2010

# Crawling Deep Web using a GA-based set covering algorithm

Shaohua Wang
*University of Windsor*

# Crawling Deep Web Using a GA-based Set Covering Algorithm

By

Shaohua Wang

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
2010

*Your file Votre référence*
*ISBN:* 978-0-494-62753-2
*Our file Notre référence*
*ISBN:* 978-0-494-62753-2

## Author's 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 anyone's 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

An ever-increasing amount of information on the web today is available only through search interfaces: the users have to type in a set of keywords in a search form in order to access the pages from certain web sites. These pages are often referred to as the Hidden Web or the Deep Web. According to recent studies, the content provided by hidden web sites is often of very high quality and can be extremely valuable to many users. This calls for deep web crawlers to excavate the data so that they can be reused, indexed, and searched upon in an integrated environment.

Crawling deep web is the process of collecting data from search interfaces by issuing queries. It often requires the selection of an appropriate set of queries so that they can cover most of the documents in the data source with low cost. This can be modeled as a set covering problem which has been extensively studied in graph theory. The conventional set covering algorithms, however, do not work well when applied to deep web crawling due to various special features of this application domain. Typically, most set covering algorithms do not take into account the distribution of the elements being covered. For deep web crawling, the sizes of the documents and the document frequency of the queries follow the power law distribution.

A new GA-based algorithm is introduced in this thesis. It targets at deep web crawling of a database with this power law distribution. The experiment shows that it outperforms the straightforward *greedy algorithm* previously introduced to the literature.

Key words: deep web, deep web crawling, set covering, genetic algorithm.

## Dedication

To my great parents. I love you.

## Acknowledgements

My thanks and appreciation to Dr. Jessica Chen for persevering with me as my advisor throughout the time it took me to complete this research and write the thesis, and for her guidance on my research and during the course of my graduate study which is the most important experience in my life. The other important person who gave me many great ideas on my thesis work is Dr. Jianguo Lu. Without his great advice, I cannot finish my thesis work.

I am grateful as well to Dr. Zhengzhong Ma to be my external reader.

I am grateful too for the support and advice from Yan Wang who is a PHD student conducting research related to mine with my advisor. I can not finish my thesis work so quickly without his help.

# Table of contents

# List of Tables

## List of Figures

## 1. Introduction

Today's search engines do not reach most of the data on the Internet– The Web has been rapidly "deepened" by massive databases online [1]: While the *surface Web* has linked billions of static HTML pages, it is believed that a far more significant amount of information is "hidden". It is behind the search interface of searchable databases. Such information may not be accessible through static URL links–They are assembled into Web pages as responses to queries submitted through the surface interface of an underlying database. In other words, they can only be accessed from surface interface. These pages are often referred to as the *Deep Web* [2] or the *Hidden Web* [3]. Figure 1 shows what a search interface looks like. It is a search interface given by a Windsor library system. Figure 2 shows the hidden information returned by the system as a response to this interface.

Figure 1 Search interface (query form) of Windsor library data base

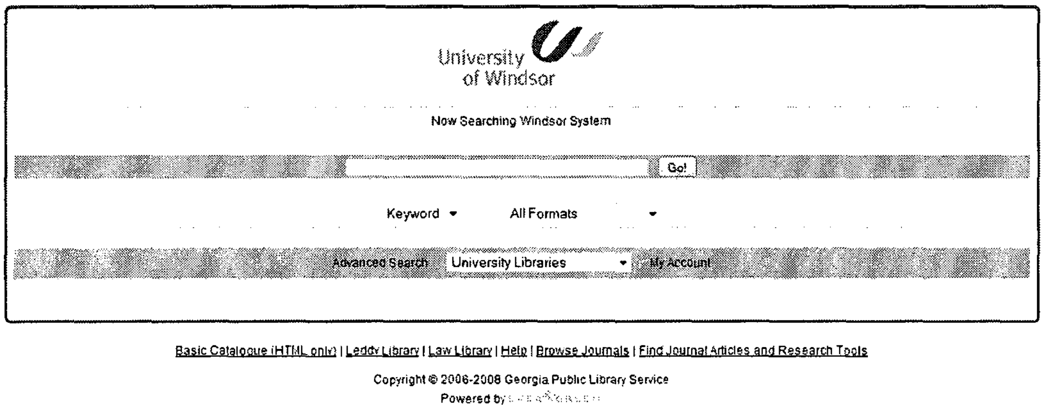
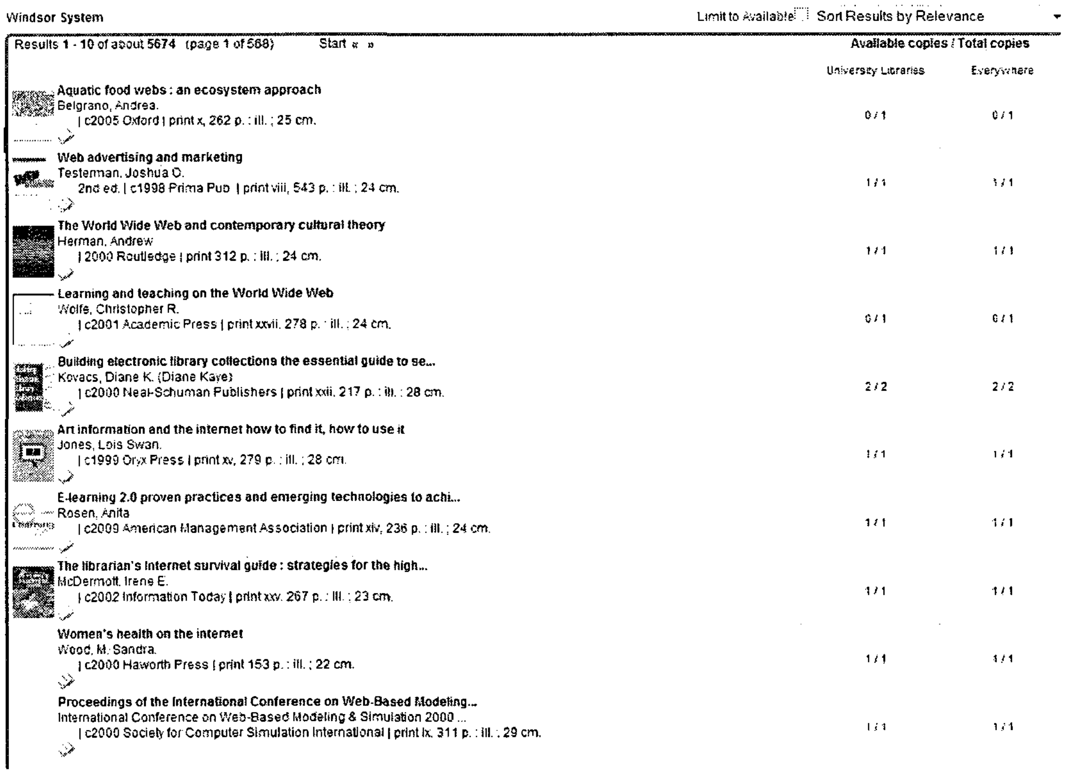

University of Windsor
Now Searching Windsor System
Go!
Keyword
All Formats
Advanced Search
University Libraries
My Account
Basic Catalogue (HTML only) | Leddy Library | Law Library | Help | Browse Journals | Find Journal Articles and Research Tools
Copyright © 2006-2008 Georgia Public Library Service
Powered by

Figure 2 The "hidden" information returned by Windsor library through the search Interface in Figure 1

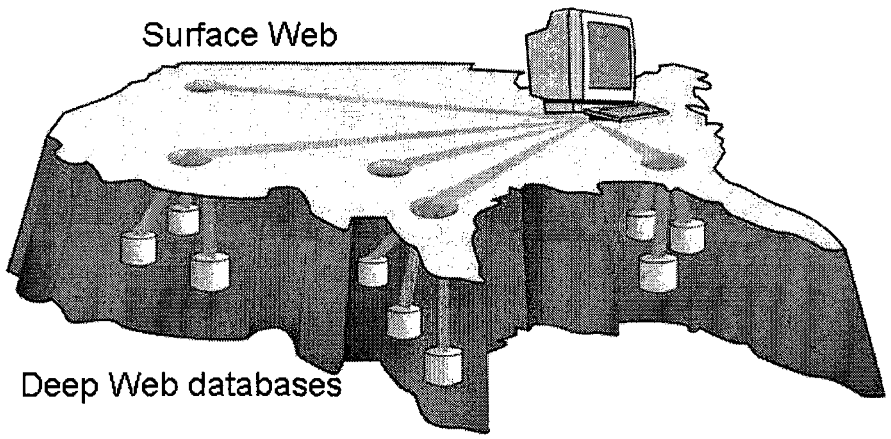


Windsor System | Limit to Available | Sort Results by Relevance

Results 1 - 10 of about 5674 (page 1 of 568) Start « »

| | Available copies / Total copies | |
|---|---|---|
| | University Libraries | Everywhere |
| **Aquatic food webs : an ecosystem approach**<br>Belgrano, Andrea.<br>\| c2005 Oxford \| print x, 262 p. : ill. ; 25 cm. | 0 / 1 | 0 / 1 |
| **Web advertising and marketing**<br>Testerman, Joshua O.<br>2nd ed. \| c1998 Prima Pub \| print viii, 543 p. : ill. ; 24 cm. | 1 / 1 | 1 / 1 |
| **The World Wide Web and contemporary cultural theory**<br>Herman, Andrew<br>\| 2000 Routledge \| print 312 p. : ill. ; 24 cm. | 1 / 1 | 1 / 1 |
| **Learning and teaching on the World Wide Web**<br>Wolfe, Christopher R.<br>\| c2001 Academic Press \| print xxvii, 278 p. : ill. ; 24 cm. | 0 / 1 | 0 / 1 |
| **Building electronic library collections the essential guide to se...**<br>Kovacs, Diane K. (Diane Kaye)<br>\| c2000 Neal-Schuman Publishers \| print xxii, 217 p. : ill. : 28 cm. | 2 / 2 | 2 / 2 |
| **Art information and the internet how to find it, how to use it**<br>Jones, Lois Swan.<br>\| c1999 Oryx Press \| print xv, 279 p. : ill. ; 28 cm. | 1 / 1 | 1 / 1 |
| **E-learning 2.0 proven practices and emerging technologies to achi...**<br>Rosen, Anita<br>\| c2009 American Management Association \| print xiv, 236 p. : ill. ; 24 cm. | 1 / 1 | 1 / 1 |
| **The librarian's Internet survival guide : strategies for the high...**<br>McDermott, Irene E.<br>\| c2002 Information Today \| print xxv, 267 p. : ill. ; 23 cm. | 1 / 1 | 1 / 1 |
| **Women's health on the internet**<br>Wood, M. Sandra.<br>\| c2000 Haworth Press \| print 153 p. : ill. ; 22 cm. | 1 / 1 | 1 / 1 |
| **Proceedings of the International Conference on Web-Based Modeling...**<br>International Conference on Web-Based Modeling & Simulation 2000 ...<br>\| c2000 Society for Computer Simulation International \| print ix, 311 p. : ill. ; 29 cm. | 1 / 1 | 1 / 1 |

Figure 3 The relationship between surface web and deep web



Source of Figure 3: http://www.er.doe.gov/News_Information/News_Room/2009/Aug%204.html

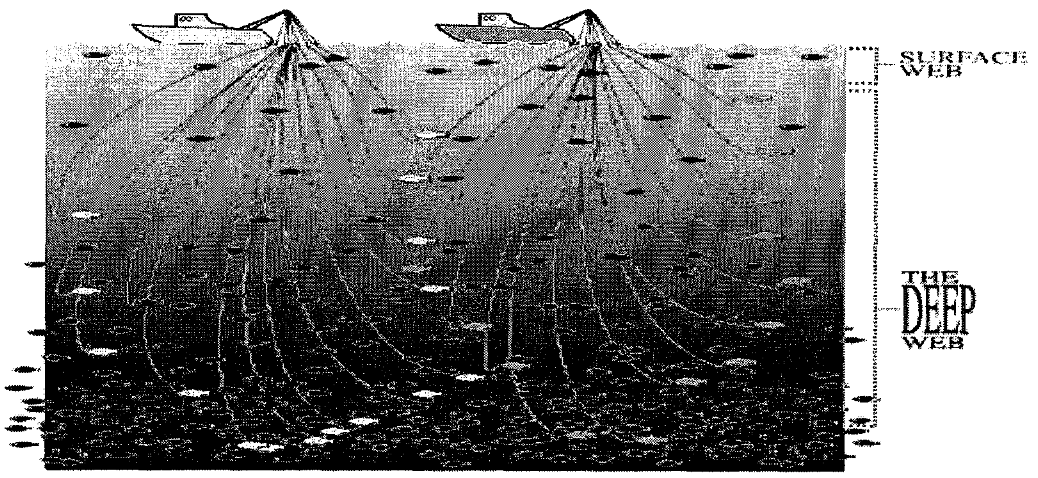
The "*hidden*" information is contained in Deep Web databases which can only be accessed through the search interface. Figure 3 illustrates the relationship between the surface web and the deep web.

According to many studies (e.g. [2]), the size of the deep web increases rapidly as more organizations put their publicly available information online through an easy-to-use search interface [1]. In [2], Chang et al. estimated that well over 100,000 deep web databases currently exist on the Web. Moreover, the content provided by many deep web databases is often of very high quality and can be extremely valuable to many users [1]. This calls for deep web crawling to excavate the data so that the "*hidden*" information can be reused, indexed, and searched.

Crawling deep web [4][5][6][7][8] is the process of collecting "*hidden*" information by issuing queries to *deep web* databases through various search interfaces including HTML forms, web services and programmable web APIs (Application Programming Interfaces). Here a *web service* refers to the API that can be accessed over a network, such as the Internet, and executed on a remote system hosting the requested services. *Web API*, when used in the context of web development, is typically a defined set of Hypertext Transfer Protocol (HTTP) request messages along with a definition of the structure of response messages, usually expressed in an Extensible Markup Language (XML), or JavaScript Object Notation (JSON) format. XML is a language to define a set of rules for encoding documents electronically. JSON is a lightweight computer data interchange format.

Figure 4 below illustrates the process of collecting "*hidden*" information. Here, a boat represents a *query* and the fish represents *a unit of "hidden" information* which can be any kind of data file e.g. a textual file, an audio file in the database. One boat can only get a limit amount of fish: each query can only bring back a set of units of information from databases. Crawling *deep web* is like the process of sending boats to the ocean to do the fishing. Crawling *deep web* is important for various reasons, such as indexing *deep web pages.*

Figure 4 The process of collecting hidden data by issuing queries



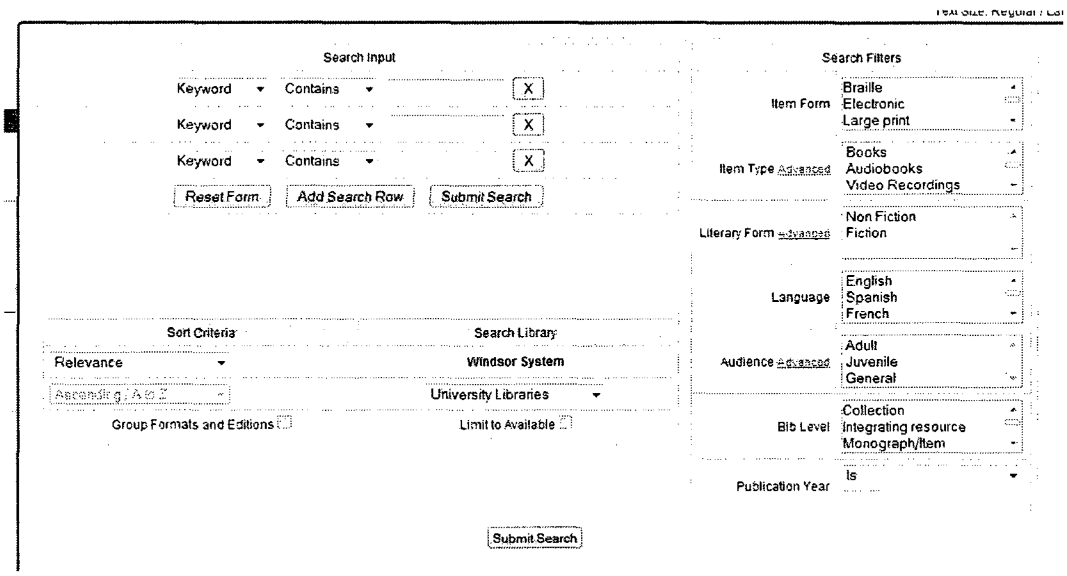
Source of Figure 4: http://www.mkbergman.com/category/deep-web/

There are two kinds of deep web databases:

- Structured database, such as relational database
- Un-structured database, such as text documents, video files

This thesis focuses on the Un-structured databases, especially textual documents databases, i.e. those data sources that contain plain text documents only. The textual data sources usually provide a simple keyword-based query interface instead of multiple attributes as studied in [8]. Here, *multiple attributes* refers to multiple keywords which can be used to search documents. An example of the query interface with multiple attributes is illustrated in Figure 5 which shows an advanced search interface of the Windsor library system. The search interface in Figure 1 is a basic one of it.

Figure 5. The advanced search interface with more search attributes



No matter what kind of database is used, for deep web crawling there are two research challenges.

- ◆ One is learning and understanding the interface and the returning result so that query submission [17] and data extraction [18] can be automated.
- ◆ Another is selecting an appropriate set of *queries* so that they can be used to download most of the *documents* in the data source with *low communication cost.*

In order to reduce the network traffic, previous work [7] [8] [18] considered to minimize the number of queries issued. Here, the cost is measured by the number of *redundant documents* that are retrieved. This is more applicable to the cases when a large number of the documents need to downloaded, especially when the downloaded documents have much larger size compared to the size of the queries. Thus, the fewer *redundant documents* are returned by sending queries in deep web crawling, the lower cost it is spent.

This thesis work focuses on the latter challenge. There are two difficulties in selecting appropriate queries.

a. First, the selection of an appropriate set of *queries* can usually be modeled as a *set covering problem* which is NP-Complete.
b. Second, the actual corpus is unknown to the crawler beforehand; hence the crawler cannot select the most suitable queries with the global knowledge of the underlying documents inside the database.

Since it is not possible to select queries directly from the entire data source, the selection are made from a sample of the database. It is shown with the framework in [9] that queries selected from a sample data source can perform on the total data source as well as on the sample one. Using this sampling method, they solved the difficulty mentioned above in Item b. Furthermore, considering the query selection problem as a set covering problem, they used greedy method [7][9] to select a proper set of queries based on a sample database.

In this thesis work, this framework is adopted and the query selection problem is viewed as set covering problem. However, a new method is proposed to select a proper set of queries. The conventional set covering algorithms, e.g. greedy algorithm

[9], do not work well when applied to deep web crawling due to various special features of this application domain. Typically, most set covering algorithms, do not take the database distribution into consideration. For deep web crawling, the sizes of the documents and the *document frequency of a query*, i.e. the number of documents in the database that contain this query, follow the power law distribution. In order to have a better algorithm for the databases with the power law distribution, a Genetic Algorithm (GA) based heuristic is proposed.

Inspired by evolutionary biology, genetic algorithm is considered as a search technique used in computing to find exact or approximate solutions to optimization and search problems. It has several basic steps such as selection, crossover and mutation. Changes in each step of GA are made and some more steps are added into GA in order to increase its effectiveness in the present setting.

Experiments are conducted on four corpora which are Wiki, Gov, Newsgroup and Reuters in order to compare our method with greedy algorithm and the results show that the proposed method outperforms the straightforward *greedy algorithm*: It can select a more proper set of queries.

This thesis is organized in the following way: Section 2 introduces preliminary knowledge of the present work. Section 3 introduces some related work. It includes one classic GA-based heuristic and three milestone works in deep web crawling.

Section 4 introduces the proposed genetic algorithm. In section 5, the experimental results are summarized. Section 6 concludes the whole thesis.

# 2. Preliminary

In this section, some concepts which are used in our proposed method are introduced.

As mentioned in the introduction, for deep web crawling, there are two research challenges. One is learning and understanding the interface and the returning result so that query submission [17] and data extraction [18] can be automated. The other is selecting appropriate queries so that most of the hidden data are harvested at a *low cost* [7] [8] [19]. For selecting appropriate queries, one of the difficulties is that the actual corpus is unknown to the crawler beforehand; hence the crawler cannot select the most suitable queries with the global knowledge of the underlying documents inside the database.

Lu et al [9] proposed a 4-step framework to face this challenge. The present work is based on this framework.

## 2.1 4-Step framework

In 4-step framework [9], a sample set of documents is first downloaded from the total database. From this sample, a set of queries is acquired from a *query pool of this sample database*, i.e. a set of terms in this sample database. This set of queries is selected in such a way that it can be used to cover most of the documents in the *original* data source with a low cost. The algorithm is as follows:

Algorithm 1: Outline of the DeepWeb Crawling algorithm [9]

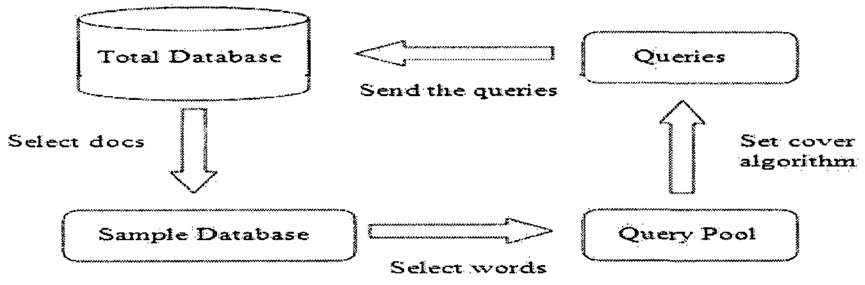
Input: the original data source *TotalDB*; sample size *s*, query pool size *p*.

Output: A set of terms kept in Queries

1. Create a sample data base *SampleDB* by randomly selecting *s* number of documents from the corpus *TotalDB*;
2. Create a query pool *QueryPool* of size *p* from the terms that occur in *SampleDB*;
3. Select a set of queries *Queries* from *QueryPool* that can cover at least 99% of the *SampleDB* by running a set covering algorithm;
4. Mapping the selected queries into *TotalDB*

The structure of this algorithm illustrated in Figure 6

Figure 6 The outline of DeepWeb Crawling algorithm



As it can be seen the third step of the 4-step framework [9] is to select a proper set of queries from query pool. The solution of the 4-step framework [9] for this point is to consider the query selection problem as a *set covering problem*.

## 2.2 Set covering problem

Let Aij be a zero-one matrix with m rows and n columns where $a_{ij}$ denotes the element in row i and column j. Let Cj $=\sum \alpha ij$ denote the cost of column j. The set covering problem (SCP) is the problem of covering all the rows of Aij by a subset of the columns at minimal (total) cost. Let Xj = 1 if column j is in the solution and Xj =0 otherwise. The SCP can be expressed as

$$\text{Minimize} \quad \sum_{j=1}^{n} CjXj \qquad (1)$$

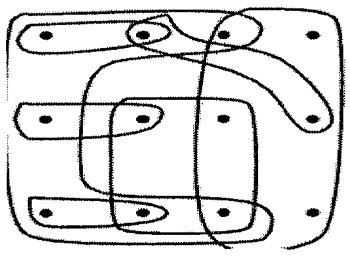
$$\text{Subject to} \quad \sum_{j=1}^{n} \alpha ijXj \geq 1, \qquad i = 1 \ldots m \qquad (2)$$

$$Xj \in \{0,1\}, \qquad j = 1 \ldots n \qquad (3)$$

Subjection (2) ensures that each row is covered by at least one column. If all the costs Cj are equal, the problem is called the *uni-cost* SCP.

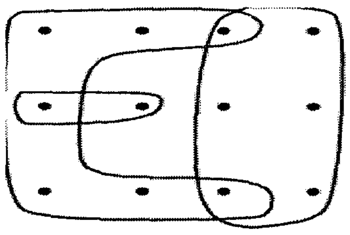
Example 1. In Figure 7, each *circle* covers a set of nodes. A set of circles $\{C_1,\ldots C_m\}$ covers all nodes N ={1,...,n}. What is the smallest number of circles T which can cover all the nodes in this figure? An answer is illustrated in Figure 8. In this example, the cost of each circle is equal. Each circle can be represented as a column; each node can be represented as a row. So the problem can be induced into finding smallest number of columns to cover all the rows.

Figure 7 A set of subsets that cover all the nodes



Source of the picture: http://www.cs.sunysb.edu/~algorith/files/set-cover.shtml

Figure 8 A smallest subset of all subsets of the initial set



Source of the picture: http://www.cs.sunysb.edu/~algorith/files/set-cover.shtml

There are plenty of heuristics to solve set covering problem. The present work proposes a GA-based heuristic within the context of deep web crawling. In the next sub section, the original genetic algorithm is introduced.

## 2.3 Genetic algorithms

A genetic algorithm (GA) can be considered as a search technique having been applied to a variety of combinatorial optimization problems [12]. The theoretical foundations of GAs were originally developed by Holland [13]. The idea of GAs is based on the evolutionary process of biological organisms in nature. During the course of the evolution, natural populations evolve according to the principles of the theory of biological evolution, especially that formulated by Charles Darwin, natural selection and "survival of the fittest". Individuals more successful in adapting to their living environment will have a good chance of surviving and reproducing; while ones less fit will not survive. This means that the genes from individuals with better fit will spread to an increasing number of individuals in each successive generation. The combination of good genes from highly adapted individuals may produce even more fit offspring. In this way, species evolve to become better adapted to their environment.

A genetic algorithm simulates these biological evolutionary processes by taking an initial population of individuals and applying genetic operators, such as crossover and mutation, in each reproduction. Each individual in the population can be represented by a string or chromosome which represents a possible solution to a given problem. The fitness of an individual is evaluated with respect to a given objective function. Highly fit solutions are given more opportunities to reproduce by exchanging pieces of their genetic information, in a crossover procedure, with other highly fit solutions. This produces new child solutions, which share some genes taken from both parents. Mutation is often applied after crossover by altering some gene(s) in the strings. The offspring can either replace the whole population (generational approach) or replace less fit individuals (steady-state approach). This evaluation-selection-reproduction cycle is repeated until a satisfactory solution is found.

Table 1 The GA is based on certain concepts of biological evolution

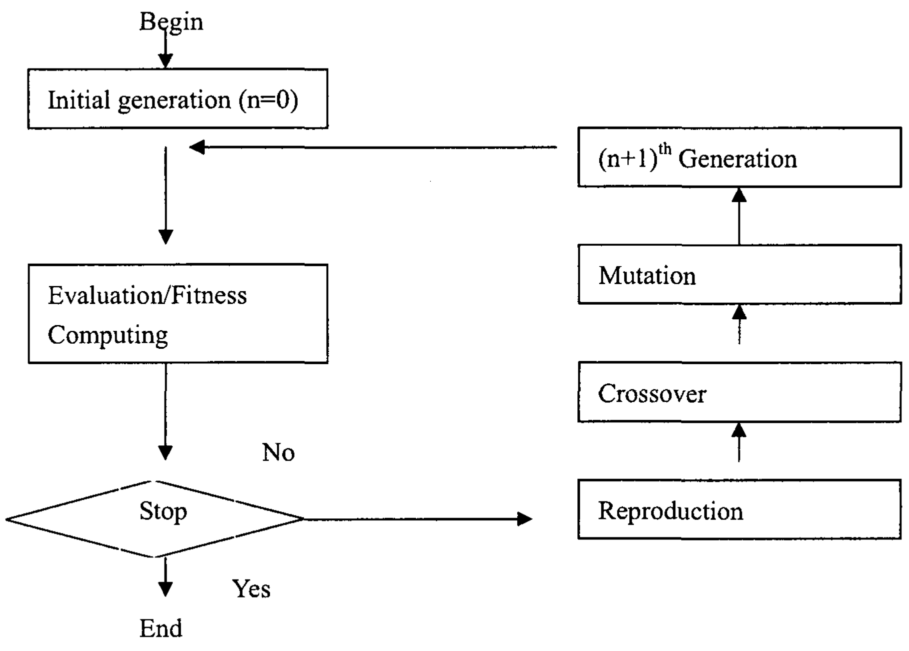
| **Genetic Term** | **Microbiological Definition** | **Model definition** |
|---|---|---|
| Chromosomes | a series of genes carrying genetic information, DNA | Solutions |
| Genes | Section of chromosome carrying trait info. | bits of a solution |
| Crossover | recombine two organisms where genetic information is exchanged | two solutions be exchanged to produce new Offspring |
| Mutation | the elements of DNA are a bit changed | Bit(s) within solutions are changed |
| Fitness | measured by success of the organism in its life | Fitness function f(x) |
| Offspring | New generation after crossover | New population of solutions |

The basic steps of a simple GA are shown below. A more comprehensive overview of GAs can be found in Refs. [12] [14] [15] [16].

```
Generate an initial population;
Evaluate fitness of individuals in the population;
Repeat
        Select parents from the population;
        Recombine (mate) parents to produce children;
        Evaluate fitness of the children;
        Replace some or all of the population by the children;
Until a satisfactory solution has been found;
```

Algorithm flow is illustrated in Fig. 9

Figure 9 Algorithm flow of basic genetic algorithm



As it can be seen, the original GA is a generic algorithm. It can be modified to be applied to some specific problem domain, such as deep web crawling. In order to apply GAs to a particular problem domain, more details such as chromosome representation and fitness function are needed. Such details for the present setting are introduced in the later sections.

## 2.4 Evaluation criteria

The proposed algorithm is partly evaluated in terms of *hit rate* (HR) and *overlapping rate* (OR).

**Definition 5** (Hit Rate, HR) [9]: Given a set of queries Q = {q1, q2,,.., qk} and a database DB. The hit rate of Q on DB, denoted by HR (Q, DB), is defined as the ratio between the number of unique data items collected by sending the queries in Q to DB and the size of the database DB.

$$HR(Q,DB)=\frac{|\bigcup_{p=1}^{k} S(q_p,DB)|}{|DB|} \qquad (4)$$

Here S(q,DB) denotes the set of documents obtained by sending query q to database DB. The numerator of the right hand side of this equation denotes the number of accumulated unique results obtained. |DB| is the size of the target dataset. In reality, this number is usually provided by the data source. In summary, the HR is used to evaluate the quantity of discovery, i.e. how many unique results, out of all the expected results have been collected.

**Definition 6** (Overlapping Rate, OR) [9]: Given a set of queries Q={q1,q2,,...qk}, the overlapping rate of Q on DB, denoted by OR(Q, DB), is defined as the ratio between the total number of collected data items and the number of unique data items retrieved by sending queries in Q to DB.

$$\text{OR (Q, DB)} = \frac{\sum_{p=1}^{k} | S(q_p, DB) |}{| \bigcup_{p=1}^{k} S(q_p, DB) |} \qquad (5)$$

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

# 3. Related work

As mentioned before, two research challenges are in deep web crawling. One is learning and understanding the query interface of database and the returning result. The other one is selecting a proper set [7] [8] [18]. Several pieces of research work [6][21][22][23] have been conducted on learning and understanding the query interfaces by discovering the correspondences among the attributes of the query interfaces in order to obtain a general definition which is a common pattern that accurately describes queries interfaces of deep web database. With the general definition, a scalable solution for a deep web crawling task can automatically find the deep web databases: it is impossible to locate deep web databases manually for deep web crawling. In the mean while, understanding the interfaces is needed for inputting queries into the search interface automatically.

With the popularity of publicly available web services that provide programmable API [20], where input and output data formats are explicitly specified, automated extraction of deep web data becomes more practical, and the problem of query selection is becoming more prominent. Following the framework of [9], a proper set of queries is selected based on a *sample database* and the *query pool* so that the cost of mapping the selected queries into *total database* can be minimized.

Selecting an optimal set of queries can be viewed as a set-covering problem, which is NP-hard. Some pieces of works [7][9][10] reduced the query selection problem into set covering problem. For the set covering problem (SCP) in the context of deep web crawling, each query can be represented by a column; each document can be represented by a row. In the matrix of SCP, the 0-1 values in a column can show which rows can be covered by this column; the 0-1 values in a row can show which columns can cover this row. In the same way, the matrix of SCP can also represent the relationship between queries and documents. Example 2 shows how the query selection problem is reduced into set covering problem.

**Example 2** Table 2 gives a matrix A, where each column represents a query in QueryPool = {q1, q2,..., q5}, and each row represents a document of SampleDB={d1, d2,..., d9}. Cj = $\sum \alpha ij$ is the document frequency (df) of $q_i$. Document frequency of a term $q_i$, as illustrated in Table 2, is the number of documents in the database that contain $q_i$. One selection of the queries is S= {q3, q4, q5}, which can be obtained by the greedy algorithm [9].

Table 2 Matrix A: an input matrix for set covering algorithm

| | **Query 1** | **Query 2** | **Query 3** | **Query 4** | **Query 5** |
|---|---|---|---|---|---|
| **Document 1** | 0 | 0 | 1 | 0 | 0 |
| **Document 2** | 0 | 0 | 1 | 1 | 0 |
| **Document 3** | 1 | 0 | 1 | 0 | 1 |
| **Document 4** | 0 | 0 | 1 | 0 | 1 |
| **Document 5** | 1 | 0 | 0 | 0 | 1 |
| **Document 6** | 1 | 1 | 0 | 1 | 0 |
| **Document 7** | 0 | 0 | 0 | 1 | 0 |
| **Document 8** | 1 | 1 | 0 | 0 | 1 |
| **Document 9** | 0 | 0 | 1 | 1 | 1 |

Several pieces of work viewed the query selection problem of deep web crawling as set covering problem, but they proposed different heuristics to solve the set covering problem in the context of deep web crawling.

In [7], the deep web extraction process is considered as a set covering problem and the adaptive data extraction method has been proposed. The adaptive data extraction method is a greedy approach that attempts to maximize the potential gain of the next query issued to the data set. In the each iteration of the method, a new query is chosen based on its "Estimate Efficiency" among all query 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* document that can be returned by sending $q_i$, and the denominator is the cost of issuing $q_i$, i.e. *redundant documents* returned by sending $q_i$.

One of the main shortcomings in [7] is that when a same query selected by the greedy approach can harvest a large chunk of new returns, it may also bring back many duplicates. In terms of network traffic, duplicated data far overweight the number of queries.

To overcome this shortcoming, the work of [9] and [10] focused on reducing the duplicates.

In [9], J.Lu et al considered the query selection problem as well-known set covering problem which has been extensively studied. They adopted the greedy algorithm in [7] to select queries.

The greedy algorithm used in [7] [9] selects the most cost-effective query in each step. Let Q be a set of queries already selected. According to these simple greedy algorithms, the next query q is selected to cover as many as possible new documents (i.e. documents not covered by any query in Q) per unit cost. The cost is in terms of the *document frequency* df. In other words, q is selected to maximize the value of *new/df* where new is the number of documents covered by q but not by any query in Q.

As an improvement of these simple greedy algorithms [7][9], Wang et al [10] introduced a new approach to select queries. They introduced the concepts of weights into the straightforward greedy algorithm [7][9] and proposed weighted greedy algorithm. If a document can only be matched by one query, apparently that query must be included into Q. In general, when selecting a query, more attention should be paid to cover *small documents* i.e. those containing fewer terms, since usually they can be matched by only very few queries. A weight is assigned to each document, where *small documents* have larger weights. With this intuition, the weight of a document is introduced:

**Definition 1** (Document Weight, dw) [10]: Let sample database D = {d1, d2, … ,dm} and query pool QP = {q1,q2,..., qn}. Each document is considered as a set of terms. Notation $q_j \in d_i$ is used to indicate that a term $q_j$ occurs in document $d_i$. The weight of a document with respect to QP and $d_i (1 \le i \le m)$, denoted by $dw_{d_i}^{QP}$ (or dw for short), is the inverse of the number of terms in QP that occurs in document $d_i$, i.e.

$$dw_{d_i}^{QP} = \frac{1}{|d_i \cap QP|} \qquad (6)$$

**Definition 2** (Query Weight, qw) [10]: The weight of a query $q_j$ $(1 \le j \le n)$ in QP with respect to D, denoted by $qw_{q_j}^{QP}$ (or qw for short), is the sum of the document weights of all documents containing term $q_j$, i. e. ,

$$qw_{q_j}^{QP} = \sum_{\forall d_i \in D, s.t. q_j \in d_i} dw_{d_i}^{QP} \qquad (7)$$

As for weighted greedy strategy, queries $q_j$ with larger number of qw are preferred. However, a larger number of qw should be obtained by fewer number of dw. Fewer number of dw means fewer df. The relationship between qw and dw is shown in Equation (7). In order to harvest more documents with less cost, fewer df is used to obtain larger qw. Therefore queries with smaller df/qw are preferred because, compared with queries having large *df/qw*, they usually lead to lower overlap. The example of the relationships among dw, qw and df is explained in Example 3 below.

**Example 3** Based on the matrix in Table 2, the weight of the documents are shown in the top part of Table 3. The document frequency, the weights of the queries, and their quotient are listed at the bottom of the table. For example, the document weight of d1 is one, the document weight of d2 is ½, and the query weight of q2 is the sum of the weight of the documents that is covered by q2, i.e., 0.66. The query weight of q1 is 1.49 which is the sum of the document weight of d3, d5, d6 and d8. The value of df of q1 is the number of non-zero weights of the documents which are covered by q1.

Table 3 Matrix B: the initial weight table of the example corresponding to Matrix A

| | **Query 1** | **Query 2** | **Query 3** | **Query 4** | **Query 5** | **Doc weight** |
|---|---|---|---|---|---|---|
| **Document 1** | 0 | 0 | 1 | 0 | 0 | 1 |
| **Document 2** | 0 | 0 | 0.5 | 0.5 | 0 | 0.5 |
| **Document 3** | 0.33 | 0 | 0.33 | 0 | 0.33 | 0.33 |
| **Document 4** | 0 | 0 | 0.5 | 0 | 0.5 | 0.5 |
| **Document 5** | 0.5 | 0 | 0 | 0 | 0.5 | 0.5 |
| **Document 6** | 0.33 | 0.33 | 0 | 0.33 | 0 | 0.33 |
| **Document 7** | 0 | 0 | 0 | 1 | 0 | 1 |
| **Document 8** | 0.33 | 0.33 | 0 | 0 | 0.33 | 0.33 |
| **Document 9** | 0 | 0 | 0.33 | 0.33 | 0.33 | 0.33 |
| **df** | 4 | 2 | 5 | 4 | 5 | |
| **qw** | 1.49 | 0.66 | 2.66 | 2.16 | 1.99 | |
| **df/qw** | 2.66 | 3.00 | 1.87 | 1.84 | 2.50 | |

The study of [10] used four corpora which are Gov, Wiki, Newsgroup and Reuters under 4-step framework introduced in Section 2. It showed that

- The relationship between the df and the number of terms follows the power law distribution in the sample databases.
- The relationship between the document size and the number of documents in the sample database follows the power law distribution.

These two relationships follow the power law distribution, which means

- most of the terms are contained by very few documents; while some small df terms are in most of the documents.
- most of the documents only contain few terms.

In order to take into account the database distribution, a new GA-based method is proposed under the framework [9], which means a different method is used to select the queries in order to get fewer duplicates at the third step of the four step framework.

The present GA-based method makes sure that terms with smaller value of df/qw are selected to the solution earlier. In addition, the nature of genetic algorithm can make sure that more possibilities of the combination of genes of solutions can be provided. Therefore in every generation, the present proposed GA may improve the solutions a little bit, which lead to an overall better result.

Since GA is used to solve the query selection problem which is viewed as set covering problem, [24] is also related to the present work. In [24], it has been presented a genetic algorithm-based heuristic for set covering problems, although it is not for a particular application domain such as deep web crawling. The authors in [24] proposed several modifications to the basic genetic procedures including a new fitness-based crossover operator (fusion), a variable mutation rate and a heuristic feasibility operator tailored specifically for set covering problem. The performance of their algorithm was evaluated on a large set of randomly generated problems. Computational results showed that the genetic algorithm-based heuristic is capable of producing better-quality solutions than a number of other heuristics [25] [26] [27].

# 4. Genetic algorithm (GA) -based algorithm

The basic genetic algorithm described in the previous section is refined in a way such that problem-specific knowledge is considered.

## 4.1 Overview of proposed GA-based algorithm

The *framework* of our proposed GA-based algorithm for the deep web crawling SCP, in line with the one introduced in Section 2.3, is given below.

```
1. Generate an initial population of solutions;
2. Repeat
       ♦ Repeat
             i. Select parents from the population using *rank selection*;
            ii. Recombine parents to form children using our proposed *crossover operator*;
           iii. Make children feasible by applying our proposed *feasibility operator* to these that are not feasible;
         Until all the solutions in parent generation are selected to do crossover;
       ♦ Replace some or all of the children population with the same amount of parent population;
       ♦ Mutate each solution in children generation based on mutation rate;
3. Until the satisfaction criteria is met;
```

*Satisfaction criteria* can be defined so that, for example, after 500 generations the algorithm terminates, or if the best solution is not changed within 50 generations. When the algorithm terminates, the best solution is given as the final result of the algorithm. The algorithm is introduced in details in the following sub-sections.

## 4.2 Representation and fitness function

In order to apply GAs to a particular problem domain, the first step in designing a genetic algorithm for a particular problem is to devise a suitable representation scheme for solution space. In the context of deep web crawling, a solution (chromosome) is a set of queries that cover all the documents. Recall that the query selection problem can be reduced into SCP where a column of SCP represents a query and a row of SCP represents a row. Hence, a solution (chromosome) is a set of columns that cover all the rows. Let *n* be the total number of columns in the SCP. $S = \{q_{i1}, q_{i2}, \ldots, q_{im}\}$ is a solution if $i_k \neq i_j$ for any $k, j \in [1,m]$, $k \neq j$; and $i_k \in [1,n]$ for any $k \in [1,m]$.

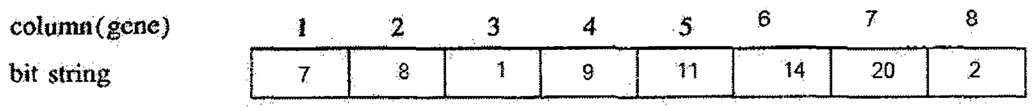
Furthermore, in this thesis work, a list of query ids is used to represent a solution (chromosome), i.e. a set of queries. The list is ordered increasingly by the values of the *df/qw* of the corresponding queries. This is to simplify the GA operations on the solutions: the value of each bit (gene) corresponds to a column id. For example, S = [7, 8, 1, 9, 11, 14, 20, 2] is a solution for a SCP with 20 columns and 10 rows. It is illustrated in Figure 10 as an 8-bit string. Here the value of the 1st bit of solution S is 7, which means the 7th column in deep web crawling SCP is in the solution.

A df/qw value of a query is considered as *small* if it is less than the average df/qw among all queries in the query pool. The queries can then be divided into two groups according to their df/qw values.

1. queries with small df/qw
2. queries with large df/qw

Clearly, queries in group 1 are better than queries in group 2. In the following, when constructing a solution, queries in group 1 will be considered before those in group 2.

Figure 10 Ordered representation of a solution (chromosome)



| column(gene) | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| bit string | 7 | 8 | 1 | 9 | 11 | 14 | 20 | 2 |

The fitness of a solution is directly related to its objective function. It is the sum of the cost of each bit of the solution. It is calculated by

$$fi = \sum_{j=1}^{m} C_j \qquad (8)$$

where Cj is the value of the cost of bit j in terms of document frequency.

## 4.3 Symbol notations used in our proposed algorithm

In this sub-section, some notations are used.

- T0 denotes the initial weight table which contains all the initial *df/qw* for each column (c.f. Table 3).
- OC denotes a list containing all the column ids of T. These column ids are placed in increasing order based on their *df/qw.*
    - $OC_i$ denotes the ith element of OC.
    - OC-S denotes the sub-list of OC containing all the column ids in OC such that the values of *df/qw* of the corresponding columns are smaller than the average.
- S denotes a solution i.e. a list of column ids whose corresponding columns cover all the rows.

- SI denotes a solution candidate: it is a list of column ids whose corresponding columns cannot cover all the rows in the SCP.
- WeightMap denotes a *map* containing the df/qw of some of the columns keyed by their column ids. A key-value pair <k,v> represents column k with its df/qw value v.

In Table 3 (Section 2.3), OC is [4,3,5,1,2]. OC-S of OC is [4,3,5]. S = [4,3,1] is a solution. SI = [4,3] is an infeasible solution. A WeightMap keeps a set of 5 key-value pairs which are {(4,1.8), (3,1.87), (5,2.5), (1,2.66), (2,3.00)}.

## 4.4 Initial population construction

The GA-based algorithm needs an initial set of solutions to do the crossover and mutation. An initial population construction strategy is needed to generate a set of solutions such that

- it has a *moderate* size
- its solutions contain more of the *better queries*

The initial set of solutions should have a moderate size -- not too small and not too big. A too small set will cause the GA-algorithm to terminate too early before reaching a good solution; while a too big set will consume too much computation time with adequate crossover and mutation operations.

The most popular way to generate the initial set of solutions is the r*andom method* which means to obtain a set of solutions by repeatedly running an algorithm that randomly generates a solution. The merit of random method is easy to reach a *moderate size* of the solutions. The drawback is that it is hard to control the number of *better queries* in the initial set of solutions, as the *random method* simply randomly selects queries to construct a solution.

In order to put more of the *better queries* into the initial set of solutions, w*eighted greedy method* can be used, which means to obtain a set of solutions by repeatedly running the *weighted greedy algorithm* [10]. *Better queries* are considered with higher priority because in each step of the *weighted greedy algorithm*, a query with the smallest df/qw is selected into the solution. However, the size of the solutions generated by applying this method is too small due to the occurrences of *identical solutions*. Identical solutions are often generated because greedy algorithms always choose the locally optimal way to proceed, and thus, the queries selected as locally optimal ones very often turn out to be the same. Experiment [10] has been conducted to execute the *weighted greedy algorithm* on the four corpora Wiki, Gov, Reuters and Newsgroup. It was shown there that, by running this *weighted greedy algorithm* 100 times on each of the corpora, the standard deviation of the results are 0, 0, 0, 2, respectively. Note that the standard deviation of the result is 0 means that there is only one solution generated; the standard deviation of the result is 2 implies there are only several solutions generated. Apparently the size of the solutions of running the *weighted greedy method* is too small.

Since the above two methods are not suitable in the present setting, a new algorithm is developed to construct the initial set of solutions Sp. As mentioned before, queries with their df/qw less than the average (Section 3, group 1) are better ones. When constructing the initial set of solutions, in order to increase the chance to get a better solution from the execution of the genetic algorithm, queries in group 1 are used as most as possible. In addition to selecting more of the *better queries*, this method also generates a *moderate size* of solutions. For example, by using present method, a set of 18 solutions for a sample database of Wiki corpora with 500 documents is obtained. Running the *weighted greedy method* on the same data set, on the other hand, gives only one solution.

In the following, the proposed method is introduced in details. Recall that in deep web crawling SCP, a column represents a query. In the following, query and column are used interchangeably.

Suppose that there are totally n queries, while m of them is in OC-S, with their df/qw values less than average. To increase the occurrences of the columns whose *df/qw* is less than the average, together with each of the queries in OC-S, a set of solutions is constructed. For the query indexed by j in OC, solutions are derived from each suffix of OC not containing that of j. A solution is obtained from query j and the shortest prefix of such a suffix. In this way the occurrences of those queries with smaller df/qw are increased. In the algorithm,

- Sp is the current set of solutions

- N is a required number of solutions given as an input
- St is a stack used to keep the record of the indexes of those column ids in OC currently selected into a solution. *column(St)* denotes the list of all column ids whose indexes in OC are in St
- index is a pointer to a column id in OC, which is the head of a suffix of OC currently considered

**Algorithm 2 (initial population construction):**

```
Input: T, N
Output: Sp
Begin:
1. Sp :=empty; St:=empty; j=1;
2. push 1 into St; index=2;
3. while |Sp| < N & j<|OC-S|
      a. while column(St) cannot cover all the rows in T & index<|T|
            i. push index into St;
            ii. index++;
      b. if columns(St) cannot cover all the rows in T & index=|T|
            i. St:= empty;
            ii. j++;
            iii. index =j+1;
            iv. push j into St;
      c. if columns(St) can cover all the rows
            i. add column(St) into Sp;
            ii. if |St|>1& index<|T|
                  1. index =index+1;
                  2. pop last element in St;
            iii. if index=|T|
                  1. pop all elements in St except the first;
                  2. index= value of the first element of St plus one;
            iv. if |St|=1
                  1. break;
4. Return SP;
End
```

Suppose that |OC|=n and |OC-S|=h. For each fixed value of j, the *index* will take each index number of OC starting from j (maximum n of them) and keep increasing from this number to n, without decreasing. So it takes $O(n^2)$ time for each j. Since j has m values, the time complexity of Algorithm 2 is $O(h^* n^2)$.

## 4.5 Parent selection method

*Parent selection* is the task of assigning reproductive opportunities to each solution in the population. There are a number of widely used methods for this. Typically, there are *proportionate selection* and *tournament selection.*

- The *proportionate selection* method calculates the probabilities of individuals being selected as proportional to their fitness. Based on such probabilities, individuals are selected for mating.
- The *tournament selection* method works by forming two pools of individuals, H1 and H2, each consisting of approximately half of the total individuals randomly drawn from the population. Two parents with the smallest fitness values in H1 and H2 respectively are repeatedly taken for mating until at least one of the two pools is empty.

According to our crossover strategy to be explained later on (Section 4.6), if two parent solutions with fitness values *a* and *b* (where a>b) are used for crossover, child solutions will have fitness values no greater than *a*. Thus, if two parent solutions with low fitness values are paired, the child solutions are guaranteed to have low fitness value. With this property, in the present thesis work, a *rank selection* method is proposed. It can be considered as a combination of the above two methods.

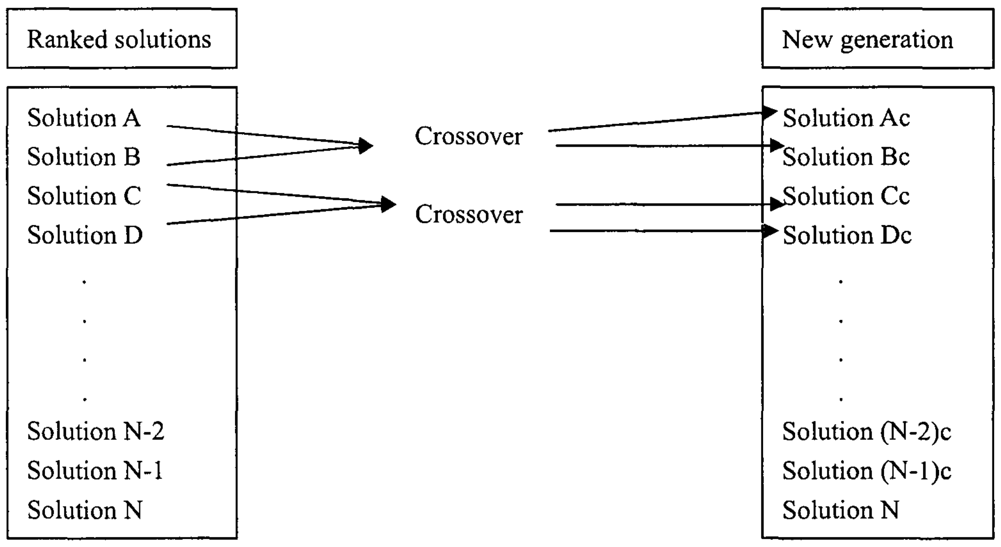
In the present method, all the solutions in the population are ranked in increasing order based on their fitness values. Solutions with smaller fitness values are ranked higher in the list. According to the algorithm, the top two solutions in the ranked list are repeatedly taken to do the crossover, and then put the new solutions generated into new generation pool.

The *proportionate selection* method also used the fitness value to rank all the solutions. However, since it selects the solutions according to the probabilities of individuals being selected as proportional to their fitness, it cannot guarantee to always pair two solutions with low fitness values to do the crossover. Different from this method, ours always pairs two solutions with low fitness values.

The *tournament selection* method also tries to select two solutions with low fitness values to pair. However, the two parents are selected from two separate pools which are grouped randomly. The lowest fitness value in one of the pools may be a large number considering all the fitness values in the two pools. Thus, it cannot guarantee to pair two parents with overall low fitness values.

Figure 11 shows how our method works in one generation. If the total number N of solutions in ranked list is even, there are N/2 pairs of parents; otherwise (if N is odd) there will be (N-1)/2 pairs of parents and the Nth solution will be moved directly into the new generation. After the new generation is obtained, all the solutions are ranked based on *their* fitness value. The new ranked generation will be considered as parent generation in the next generation.

Figure 11 Parent selection method (N is an odd number)



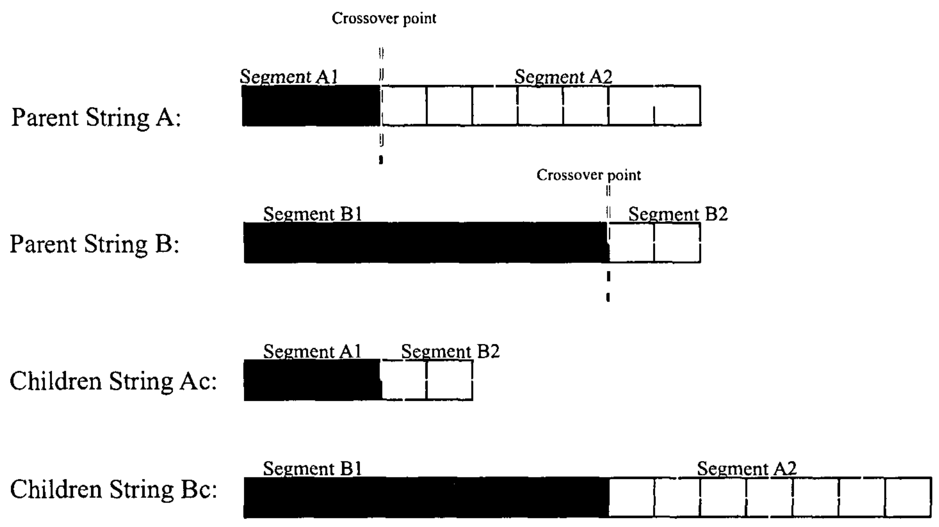
In Figure 11, there are N solutions to be selected to do the crossover. They are ranked based on their fitness value. After the new generation is obtained, all the solutions are ranked based on their fitness value.

## 4.6 Crossover operator

In traditional GA, simple crossover operators such as *cut-and-splice*[29], *one-point*[24], *two-point* [24] crossover are often used. The current method is similar to the cut-and-splice crossover.

With the cut-and-splice approach, each parent string is split into two *segments* by a *crossover point* which is randomly generated. Two parent strings will have separate choices of *crossover point*. The two child strings are obtained by swapping the second segments of the two parent strings. Figure 12 shows how cut-and-splice crossover works. Clearly, the application of the cut-and-splice very often results in a change in the lengths of the children strings.

Figure 12 How cut-and-splice crossover works



No matter which crossover technique is used, the results may not be feasible. A *good crossover point* should decrease the probability of generating solution candidates after crossover. Adopting a *good crossover point* will, thus, avoid handling solution candidates, which increases the running efficiency of the algorithm.

For cut-and-splice crossover, the crossover point for each parent string is chosen randomly, without any restriction. This may lead to some extreme situations. For example, suppose that the *crossover point* for parent strings P=[$P_1$,$P_2$,...,$P_m$] and P'=[$P'_1$,$P'_2$,...,$P'_n$] are 2 and n-1 respectively. The child strings C and C' are C:=[$P_1$,$P_2$,$P'_n$], C':=[$P'_1$, ...,$P'_{n-1}$,$P_3$,...,$P_m$]. As it can be seen, C is very short, and usually it cannot be a (feasible) solution. Similarly, C' is very long usually with a fitness worse than those of the parents. Hence, in these situations, the use of cut-and-splice crossover operator leads to low probability of generating (feasible) solutions.

The following way is proposed to define a *good crossover point*. The present method is similar to the cut-and-splice crossover except that the crossover point for each of the parents is not randomly chosen: It is chosen to make the two segments of the chromosome having approximately the same sums of the document frequency of the queries in each segment. Let Sum1 be the document frequency of segment [$P_1$...$P_k$] of P and Sum2 the sum of the document frequency of the other segment [$P_{k+1}$...$P_m$] of P. If k can make Sum1 and Sum2 almost equal, then k can be considered as a crossover point. In other words, k is considered a crossover point if equation 9 holds for P,

$$\sum_{i=1}^{k} c_i \approx \sum_{j=k+1}^{m} c_j \quad (9)$$

where m is the length of the solution, and Cj is the cost of bit j in terms of document frequency.

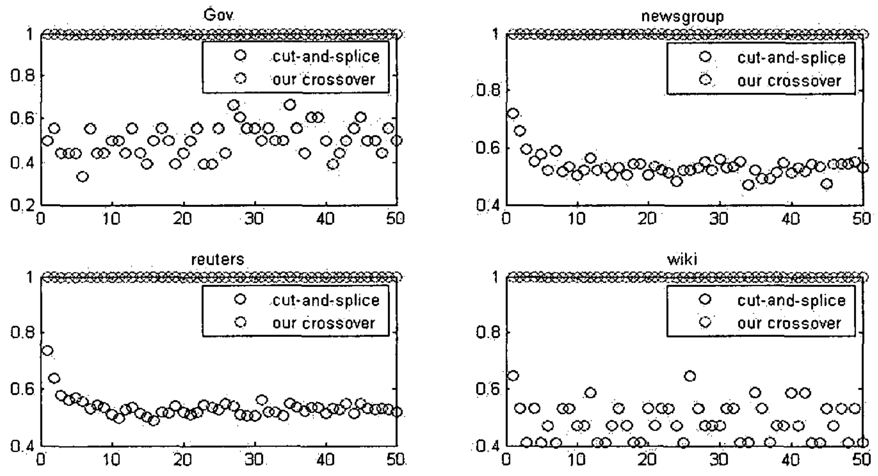
According to the crossover strategy, if two parent solutions with fitness values *a* and *b* (where a>b) are used for crossover, since both two parents are divided into two segments which have almost of the same sum of df, the fitness values of two children are all about a/2+b/2. As b≤a/2+b/2≤a, the child solutions will have fitness values no greater than *a*. This property is used in the parent selection strategy (c.f. Section 4.5).

As mentioned, when combining the two parent chromosomes (solutions), the goal to achieve is that the resulting children have more chance to be feasible. The intuition behind this crossover operator towards this goal can be understood in the following way.

A solution is a list of columns which can cover all the rows. The overall fitness of a solution is determined by the sum of df of each column of the solution. Our crossover point splits each parent into two segments of columns with almost the same sum of dfs, which means each segment can cover at least half of the rows. Therefore, after swapping the segments between two parents, the two new children have good chance to cover all the rows. In this way the probability of generating infeasible solutions is reduced after crossover.

An experiment have been conducted to compare between the cut-and-splice method and the present crossover, the ability of generating (feasible) solutions on four data corpora which are Wiki, Gov, Newsgroup and Reuters. Starting with the same pool of solutions, the next generation is generated using each of the methods; the ratio of the number of (feasible) solutions over all resulting children is calculated and then all the infeasible children are made feasible in each generation. The results from running the four corpora for 50 generations are given in Figure 13.

Figure 13 Experiment results of comparison of cut-and-splice and our method



In Figure 13, the X-axis is the number of generation(s) the GA has run; Y-axis is the ratio of (feasible) solutions over all resulting children in one generation. According to Figure 13 the crossover operator introduced in this thesis can keep all the children feasible within 50 generations, while the cut-and-splice operator can only have about 50% of children feasible on average for the four corpora. Therefore our method outperforms the cut-and-splice crossover operator in terms of keeping children feasible.

In the above experiment, the OR of each solution in the starting pool of the solutions is very high (≈ 4.5), which means the number of documents brought back by sending all the queries in a solution to SampleDB is 4.5 times more than the size of SampleDB.

By using our crossover operator, each of the two parents are split into two segments, each having its OR close to 2.25. Therefore the OR of a child will have much chance to be greater than 1 after recombining two segments from different parents. In addition, the crossover operator does not change a lot the OR of a solution in a generation. This explains why the percentage of the feasible solutions over all the child results for the proposed crossover is high.

Note that although the proposed method can reduce the probability of generating infeasible solutions, there may still be some infeasible solutions generated. There are two ways of dealing with infeasible solutions:

- remove them from the set of children;
- transform them into (feasible) solutions.

In order to obtain more solutions during the crossover operation, the latter approach is chosen. The proposed feasibility operator is described in later sections. This feasibility operator, as well as the mutation algorithm to be introduced later, makes use of the *dynamic calculation of df/qw*. In the following, the *dynamic calculation of df/qw* is introduced.

## 4.7 Dynamic calculation of df/qw

During the procedure of the GA calculation, very often sorting the columns is needed according to their *df/qw* so that those with lower *df/qw* values can be selected first. In Section 3, the definition of *df/qw* is given. The value of df/qw is calculated from a given weight table *T* for each of its columns. Note that in the case that some columns have already been selected, and the rest of the columns would like to be sorted in order to be selected according to their df/qw values, the weight table used for the calculation of df/qw should consist of only those *columns to be selected* -- it should not include those already selected into the solution. When some columns are selected and removed from the set of columns to be considered, those documents covered by these columns may also be taken away from being considered. Note that the *document weights* for the rest of the documents do not change, but the *query weight* of each remaining column may be changed due to the removal of some of the documents from being considered. Finally, while *document frequency (df)* does not change, df/qw may be changed. In general, during the GA execution, the *set of columns to be selected* keeps changing, and thus, the calculation of df/qw becomes *dynamic*.

Example 4. Table 4 (a) shows an initial table T and the df/qw of each column. Based on the df/qw, the columns are sorted in increasing order as [q4, q3, q5, q1, q2]. Now suppose that q4 and q3 have been selected into a solution. As a consequence, documents d1, d2, d3, d4, d6, d7, d9 are covered. The rest will be chosen among columns q1, q2, q5 in order to cover documents d5, d8. The rest of columns {q1, q2, q5} should be sorted in order to select the next column into the solution. Table 4 (b) shows a new table after removing from the initial weight table T columns 3 and 4, and documents d1, d2, d3, d4, d6, d7, d9. Note that in this example, the document weights of d5, d8 do not change; document frequencies of q1, q2, q5 do not change. However, due to the removal of documents d3, d6, the query weight of q1 is changed from 1.49 to 0.83. As a consequence, while df of q1 is still 4, the df/qw of q1 changes from 2.66 to 4.82. Based on the new table, columns q1, q2, q5 are sorted as [q1, q5, q2].

Table 4 Dynamic calculation of df/qw.

Table 4 (a)

| | Query 1 | Query 2 | Query 3 | Query 4 | Query 5 | Doc weight |
|---|---|---|---|---|---|---|
| **Document 1** | 0 | 0 | 1 | 0 | 0 | 1 |
| **Document 2** | 0 | 0 | 0.5 | 0.5 | 0 | 0.5 |
| **Document 3** | 0.33 | 0 | 0.33 | 0 | 0.33 | 0.33 |
| **Document 4** | 0 | 0 | 0.5 | 0 | 0.5 | 0.5 |
| **Document 5** | 0.5 | 0 | 0 | 0 | 0.5 | 0.5 |
| **Document 6** | 0.33 | 0.33 | 0 | 0.33 | 0 | 0.33 |
| **Document 7** | 0 | 0 | 0 | 1 | 0 | 1 |
| **Document 8** | 0.33 | 0.33 | 0 | 0 | 0.33 | 0.33 |
| **Document 9** | 0 | 0 | 0.33 | 0.33 | 0.33 | 0.33 |
| **df** | 4 | 2 | 5 | 4 | 5 | |
| **qw** | 1.49 | 0.66 | 2.66 | 2.16 | 1.99 | |
| **df/qw** | 2.66 | 3.00 | 1.87 | 1.84 | 2.50 | |

Table 4 (b)

| | Query 1 | Query 2 | Query 5 | doc weight |
|---|---|---|---|---|
| **Document 5** | 0.5 | 0 | 0.5 | 0.5 |
| **Document 8** | 0.33 | 0.33 | 0.33 | 0.33 |
| **df** | 4 | 2 | 5 | |
| **qw** | 0.83 | 0.33 | 0.83 | |
| **df/qw** | 4.82 | 6.06 | 6.02 | |

Given a table T and a set Q of columns that appear in T but are selected, the following algorithm calculated the df/qw for all the columns not in Q and returns a sorted list CQP of them.

**Algorithm 3 (sorting with dynamic calculation of df/qw):**
Input: T0, Q

```
Output: CQP
Begin:
    1. WeightMap = empty;
    2. Make a copy of T0 as TC;
    3. **For** each column in Q
        a. Remove all the rows covered by current column from TC;
        b. remove current column itself from TC;
    4. **For** each column in TC
        a. Recalculate *df/qw* for current column;
        b. Put current column's id into WeightMap as a key and its df/qw as its
           value;
    5. Sort WeightMap based on values in increasing order;
    6. Iterate each entry of WeighMap and put entry's key into CQP;
    7. Return CQP;
End.
```

Let m, n be the number of columns and number of documents in T, and k the number of columns in Q. The first for-loop will take O(m*n*k) time, and the second for-loop will take O(m*n) time. The total time complexity of this algorithm is O(m*n*k).

This algorithm will be used in two cases:

- *feasibility operator* which makes a solution candidate feasible by adding more columns. In this case, the inputs of the algorithm 4 are T0 and a solution candidate. The output of algorithm 3 is used by the feasibility operator to select columns.
- *mutation operator* which modifies a solution by randomly removing some column(s) and adding new one(s) to make it feasible again. In this case, Algorithm 3 is used for the second step where its inputs are T0 and a solution candidate. The output of the algorithm is used by mutation operator to select columns to be added into the solution candidate.

## 4.8 Feasibility operator

Here a heuristic operator is proposed to make the solution candidates feasible. It tries to keep the overlapping rate low when adding more columns. In algorithm 4, the inputs are the initial weight table T0 and a solution candidate SI. The output is a solution S.

**Algorithm 4 (feasibility operator algorithm):**

```
Input: T0, SI
Output: S
Begin:
    1. S=SI;
    2. a list of columns CQP= Algorithm 3 (Input: T0, SI);
    3. For each column in CQP
        a. If S cannot cover all the rows in deep web crawling SCP
            i. add current column into S;
        b. If S can cover all the rows
            i. Break;
    4. Return S;
End.
```

As mentioned before, step 2 will take O (m*n*k) time. Step 3.a takes O (m*n) time so the for-loop of step 3 will take O $(m*n^2)$ time. Overall, since $k \leqslant n$, the time complexity of this algorithm is O $(m*n^2)$.

## 4.9 Population replacement model

Usually there are two population replacement models: *incremental replacement* [24] and *generational replacement* [24] model.

Incremental replacement: whenever a new child has been generated, it will be used to replace a randomly chosen member in the population -- usually one with an above-average fitness value. Note that above–average fitness means less fit. This type of replacement method is called incremental replacement or steady-state replacement.

Generational replacement: a new population of children is generated and the whole parent population is replaced by the new population of children. In the experiment, the generational method is used. Note that the time spent on this step is ignorable compared to other steps.

## 4.10 Mutation operator and mutation rules

Mutation is applied to each child after crossover. There are many ways to do the mutation based on different chromosome representations.

◆ With binary representation, the *bit inversion* can be used– select some bits to be inverted. For example, suppose that there is a chromosome whose 0-1 sequence is 11001001. After mutation: 1*1*001001 =>1*0*001001, the second bit was inverted.

◆ With our non-binary representation, the mutation is realized by changing some column id(s) into others that are not present in the current solution (chromosome).

In the present mutation algorithm, first some column(s) are randomly selected to be removed from a solution S, and new column(s) are selected into S to make it a new solution S'. If the fitness value of S' is less than that of S, S is replaced with S' in the population; otherwise S is kept in the population. In the present work, *mutation rate* is the ratio of the number of columns mutated over the number of all columns in a solution. The *mutation rate* is used to calculate the number of columns to be mutated.

The mutation algorithm is described below. The inputs of the algorithm are the initial weight table T0, a solution S and a mutation rate. The output is a solution.

```
Algorithm 5 (Mutation algorithm):
Input: T0, S and mutation rate
Output: S'
Begin:
    1. make a copy of S as TS;
    2. let n=(mutation rate)* |S|;
    3. randomly generate n column ids from TS;
    4. remove those n column ids from TS;
    5. a list of column ids CQP=Algorithm 3 (Input: T0, S);
    6. For each column id in CQP
        a. If TS cannot cover all the rows in SCP
            i. Put current column id into TS;
        b. Else Break;
    7. If the fitness of TS is better than that of S
        a. S'= TS; Return S';
    8. Else Return S;
End.
```

As mentioned before, step 5 will take O $(m*n*k)$ time. Step 6.a takes O $(m*n)$ time so the for-loop of step 8 will take O $(m*n^2)$ time. Overall, $k \leqslant n$, the time complexity of this algorithm is O $(m*n^2)$.

## 4.11 The time complexity of our proposed GA

Recall that n is the number of columns in T0, m is the number of rows in T0. Furthermore, let

- N be the number of solutions in one generation

- ST be the satisfaction criteria which is a number of generations our GA will run

The time complexity of our GA is determined by two parts:

1. generating an initial set of solutions
2. repeatedly do crossover, replacement, mutation

The time complexity of generating an initial set of solutions is $O(n^3)$. The time complexity of item 2 is O(ST*y) where y is the time spent for the following operations on one generation.

i. selecting parents,
ii. performing crossover
iii. applying feasibility operator.

The time complexity of selecting parents can be ignored compared with the other two steps. The time complexity of performing a crossover for a pair of parents is O(n). After crossover, it is needed to check whether a child is feasible or not. The time spent for this check is O(m*n). The time spent on applying feasibility operator to make a solution candidate feasible is $O(m*n^2)$. Thus, the time spent on item 2 is $O(ST*N*m*n^2)$.

Since the time spent on generating an initial set of solutions and for the calculation of the generations are $O(n^3)$ and $O(ST*N*m*n^2)$ respectively, the overall time complexity is $O(ST*N*m*n^2 + n^3)$.

# 5. Experiment

## 5.1 Experiment setting

### 5.1.1 Data setting

The experiments have been run on the same data as that of [9] from four corpora: Reuters, Gov, Wikipedia and Newsgroup. These are the test data used by many researchers e.g. [9][10] in information retrieval. All SampleDBs and Query Pools are generated from our search engine [9]. The size of sampleDBs is 500. The sizes of the Query Pools for the sampleDB of Retuers, the sampleDB of Gov, the sampleDB of Wikipedia and the sampleDB of Newsgroup are 1063, 857,1179,837 respectively.

### 5.1.2 Parameter setting

There are several parameters that will affect the performance of the proposed GA-based method. How they are determined will be explained below.

- *Initial population size*: with the given SampleDBs and *query pools* of Newsgroup, Wikipedia and Gov, the initial population construction algorithm generates at most 168, 17, 18 initial solutions respectively. With the given SampleDBs and *query pools* of Reuters, the initial population construction algorithm generates a lot of solutions and the first 300 solutions generated are taken.
- *Population size*: the population size in the present setting remains the same as the initial population size through out the execution of the algorithm.
- *Mutation rate*: Based on plenty of experiments, it is found that when mutation rate is 0.03, the result is the best for the four corpora.

- Satisfaction criteria: the proposed GA-based method will terminate after having calculated 500 generations.

Furthermore, hit rate (HR) is set to 1, which means all the documents from SampleDB are collected, then compare the results with greedy algorithm [7] [9] in terms of OR.

The parameter setting above is used to conduct an experiment based on four corpora (Wiki, Newsgroup, Gov and Reuters) to compare the performance of our GA method with that of greedy algorithm [7][9]. The following sub-section shows the results.

## 5.2 Experiment result

Table 5 shows the values of OR (Overlapping Rate) and their statistics including minimum (min), maximum (max), average (avg) and SD (standard deviation) collected from running the proposed method (GA) and greedy method (G) 12 times on each SampleDB of the four data corpora. Although all the tests are run with a fixed set of parameters, each time the result will be different because the algorithm involves randomly selections in several steps. Each time the greedy algorithm can produce a different result because the initial query is randomly selected and the locally optimal choices may not be unique.

Table 5 The OR achieved by the proposed method and greedy method for four copora. The mutation rate is 0.03 and the generations are 500

| | Reuters | | Wiki | | Gov | | Newsgroup | |
|---|---|---|---|---|---|---|---|---|
| | G | GA | G | GA | G | GA | G | GA |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Max OR | 2.24 | 1.83 | 2.65 | 2.18 | 3.24 | 2.3 | 2.53 | 1.94 |
| Min OR | 2.18 | 1.69 | 2.59 | 1.95 | 3.12 | 2.19 | 2.36 | 1.73 |
| Avg OR | 2.21 | 1.75 | 2.60 | 2.02 | 3.20 | 2.26 | 2.47 | 1.84 |
| SD | 0.015 | 0.04 | 0.017 | 0.065 | 0.031 | 0.037 | 0.045 | 0.08 |

As explained before, the weighted greedy algorithm [10] outperforms greedy algorithm [7][9] consistently. A similar approach as weighted greedy algorithm was used to construct the initial population. This, together with improvement via crossover and mutation of our GA, greatly increased the chance to achieve a better result than that of greedy algorithm.

Table 5 shows that in general the proposed GA based method outperforms the normal greedy method. In particular,

- Even the maximum OR of the proposed GA based method is less than the minimal OR of the greedy method when HR is 1 on any data corpora.
- On the average the proposed method outperforms the normal greedy method by approximately 20%-30%.

In order to achieve this improvement, our method consumes more time than that of greedy algorithm. Using a PC with CPU 2.0 GHz, RAM 3.5 GB, the time consumed for running greedy algorithm is usually several minutes, while the time consumed for running the proposed GA-based method takes about an hour. Such extra time spent can be ignored when a large amount of documents need to be downloaded. For example, even to download 100,000 amazon links it usually may take 1.5 days. Let alone to download the documents associated by the links.

## 5.3 Parameter consideration

Several parameters may affect the performance of the genetic algorithm. The experiments are carried out to investigate the effects of the *satisfaction criterion* (number of generations computed) and the *mutation rate* on the performance of the algorithm.

In order to see how the *satisfaction criterion* of the proposed GA-based method affects its performance, all parameters of the proposed GA-based method are fixed except the *satisfaction criterion* which is set to those values as listed in Section 5.1.2. The number of generations has been set to be 50, 100, 200, 300, 400, 500, 600, 700, 800 and have run 10 times of each case on all four corpora, Gov, Wiki, Reuters and Newsgroup. The results of overlapping rates of each of the 10 experiments of each case on Gov, Wikipedia, Reuters and newsgroup, as well as the average of the overlapping rate, are listed in Table 6, Table 7, Table 8 and Table 9. In these tables, *Avg* stands for average overlapping rate.

Table 6 The overlapping rate after certain umber of generations in 10 test cases for Gov corpus

| Number of generations | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 2.99 | 2.97 | 3.17 | 3.09 | 3.1 | 3.1 | 3.15 | 3.03 | 3.15 | 3.06 | 3.08 |
| 100 | 2.68 | 2.71 | 2.67 | 2.61 | 2.62 | 2.64 | 2.66 | 2.63 | 2.64 | 2.54 | 2.64 |
| 200 | 2.39 | 2.41 | 2.36 | 2.38 | 2.38 | 2.38 | 2.37 | 2.39 | 2.34 | 2.35 | 2.3 |
| 300 | 2.32 | 2.32 | 2.31 | 2.32 | 2.28 | 2.31 | 2.29 | 2.28 | 2.29 | 2.32 | 2.3 |

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 400 | 2.28 | 2.29 | 2.28 | 2.27 | 2.3 | 2.3 | 2.3 | 2.31 | 2.3 | 2.27 | 2.29 |
| 500 | 2.26 | 2.3 | 2.26 | 2.27 | 2.27 | 2.29 | 2.28 | 2.26 | 2.27 | 2.28 | 2.27 |
| 600 | 2.26 | 2.24 | 2.22 | 2.27 | 2.26 | 2.27 | 2.24 | 2.25 | 2.27 | 2.25 | 2.25 |
| 700 | 2.25 | 2.25 | 2.25 | 2.24 | 2.27 | 2.23 | 2.22 | 2.23 | 2.21 | 2.23 | 2.24 |
| 800 | 2.26 | 2.23 | 2.26 | 2.22 | 2.24 | 2.24 | 2.22 | 2.25 | 2.22 | 2.26 | 2.24 |

Table 7 The overlapping rate after certain umber of generations in 10 test cases for Wikipedia corpus

| Number of generations | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 2.91 | 3.16 | 3.00 | 3.15 | 3.03 | 3.07 | 3.14 | 2.86 | 3.25 | 2.93 | 3.05 |
| 100 | 2.53 | 2.69 | 2.62 | 2.54 | 2.63 | 2.71 | 2.62 | 2.69 | 2.4 | 2.55 | 2.6 |
| 200 | 2.15 | 2.18 | 2.26 | 2.31 | 2.27 | 2.29 | 2.25 | 2.31 | 2.32 | 2.26 | 2.26 |
| 300 | 2.16 | 2.17 | 2.23 | 2.19 | 2.14 | 2.16 | 2.08 | 2.17 | 2.08 | 2.16 | 2.15 |
| 400 | 2.1 | 2.1 | 2.12 | 2.15 | 2.06 | 2.07 | 2.13 | 2.07 | 2.13 | 2.12 | 2.11 |
| 500 | 1.96 | 21.99 | 2.0 | 2.1 | 1.95 | 2.02 | 2.02 | 1.97 | 2.18 | 2.05 | 2.02 |
| 600 | 2.04 | 2.02 | 2.03 | 2.0 | 2.0 | 2.01 | 2.02 | 2.03 | 2.0 | 2.05 | 2.02 |
| 700 | 1.99 | 2.02 | 2 | 2 | 2.01 | 2 | 2.02 | 2 | 2 | 1.98 | 2.01 |
| 800 | 2.06 | 2.02 | 2 | 2 | 2 | 2.02 | 1.99 | 2.25 | 2 | 1.98 | 2.0 |

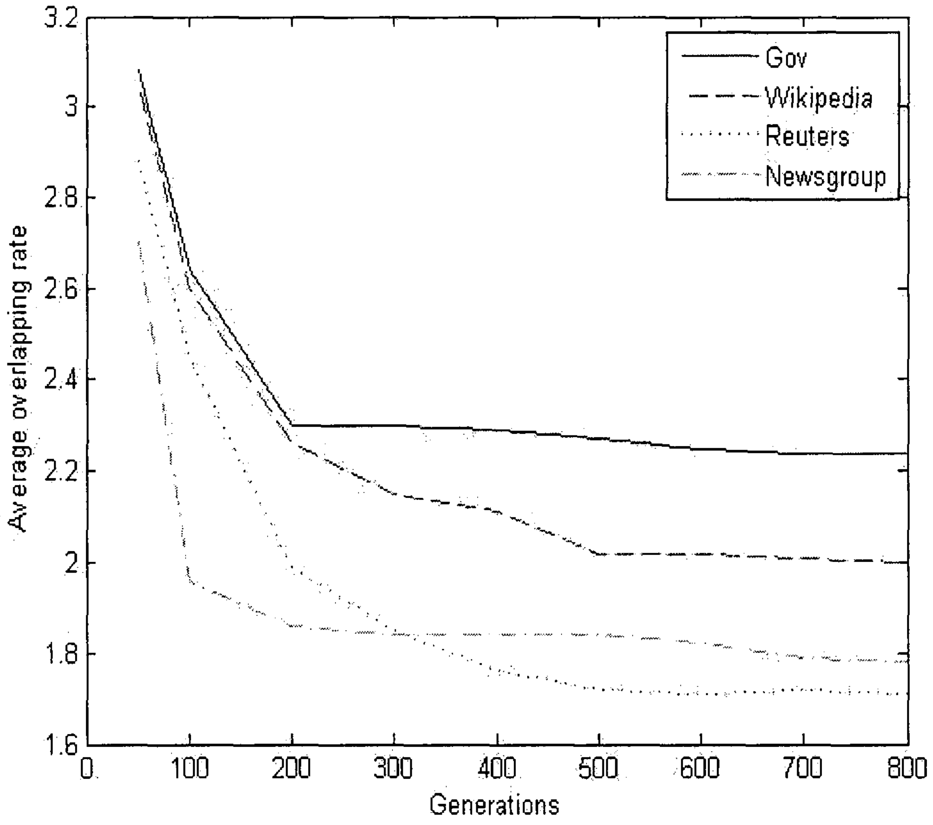
Table 8 The overlapping rate after certain umber of generations in 10 test cases for Reuters corpus

| Number of generations | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 2.79 | 2.94 | 2.95 | 2.94 | 2.99 | 2.75 | 2.84 | 2.7 | 2.91 | 2.96 | 2.88 |
| 100 | 2.49 | 2.4 | 2.55 | 2.54 | 2.32 | 2.46 | 2.36 | 2.56 | 2.38 | 2.43 | 2.45 |
| 200 | 2.01 | 1.9 | 2.03 | 2.01 | 1.96 | 2.06 | 1.94 | 1.92 | 1.99 | 2.02 | 1.99 |
| 300 | 1.89 | 1.82 | 1.87 | 1.82 | 1.84 | 1.88 | 1.86 | 1.84 | 1.9 | 1.82 | 1.85 |
| 400 | 1.8 | 1.7 | 1.72 | 1.76 | 1.74 | 1.73 | 1.76 | 1.77 | 1.81 | 1.77 | 1.76 |
| 500 | 1.7 | 1.74 | 1.72 | 1.73 | 1.76 | 1.72 | 1.73 | 1.71 | 1.72 | 1.7 | 1.72 |
| 600 | 1.7 | 1.66 | 1.7 | 1.71 | 1.69 | 1.76 | 1.74 | 1.75 | 1.72 | 1.7 | 1.71 |
| 700 | 1.68 | 1.68 | 1.67 | 1.75 | 1.75 | 1.68 | 1.68 | 1.78 | 1.75 | 1.72 | 1.72 |
| 800 | 1.78 | 1.76 | 1.67 | 1.74 | 1.77 | 1.68 | 1.67 | 1.7 | 1.7 | 1.67 | 1.71 |

Table 9 The overlapping rate after certain umber of generations in 10 test cases for Newsgroup corpus

| Number of generations | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | Avg |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 50 | 2.73 | 2.45 | 2.6 | 2.54 | 2.86 | 2.75 | 2.89 | 2.67 | 2.58 | 2.89 | 2.7 |
| 100 | 1.98 | 1.96 | 2 | 1.86 | 1.99 | 1.89 | 2.01 | 2.05 | 1.99 | 1.86 | 1.96 |
| 200 | 1.8 | 1.84 | 1.78 | 1.87 | 1.86 | 1.88 | 1.86 | 1.92 | 1.9 | 1.9 | 1.86 |
| 300 | 1.97 | 1.82 | 1.96 | 1.89 | 1.81 | 1.84 | 1.82 | 1.86 | 1.86 | 1.87 | 1.84 |
| 400 | 1.79 | 1.76 | 1.87 | 1.83 | 1.9 | 1.84 | 1.82 | 1.86 | 1.86 | 1.87 | 1.84 |
| 500 | 1.94 | 1.78 | 1.86 | 1.79 | 1.74 | 1.94 | 1.73 | 1.73 | 1.88 | 1.86 | 1.84 |
| 600 | 1.79 | 1.84 | 1.82 | 1.79 | 1.86 | 1.84 | 1.8 | 1.84 | 1.78 | 1.83 | 1.82 |
| 700 | 1.72 | 1.79 | 1.72 | 1.9 | 1.76 | 1.73 | 1.85 | 1.75 | 1.82 | 1.82 | 1.79 |
| 800 | 1.91 | 1.76 | 1.79 | 1.75 | 1.75 | 1.79 | 1.86 | 1.71 | 1.74 | 1.73 | 1.78 |

Figure 14 The relationship between generations and average overlapping rate of each corpus. Data is obtained from the mean of each 10 tests of each case on four Corpora



Corresponding to Table 6, Table 7, Table 8 and Table 9, Figure 14 shows the relationship between the number of generations and the average overlapping rate. From this figure it can be seen that

- For Gov corpora, the average overlapping rate decreases dramatically with the first 200 generations and decreases slowly after 200 generations.
- For Wikipedia corpora, the average overlapping rate decreases dramatically with the first 500 generations and decreases slowly after 500 generations.
- For Reuters corpora, the average overlapping rate decreases dramatically with the first 500 generations and decreases slowly after 500 generations.
- For Newsgroup corpora, the average overlapping rate decreases dramatically with the first 200 generations and decreases slowly after 200 generations.

Overall, *the result is greatly improved at the beginning by increasing the number of generations but beyond a certain point there will be little improvement of the performance.*

In order to see how the *mutation rate* of the proposed GA-based method affects its performance, all parameters of the proposed GA-based method are fixed except the *mutation rate* to which is set to those values as listed in Section 5.1.2. The *mutation rate* has been set to be 0.01, 0.02, 0.03, 0.04, 0.05, 0.1 and have run 10 times of each case on four corpora, Gov, Wikipedia, Reuters and Newsgroup corpora. The resulting overlapping rate of each of the 10 experiments of each case on each corpus, as well as the average of the overlapping rate, is listed in Table 10, Table 11, Table 12 and Table 13.

In the tables below, *Max OR* refers to the maximum overlapping rate among the 10 results; *Min OR* refers to the minimum overlapping rate among the 10 results; and *Avg OR* refers to the average overlapping rate among the 10 results. (G:greedy method; GA: GA-based heuristic; K: mutation rate)

Table 10 The experiment results of each case of mutation rate on a SampleDB of Gov Corpus.

| Gov | | K=0.01 | K=0.02 | K=0.03 | K=0.04 | K=0.05 | K=0.1 |
|---|---|---|---|---|---|---|---|
| | G | GA | GA | GA | GA | GA | GA |
| Max OR | 3.2367 | 2.5429 | 2.2796 | 2.3286 | 2.2939 | 2.3224 | 2.4204 |
| Min OR | 3.1224 | 2.4612 | 2.26 | 2.1878 | 2.2429 | 2.2653 | 2.3612 |
| Avg OR | 3.2025 | 2.4998 | 2.2702 | 2.2588 | 2.2712 | 2.2925 | 2.388 |

Table 11 The experiment results of each case of mutation rate on a SampleDB of Wiki Corpus.

| Wiki | | K=0.01 | K=0.02 | K=0.03 | K=0.04 | K=0.05 | K=0.1 |
|---|---|---|---|---|---|---|---|
| | G | GA | GA | GA | GA | GA | GA |
| Max OR | 2.6479 | 2.5151 | 2.0765 | 2.1771 | 2.1268 | 2.1469 | 2.3058 |
| Min OR | 2.5855 | 2.1549 | 1.99 | 1.9537 | 2.0463 | 2.085 | 2.1972 |
| Avg OR | 2.603 | 2.3447 | 2.0408 | 2.0236 | 2.0888 | 2.1083 | 2.2682 |

Table 12 The experiment results of each case of mutation rate on a SampleDB of Reuters Corpus.

| Reuters | | K=0.01 | K=0.02 | K=0.03 | K=0.04 | K=0.05 | K=0.1 |
|---|---|---|---|---|---|---|---|
| | G | GA | GA | GA | GA | GA | GA |
| Max OR | 2.243 | 2.3988 | 1.8457 | 1.8297 | 1.8176 | 1.8277 | 2.016 |
| Min OR | 2.176 | 1.8497 | 1.6954 | 1.6874 | 1.6834 | 1.7154 | 1.8297 |
| Avg OR | 2.207 | 2.1867 | 1.7686 | 1.7505 | 1.775 | 1.7867 | 1.9293 |

Table 13 The experiment results of each case of mutation rate on a SampleDB of Newsgroup Corpus.

| Newsgroup | | K=0.01 | K=0.02 | K=0.03 | K=0.04 | K=0.05 | K=0.1 |
|---|---|---|---|---|---|---|---|
| | G | GA | GA | GA | GA | GA | GA |
| Max OR | 2.526 | 4.364 | 1.992 | 1.942 | 1.99 | 1.882 | 2.924 |
| Min OR | 2.358 | 2.448 | 1.752 | 1.726 | 1.758 | 1.794 | 1.892 |
| Avg OR | 2.474 | 3.1082 | 1.804 | 1.836 | 1.845 | 1.83 | 2.032 |

From Table 10, 11, 12 it can be seen that when K =0.03, the *Avg OR* of GA is much smaller than that of G, while when K =0.01 and K=0.1, the *Avg OR* of GA is very close to that of G. From Table 13, it can be seen that when K=0.02, the *Avg OR* of GA is much smaller than that of G while when K =0.01 and K=0.1, the *Avg OR* of GA is either greater than or close to that of G. Thus, the mutation rate should not be too high or too low. Either one may affect the performance of the proposed GA-based algorithm. *In summary, the performance of the proposed GA-based algorithm depends on the mutation rate, and good result may be obtained with a small mutation rate.*

# 6. Conclusion

An essential part of the task of deep web crawling is to select a suitable set of queries so that they can be sent to retrieve most of the documents with low cost. In this thesis, the hitting rate is set to 1, and the low cost is determined by the overlapping rate. To reach low overlapping rate, the selection of a set of suitable queries is reduced to set covering problem. This thesis work presents a GA-based set covering algorithm to select queries.

There are several steps in the GA algorithm where suitable methods and algorithms need to be found. In this thesis, algorithms are introduced for constructing initial population, for selecting parents, for performing crossover, and for conducting mutations. They are all well developed so that overall, it gives results better than those from using the greedy method.

We have carried out experiments using data from four corpuses wikipedia, Gov, Reuters and Newsgroup. We have showed with experimental results on how the parameters, such as mutation rate and the number of generations computed, affect the performance of the proposed GA algorithm. It is shown

- *the result is greatly improved at the beginning by increasing the number of generations but beyond a certain point there will be little improvement of the performance.*
- *the performance of the proposed GA-based algorithm depends on the mutation rate, and*

*good result may be obtained with a small mutation rate.*

With well set values of the parameters, the GA-based algorithm outperforms the greedy algorithm by approximately 20-30%.

## References

[1] Thanaa M. Ghanem and Walid G. Aref. Databases deepen the web. *IEEE Computer*, 73(1):116–117, 2004.

[2] Michael K. Bergman, The Deep Web: Surfacing Hidden Value, the Journal of Electronic Publishing 7 (1). 2001.

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

[4] L. Barbosa, J Freire: Siphoning hidden-web data through keyword-based interfaces. In: Proc. Of SBBD. (2004).

[5] C.H. Chang, M.Kayed, M.R. Girgis, K. F. Shaalan: A survey of web information extraction systems. IEEE Transactions on Knowledge and Data Engineering 18(10) (Oct. 2006) 1411-1428.

[6] S.W.Liddle, D.W. Embley, D.T.Scott, S.H.Yau: Extracting data behind web forms. In: Proc. Of Advanced Conceptual Modeling Techniques. (2002).

[7] A. Ntoulas, P.Zerfos, J.Cho: Downloading textual hidden web content through keyword queries. In: Proc. Of the Joint Conference on Digital Libraries (JCDL). (2005) 100-109.

[8] P.Wu, J. R. Wen, H.Liu, W.Y.Ma: Query selection techniques for efficient crawling of structured web sources. In: Proc. of ICDE. (2006) 47-56.

[9] J.Lu, Y.Wang, J.liang, J.Chen, J.Liu: An approach to deep web crawling by sampling. In: Proc. of Web Intelligence. (2008) 718-724.

[10] Y. Wang, J.Lu, J. Chen: Crawling Deep Web Using a New Set Covering Algorithm. ADMA (2009).

[11] J.J. Dongarra, "performance of various computers using standard linear equations software"', Computer Architecture News 20. (1992) 22-44.

[12] C. R. Reeves, Modern Heuristic Techniques for Combinatorial Problems, Blackwell Scietific,1993.

[13] J.H. Holand, Adaption in Natural and Artificial Systems, MIT Press, Cambridge, MA, 1975.

[14] D. Beasley, D.R. Bull, and R.P. Martin, "An overview of genetic algorithms: Part I Fundamentals", University Computing 15 (1993) 58-69.

[15] D. Beasley, D.R. Bull, and R.P. Martin, "An overview of genetic algorithms: Part II. Research topics", University Computing 15 (1993) 170-181.

[16] D. Goldberg, Genetic algorithms in search, Optimization and Machin Learning, Addison-Wesley, New York, 1989.

[17] 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), pp. 33-41.

[18] Alvarez, M., et al., *Extracting lists of data records from semistructured web pages*. 2007, Elsevier.

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

[20] www.ProgrammableWeb.com, 2007.

[21] B. He and K. C.-C. Chang. Statistical Schema Matching across Web Query Interfaces. In Proc. of SIGMOD, pages 217–228, 2003.

[22] W. Wu, C. Yu, A. Doan, and W. Meng. An Interactive Clustering-based Approach to Integrating Source Query interfaces on the Deep Web. In Proc. of SIGMOD, pages 95–106, 2004.

[23] H. He, W. Meng, C. T. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. VLDB Journal, 13(3):256–273, 2004.

[24] J.E. Beasley *, RC. Chu.A genetic algorithm for the set covering problem. European Journal of operational Research 94 (1996) 392-404.

[25] J.E. Beasley, "A Lagrangian heuristic for set-covering problems", *Naval Research Logistics* 37 (1990) 151-164.

[26] E. Balas and A. Ho, "Set covering algorithms using cutting planes, heuristics, and subgradient optimization: A computational study", *Mathematical Programming Study 12* (1980) 37-60.

[27] EJ. Vasko and G.R. Wilson, "An efficient heuristic for large set covering problems", *Naval Research Logistics Quarterly* 31 (1984) 163-171.

[28] J. Richardson, M. Palmer, G. Liepins, and M. Hilliard, "Some guidelines for genetic algorithms with penalty functions", in: J. Schaffer, ed., *Proc. Third International Conference on Genetic Algorithms,* Morgan Kaufmann, 1989, 191-197.

[29] http://en.wikipedia.org/wiki/Crossover_%28genetic_algorithm%29.

## Vita Auctoris

NAME: Shaohua Wang

PLACE OF BIRTH: Jilin, China

YEAR OF BIRTH: 1984

EDUCATION
Jilin University, Changchun, China
2003-2007 B.Sc.
University of Windsor, Windsor, Ontario
2007-2010 M.Sc.