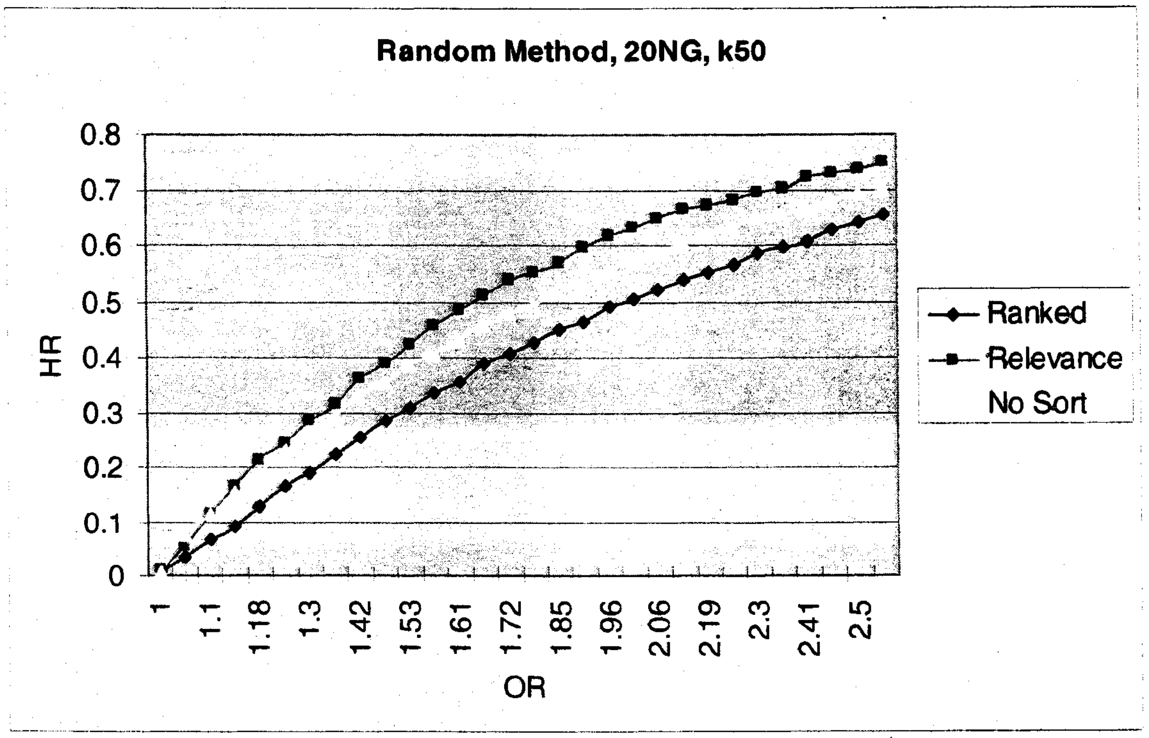


Figure 17: Comparison of sorting policies (Random Method)

From all the two figures shown above, we can see that the sort-by-relevance policy gives us the most advantage in terms of both overlapping rate *OR* and fraction of documents obtained *HR*. This is because that sort-by-relevance method provided by the data source engine uses TF/IDF score [2] to sort the results, so that documents which are close to the term that we search are returned. Recall the set covering model in Figure 3. By using sort-by-relevance policy, those the documents in a set that are close to the center are returned as results to the user, while those are scattered further to the center are ignored.

6.5.2. The Impact of *k*

As described earlier, each search engine has its own constraints on the maximum number of returns per query (denoted as *k*). This section investigates how different methods are affected under different k values. We pick *k* to be 50, 100, 150 and 400. For each *k* value, we compare how the four different methods perform.

In this section, we use a different setting from previous experiments. Specifically, 3000 documents from the 20NG data set are chosen to carry out the dictionary-generating process to generate a dictionary with 1000 keywords. Then, we use the dictionary generated to search the full data sets (20,417 for 20NG and 56,984 for OHSUMED). We do this because we want to show the differences more obviously. Also, because of the size of the data set, two methods (cluster and random) are used, each of which with two different word sampling methods (popular and random).

Data Source	**Word Sampling**	**Word Selection**	**Sorting Policy**	**k**
20NG(full), OHSUMED(full)	Popular, Random	Cluster, Random	No Sort	50,100,150,400

Table 16: Test environment for different *k* values for both data sources.

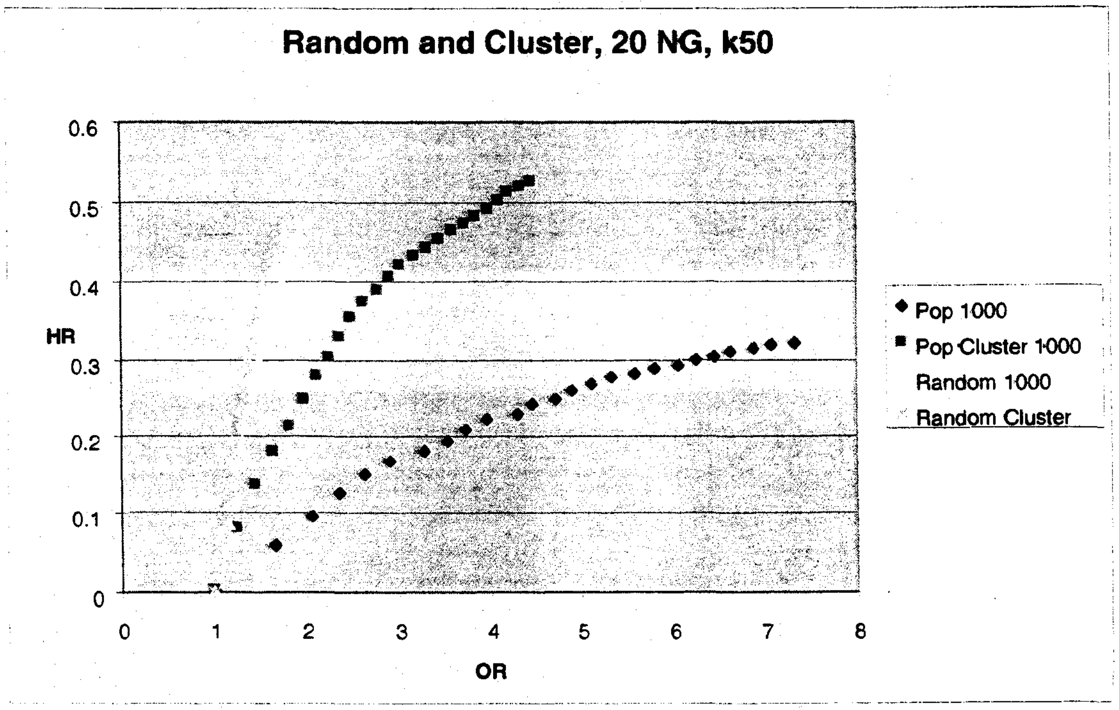


Figure 18: Comparison of k values, 20NG, k=50.

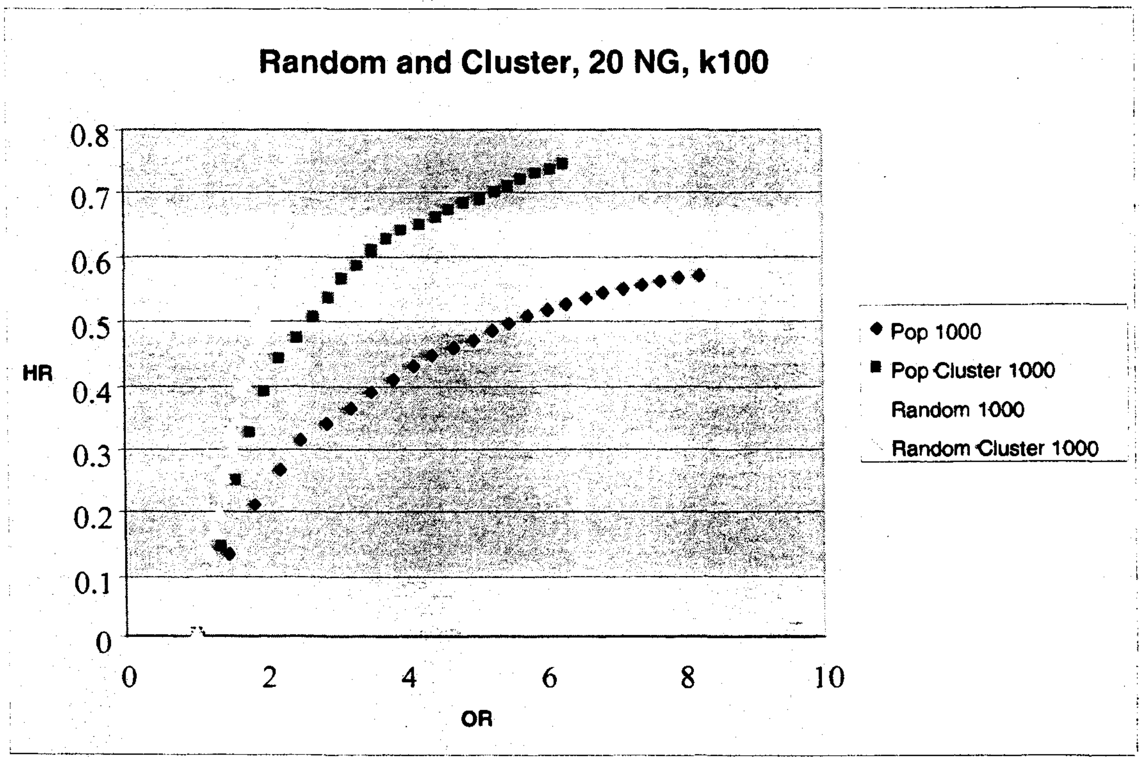


Figure 19: Comparison of k values, 20NG, k=100.

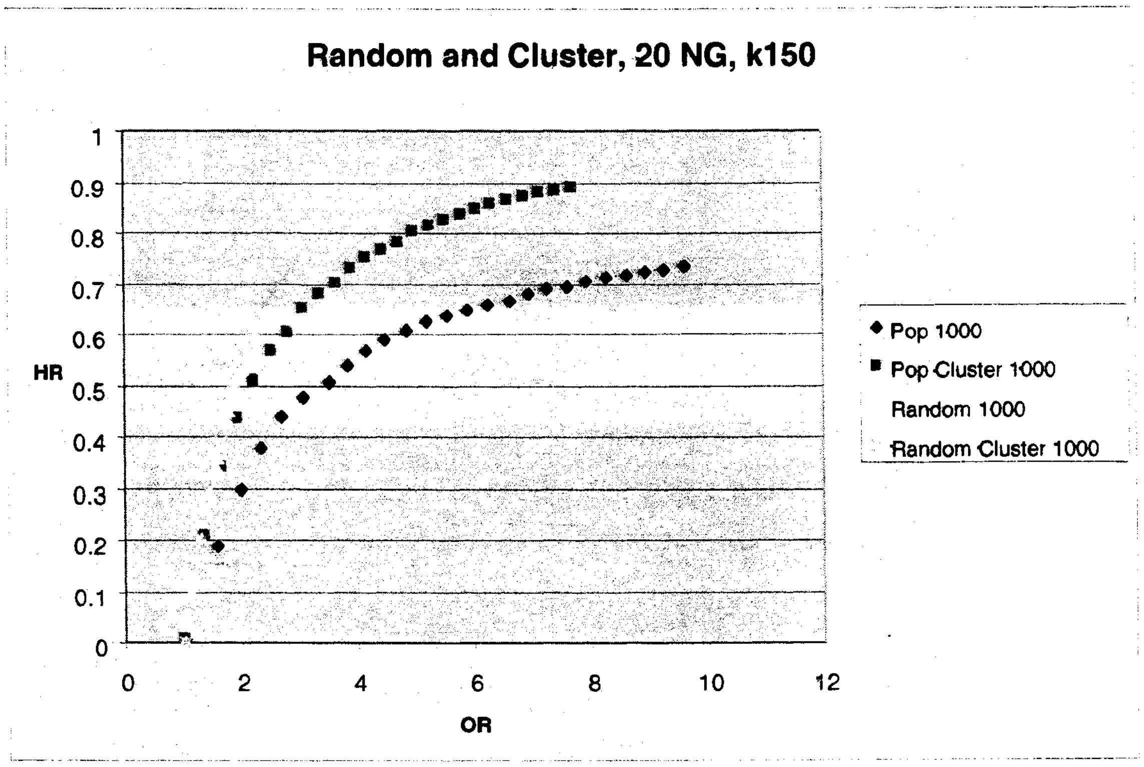


Figure 20: Comparison of k values, 20NG, k=150.

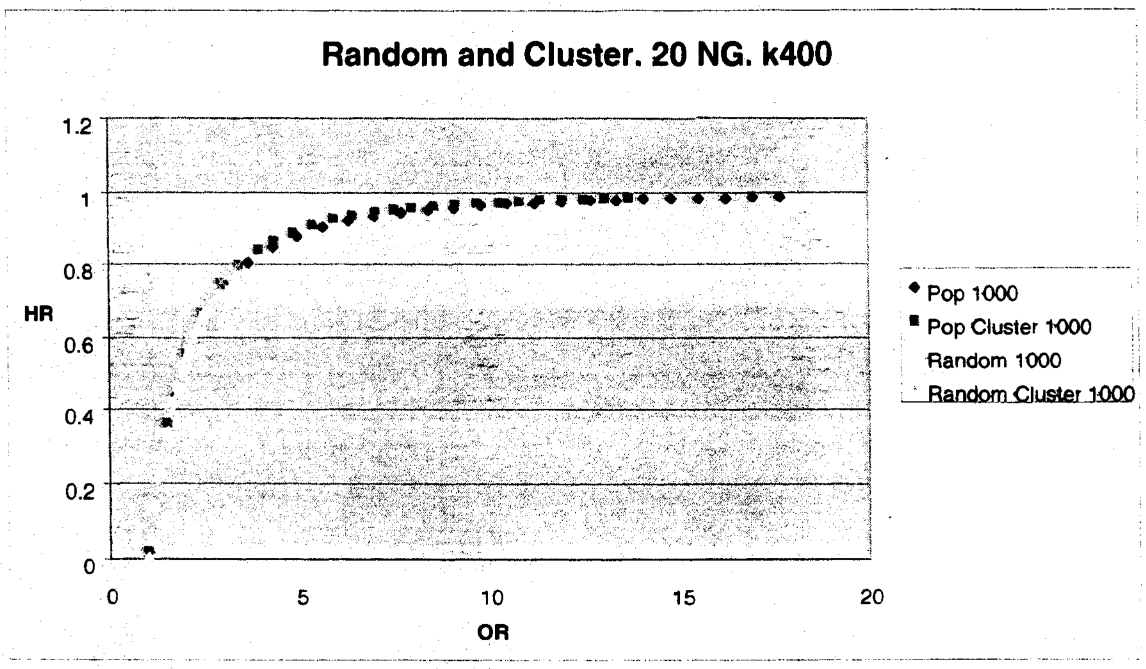


Figure 21: Comparison of k values, 20NG, k=400.

As we can see, as *k* gets larger, the differences amongst the four lines get smaller. When *k* reaches 400, there is hardly any difference between the four methods in terms of *OR*. However, in terms of *HR*, the popular-sampled cluster method is winning. With the same number of queries issued, and when *k* is less than 1.9% of the total number of documents in target space, the popular-sampled cluster method is the winner considering both *OR* and *HR*.

Similar results are shown in the OHSUMED data set. The difference is that the *k* threshold is proportionally larger than that of the newsgroup database, because of the bigger size of the data set. Again, we can see that when *k* reaches 1.9% of the total expected number of documents, the performance of four methods converge in terms of *OR*. In terms of *HR*, the popular-sampled cluster method has the best performance. The random-sampled cluster method is the best when *k*=200, in terms of both *OR* and *HR*. As *k* gets bigger, the random-sampled cluster method is only better in terms of *OR*, but not *HR*.

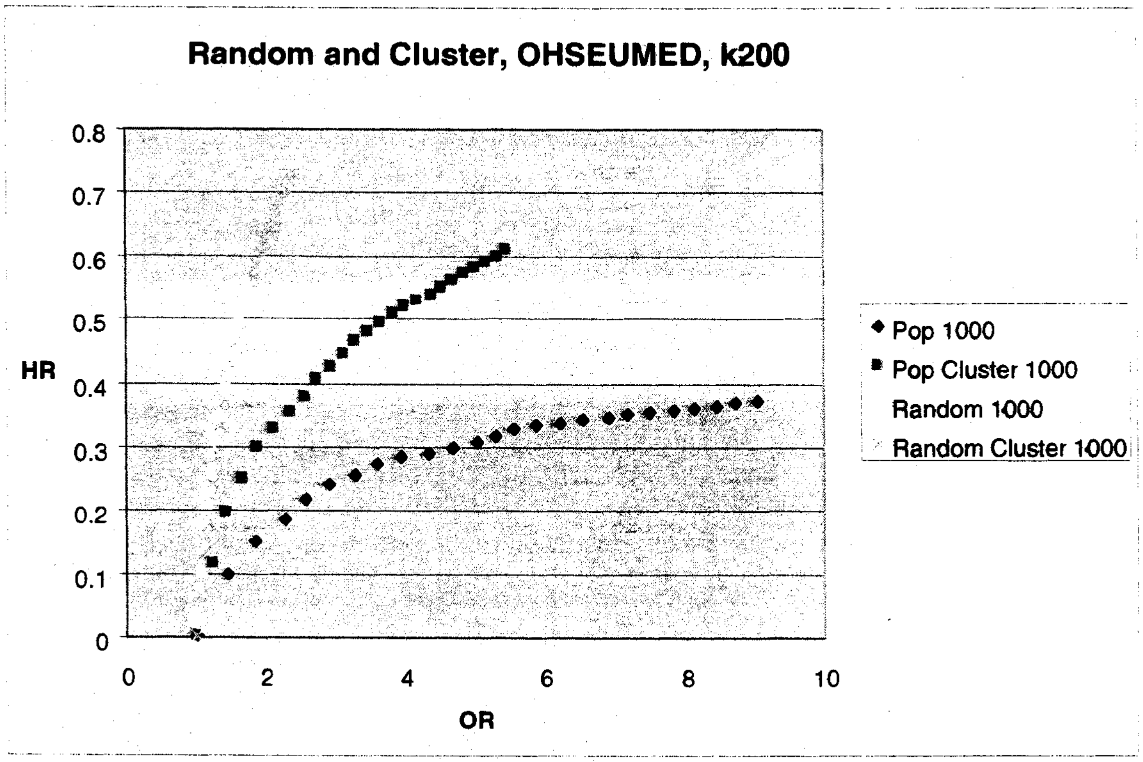


Figure 22: Comparison of k values, OHSUMED, k=200.

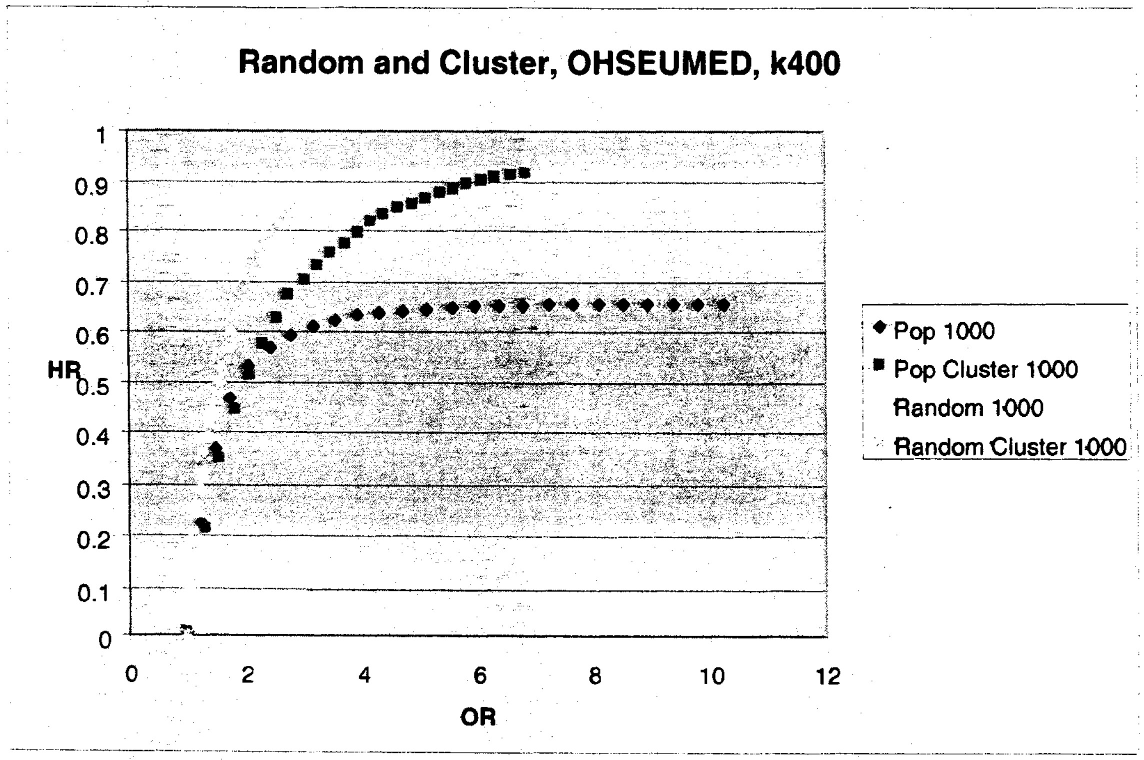


Figure 23: Comparison of k values, OHSUMED, k=400.

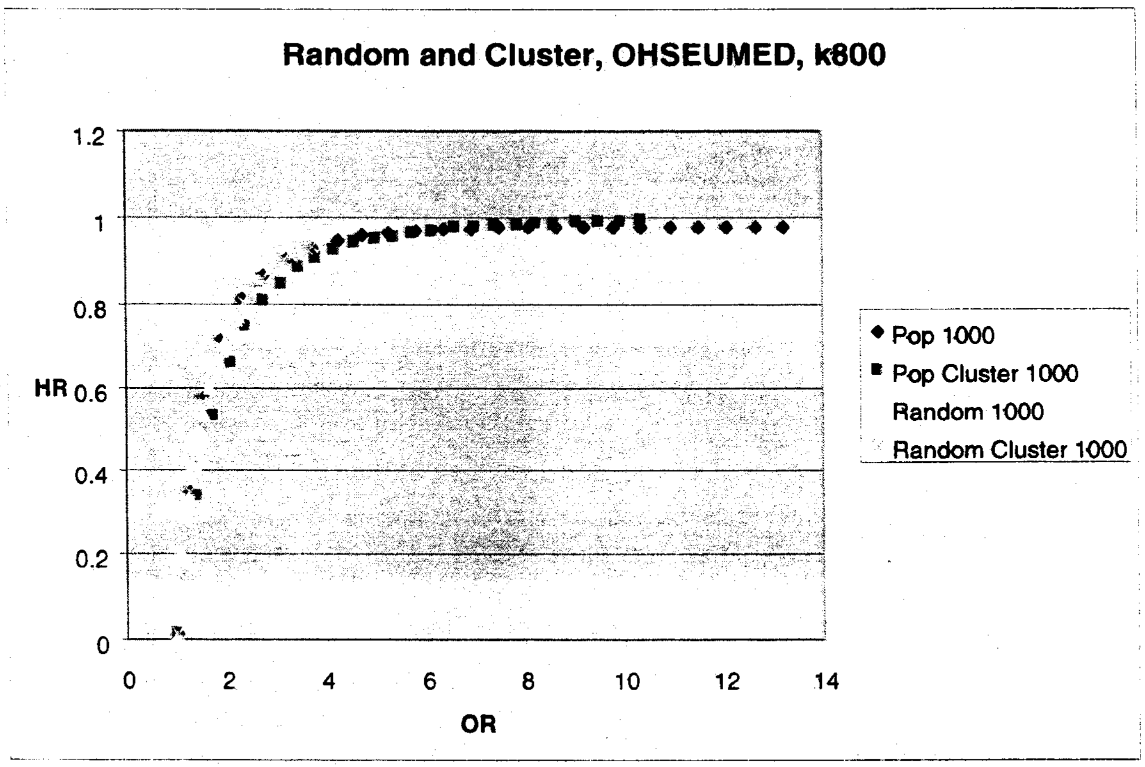


Figure 24: Comparison of k values, OHSUMED, k=800.

CHAPTER VII

CONCLUSION AND FUTURE WORK

As more documents on the internet are in the deep web, it is urgent for search engines to index and to make various deep web data sources searchable. The data extraction process presented in this thesis is a new way of extracting data from the deep web using a modified clustering algorithm aiming on minimizing the overlap rate of the results. This thesis also provides comparison between different deep web extraction methods as well as how they perform under different data source environments.

From our experiments, we conclude that given an optimal word frequency range, cluster method with set-intersection distance function outperforms all other methods (set packing, set covering and random) in the 20 Newsgroup data set. Even though the cluster method underperforms the set covering method in the Ohsumed medical abstracts data set, it is still better than the random method and set packing method.

The cluster method does have its limitations. For example, we cannot experiment on the entire data set using cluster method because of the memory limitation that is required by this method. Also, it will be necessary in the future to have a method that automatically calculates an optimal frequency range for the word sampling process.

In this thesis, we also discovered that sort-by-rank policy imposed by search engines can reduce the efficiency of deep-web extraction process. Sort-by-relevance and unsorted policies generally benefit the process.

Another restriction imposed by search engines, the limit of the number of returns, is a very important factor that affects the data extraction effeciency. The stricter the limitation, the more necessary it is for a deep web crawler to use good word sampling methods and word selection methods. We discovered that the smaller the *k* values, the harder it is for the data extraction process to perform well.

The scope of the thesis is limited to textual database with simple keywords query interface, which is common for document searching on the web. Query forms with multiple attributes or complex query grammar are not considered. In particular, a practical data extractor should utilize the query grammar to achieve better result.

References

[1] Apache Lucene. http://lucene.apache.org/.

[2] R Baeza-Yates, B Ribeiro-Neto, "Modern Information Retrieval", Addison Wesley, 2000.

[3] L. Barbosa and J. Freire, "Siphoning hidden-web data through keyword-based interfaces", In SBBD, 2004.

[4] K. Bharat and A. Broder. 1998. A Technique for Measuring the Relative Size and Overlap of the Public Web Search Engines. In Proceedings of WWW7, pages 379--388.

[5] Michael K. Bergman (Aug 2001). "The Deep Web: Surfacing Hidden Value". The Journal of Electronic Publishing 7 (1).

[6] Alfonso F. Cárdenas, Analysis and performance of inverted data base structures, Communications of the ACM, v.18 n.5, p.253-263, May 1975.

[7] James Caverlee, Ling Liu, and David Buttler, Probe, Cluster, and Discover: Focused Extraction of QA-Pagelets from the Deep Web

[8] Soumen Chakrabarti, Mining the Web – Discovering Knowledge from Hypertext Data, Morgan Kaufmann Publishers, 2003.

[9] S. Chakrabarti, M. van den Berg, B. Dom. Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery, in: 8th World Wide Web Conf., May 1999.

[10]K. C.-C. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the web: Observations and implications. Technical report, UIUC.

[11]T.H. Cormen, C.E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 2nd Edition. MIT Press/McGraw Hill, 2001.

[12]V. Crescenzi, G. Mecca, P. Merialdo, RoadRunner: towards automatic data extraction from large web sites, VLDB J. (2001) 109–118.

[13]M. Diligenti, F. Coetzee, S. Lawrence, C.L. Giles, M. Gori. Focused Crawling using Context Graphs, in: 26th Int. Conf. on Very Large Databases, VLDB 2000, September 2000.

[14]Ferris, C. and Farrell, J., "What are Web Services?", Commum, ACM 46, 6 2003, pp.31.

[15]Daniela Florescu, Alon Y. Levy, and Alberto O. Mendelzon. Database techniques for the world-wide web: A survey. SIGMOD Record, 27(3):59–74, 1998.

[16]G. Gonnet and R. Baeza-Yates. Handbook of Algorithms and Data Structures. Addison-Wesley, Workingham, England, 2nd edition, 1991.

[17]Google. http://www.google.com/ .

[18]Lars Holst, A Unified Approach to Limit Theorems for Urn Models, Journal of Applied Probability, Vol. 16, No. 1 (Mar., 1979), pp. 154-162.

[19]Hypertext markup language (HTML). http://www.w3.org/MarkUp.

[20]Jain, A. K., Murty, M. N., and Flynn, P. J. 1999. Data clustering: a review. ACM Comput. Surv. 31, 3 (Sep. 1999), 264-323.

[21]Richard M. Karp. Reducibility Among Combinatorial Problems. In Complexity of Computer Computations, Proc. Sympos. IBM Thomas J. Watson Res. Center, Yorktown Heights, N.Y.. New York: Plenum, p.85-103. 1972.

[22]C.A. Knoblock, K. Lerman, S. Minton, I. Muslea, Accurately and reliably extracting data from the web: a machine learning approach, IEEE Data Eng. Bull. 23 (4) (2000) 33–41.

[23]A. H. F. Laender, B. A. Ribeiro-Neto, A. S. da Silva and J. S. Teixeira, A Brief Survey of Web Data Extraction Tools.

[24]Stephen W. Liddle, David W. Embley, Del T. Scott, Sai Ho Yau, Extracting Data behind Web Forms, Advanced Conceptual Modeling Techniques 2002.

[25]Lingpipe. http://www.alias-i.com/lingpipe/

[26]Lipton, R. J., Naughton, J. F., and Schneider, D. A. 1990. Practical selectivity estimation through adaptive sampling. In Proceedings of the 1990 ACM SIGMOD international Conference on Management of Data (Atlantic City, New Jersey, United States, May 23 - 26, 1990). SIGMOD '90. ACM Press, New York, NY, 1-11. DOI= http://doi.acm.org/10.1145/93597.93611.

[27]Jianguo Lu, Combinatorial Models for Extracting Data from Unranked and Ranked Textual Deep Web, submitted.

[28]Matias, Y., Vitter, J. S., and Wang, M. 1998. Wavelet-based histograms for selectivity estimation. In Proceedings of the 1998 ACM SIGMOD international Conference on Management of Data (Seattle, Washington, United States, June 01 - 04, 1998). A. Tiwary and M. Franklin, Eds. SIGMOD '98. ACM Press, New York, NY, 448-459. DOI= http://doi.acm.org/10.1145/276304.276344.

[29]Alexandros Ntoulas, Petros Zerfos, and Junghoo Cho (2005). "Downloading Textual Hidden Web Content Through Keyword Queries". In Proceedings of the Joint Conference on Digital Libraries (JCDL), 100-109

[30]Helen J. Peat, Peter Willett, The limitations of term co-occurrence data for query expansion in document retrieval systems.

[31]S. Raghavan, H. Garcia-Molina. Crawling the Hidden Web, in: Proc. of the 27th Int. Conf. on Very Large Data Bases (VLDB 2001), September 2001.

[32]E. Rasmussen, Clustering algorithms. In W.B. Franks and R. Baeza-Yates, editors, Information Retrieval: Data Structure and Algorithms, Chap. 16. Prentice Hall, 1992.

[33]Denis Shestakov, Sourav S. Bhowmick and Ee-Peng Lim, DEQUE: querying the deep web,

[34]Steven S. Skiena, The Algorithm Design Manual, Springer-Verlag, New York., 1997.

[35]Text Categorization Corpora. http://ai-nlp.info.uniroma2.it/moschitti/corpora.htm.

[36]Web service definition language (WSDL). http://www.w3.org/TR/wsdl.html.

[37]P. Willett, Recent trends in hierarchic document clustering: A critical review. Information Processing and Management, 24(5), 1988.

[38]Ping Wu, Ji-Rong Wen, Huan Liu, Wei-Ying Ma, "Query Selection Techniques for Efficient Crawling of Structured Web Sources," ICDE, p. 47, 2006.

[39]Xu, J. and Croft, W. B. 1996. Query expansion using local and global document analysis. In Proceedings of the 19th Annual international ACM SIGIR Conference on Research and Development in information Retrieval (Zurich, Switzerland, August 18 - 22, 1996). SIGIR '96. ACM Press, New York, NY, 4-11.

[40]Yasuhiro Yamada, Testbed for Information Extraction from Deep Web.

[41]Yao,S.B., 1977, Approximating Block Access in Database Organizations, ACM Comm., v.20, No. 4, pp. 260-261

[42]Yijun Yu, Jianguo Lu, Juan Fernandez-Ramil, Phil Yuan, Comparing Web Services with Other Software Components, accepted by IEEE ICWS, 2007.

[43]G. Zipf. Human Behaviour and the Principle of Least Effort. Addison-Wesley, 1949.

APPENDIX

The 150 keywords and their estimated returns generated from cluster method for 20NG data.

Keyword	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*
include	124	state	257	wrong	216	religion	119	news	233
space	133	real	245	big	198	lot	260	place	199
sun	119	isn	209	net	202	ll	342	cc	135
number	235	email	189	small	162	love	114	wouldn	150
line	200	high	234	ca	308	car	143	put	255
send	156	today	154	times	187	claim	131	key	137
mail	308	de	139	making	152	access	127	bad	196
simply	136	question	296	group	199	problem	368	ve	480
order	202	software	189	works	149	note	154	pc	120
bit	241	called	216	feel	134	thing	329	mike	120
uk	136	interested	193	add	116	fax	156	work	402
available	195	man	196	won	200	give	288	care	136
means	162	st	134	fact	276	steve	114	play	129
general	157	heard	194	free	178	book	135	systems	157
full	134	single	125	opinions	183	large	153	john	189
cs	248	told	155	power	209	interesting	111	current	123
year	253	haven	133	deal	127	sort	133	support	199
years	321	hard	219	left	158	remember	191	windows	180
word	145	control	144	nice	119	read	304	long	301
david	223	local	142	drive	170	running	152	example	189
computer	195	things	330	doesn	319	makes	184	similar	107
subject	148	mark	141	change	163	hand	150	original	123
standard	166	government	228	open	117	michael	111		
thought	191	answer	138	start	186	mind	161		
address	126	reason	214	life	181	hear	116		
idea	179	understand	135	found	210	fine	123		
pretty	170	person	188	case	250	days	147		
correct	110	b	270	list	160	area	140		
day	244	system	335	money	147	point	367		
based	186	run	196	guess	148	world	255		
law	163	kind	204	back	344	couple	120		
bill	152	stop	130	live	136	matter	159		

The 150 keywords and their estimated returns generated from Set Covering method for 20NG data.

Keyword	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*	*Keyword*	*Est. Return*
sciences	17	rear	25	present	89	works	149	kidding	8
lisa	4	plugs	10	mission	33	graphics	88	usr	15
specific	75	craig	29	dwarner	3	mel	3	lean	9
motorcycles	13	mormons	17	patrick	26	larry	34	showing	41
bus	57	women	88	iii	40	sandvik	22	win	101
board	81	gateway	32	band	28	usual	40	didn	243
firm	19	human	123	ati	22	total	72	idiots	5
interior	11	times	187	argic	22	fbi	78	favorite	26
middle	66	suite	12	major	115	thomas	63	years	321
including	123	convert	40	quiet	20	bits	35	news	233
vcr	8	ground	45	forget	85	decvax	5	includes	62
hour	35	water	54	change	163	posted	94	talking	119
comp	48	due	107	month	65	police	70	life	181
circuit	22	dale	20	prevent	59	charge	46	clear	109
mentioned	92	means	162	russia	26	door	48	yesterday	34
send	156	uni	27	logo	10	happy	72	buy	134
records	28	turbo	25	francisco	26	james	81	inn	11
chain	24	south	48	bat	17	modern	49	server	48
reference	81	finger	16	boy	36	ending	10	davidian	18
grab	18	expressed	54	states	102	relative	23	compatible	32
scrolling	3	don	883	religious	107	greatly	61	giving	72
disk	76	give	288	danny	11	covington	6	site	40
love	114	thread	60	reply	87	peace	65		
wondering	71	sunos	22	solntze	21	bell	51		
wife	61	australia	35	program	160	sunlight	7		
pen	12	physical	40	interest	89	bikes	20		
capable	32	great	232	ball	32	money	147		
fast	103	playing	51	european	34	michigan	20		
recently	92	decide	49	stand	68	receive	30		
mac	82	dave	93	stuff	171	widget	18		
double	41	period	57	create	62	conlon	9		
cases	60	made	289	sick	40	responses	39		

The 150 keywords and their estimated returns generated from Set Packing method for 20NG data.

Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return
reasonable	87	lower	56	image	65	disease	28	large	153
deaths	21	correct	110	wednesday	20	freedom	70	title	50
arms	60	electronic	31	charges	24	cities	21	bodies	22
interest	89	screen	68	road	75	united	65	rain	23
perfectly	35	period	57	granted	31	catch	29	accident	27
house	108	met	30	tested	32	ohio	33	matt	25
stop	130	confused	26	complex	28	christ	73	decide	49
war	89	color	88	cool	33	fax	156	maximum	24
stolen	26	machine	123	opposition	21	knowledge	88	friends	63
statements	44	proposed	37	sp	26	equally	31	age	40
creating	29	archive	23	treat	23	destruction	23	wustl	24
empire	20	discussions	26	noise	36	experts	23	couldn	79
cases	60	response	76	document	28	tells	36	washington	96
brother	36	sitting	33	greatly	61	sunday	21	arizona	22
resource	24	inch	20	interested	193	tank	29	select	20
auto	40	established	41	goal	44	dropped	20	ridiculous	25
george	51	played	61	b	270	theory	60	congress	50
switch	40	making	152	referring	32	connector	20	listen	41
sys	22	berkeley	34	hour	35	rape	24	processor	27
hundred	39	pay	131	returned	22	key	137	kevin	32
stands	24	date	84	station	48	western	58	forced	43
davidians	36	passes	26	columbia	34	posts	62	water	54
features	36	hill	31	production	23	official	55		
stored	26	manner	33	jones	28	si	22		
wpd	23	story	81	supply	48	sweden	22		
equipment	51	mouth	25	experienced	27	ch	33		
count	43	prime	23	numerous	22	box	125		
love	114	th	108	FALSE	59	astronomy	22		
detroit	28	playing	51	reply	87	ucs	30		
replaced	43	calling	40	differences	29	loaded	26		
thousand	27	hurt	32	debate	44	examples	39		
product	58	dick	22	lock	24	responses	39		

The 150 keywords and their estimated returns generated from Random method for 20NG data.

Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return	Keyword	Est. Return
build	65	figure	82	continue	50	state	257	facts	50
tank	29	brand	35	amendment	34	meet	45	si	22
standard	166	issues	63	examples	39	cost	114	behavior	47
business	87	named	29	wouldn	150	simply	136	texas	58
supported	31	access	127	specs	23	persons	25	rear	25
quote	49	stop	130	burn	28	order	202	brian	72
pain	44	eliminate	22	director	20	jesus	91	selling	43
cash	20	sun	119	window	96	bush	23	reduce	31
tom	62	losing	28	design	72	incident	30	stupid	61
written	82	property	32	battery	28	lee	35	british	44
eff	26	email	189	tells	36	reflect	30	manner	33
authority	35	slot	21	thing	329	telephone	38	traffic	35
wings	38	details	61	controlled	22	period	57	industry	29
electronics	33	till	29	gordon	34	experience	88	passed	34
ma	62	london	23	stanford	49	jonathan	24	ram	60
expressed	54	greater	45	mchp	23	secure	47	killed	70
budget	31	adding	28	tested	32	northern	20	fail	27
city	89	simms	25	worry	40	fix	29	cc	135
exactly	129	ended	24	base	67	offered	37	empire	20
days	147	columbia	34	crime	69	tree	22	guaranteed	21
discussion	83	labs	22	straight	35	items	34	turns	33
vancouver	23	users	54	kent	32	description	35	water	54
chain	24	half	69	es	24	sources	53		
annoying	20	joe	58	umd	30	cable	33		
utility	22	installed	50	origin	25	management	26		
admit	46	bearing	20	quick	53	ha	21		
define	42	produced	27	approach	60	message	126		
male	27	surely	40	touch	31	compatible	32		
saved	20	doc	20	wave	24	cities	21		
brother	36	sad	33	purdue	31	paid	49		
degree	29	experienced	27	ridiculous	25	systems	157		
respond	44	wanted	111	ready	40	matter	159		

VITA AUCTORIS

Mr. Xiaolei Yuan was born in 1982 in Shanghai, China.

He started studying Computer Science in University of Toronto in 2000 and obtained his Bachelor of Science degree with distinction in Computer Science in 2004. Later, he was enrolled as master student at School of Computer Science, University of Windsor. He studied there for the next two and half years under Dr. Jianguo Lu's supervision.

Personal Information:
Name: Xiaolei (Phil) Yuan

Email: yuanf@uwindsor.ca