Identifying Redundant Search Engines in a Very Large Scale Metasearch Engine Context

Ronak Desai[1], Qi Yang[2], Zonghuan Wu[3], Weiyi Meng[1], Clement Yu[4]
[1]Dept. of CS, SUNY Binghamton, Binghamton, NY 13902, USA, {rdesai1, meng}@binghamton.edu
[2]Dept. of CSSE, University of Wisconsin-Platteville, Platteville, WI 53818, USA, yangq@uwplatt.edu
[3]CACS, University of Louisiana at Lafayette, Lafayette, LA 70504, USA, zwu@cacs.louisiana.edu
[4]Dept. of Computer Science, University of Illinois at Chicago, Chicago, IL 60607, USA, yu@cs.uic.edu

Abstract
For a given set of search engines, a search engine is redundant if its searchable contents can be found from other search engines in this set. In this paper, we propose a method to identify redundant search engines in a very large-scale metasearch engine context. The general problem is equivalent to an NP hard problem – the set-covering problem. Due to the large number of search engines that need to be considered and the large sizes of these search engines, approximate solutions must be developed. In this paper, we propose a general methodology to tackle this problem and within the context of this methodology, we propose several new heuristic algorithms for solving the set-covering problem.

Categories and Subject Descriptors
H.3.4 [**Information Storage and Retrieval**]: Systems and Software – Distributed Systems; H.3.5: Online Information Services –Web-based Services.

General Terms
Algorithms, Performance, Design, Experimentation.

Keywords
Redundant search engine identification, set-covering problem

1. INTRODUCTION
Today the Web has become the largest information source in the world. As a result, millions of Web users and many applications try to find desired information on the Web on a daily basis. One of the most convenient ways for finding information on the Web is to submit queries to a Web search engine. It is well known that searching is now the second most popular activity on the Internet, behind only emailing.

The Web consists of two parts, the Surface Web and the Deep Web. The former contains web pages that are publicly indexable (i.e., these web pages have static URLs) and the latter contains documents that are not publicly indexable (such as private document collections) but are Web accessible through dedicated search interfaces as well as database records that are web accessible but are stored in database systems. It is estimated that the size of the Deep Web is hundreds of times larger than that of the Surface Web [1]. Most popular search engines such as Google and Yahoo are primarily designed to search the Surface Web and they use web crawlers to find publicly indexable web pages and make them searchable. In contrast, data in the Deep Web are searchable only through special search portals. For example, a publisher may create a web interface to allow web users to search papers/articles that are published by some journals owned by the publisher. In this case, these papers/articles are not directly available on the Web but are Web accessible through the special web interface. The total number of search engines on the Web is estimated to be several hundreds of thousands [1, 3].

It is of great interest to develop a single system that can provide access to all web pages on the Web, including both the Surface Web and the Deep Web. One possible solution to accomplish this is to create a very large-scale *metasearch engine* that connects to all useful search engines on the Web. A metasearch engine is a system that supports unified access to multiple existing search engines [10]. A user query submitted to a metasearch engine is passed to (some selected) search engines connected to the metasearch engine; when the results retrieved by the search engines are returned back to the metasearch engine, they are merged into a single result list for presentation to the user. As there are hundreds of thousands of search engines, including both Surface Web search engines and Deep Web search engines, a metasearch engine connecting to all useful search engines is an extremely large-scale system. Building such a metasearch engine requires highly innovative solutions to many challenging problems. For example, techniques are needed to automatically discover all useful search engines on the Web, to automatically incorporate search engines into the metasearch engine, and to accurately determine a small number of appropriate search engines to invoke for any given user query as it is neither efficient nor effective to pass each user query to a large number of search engines. Our WebScales project aims to develop the technologies for building very large-scale metasearch engines and many of the related issues have been addressed [11]. WebScales' eventual goal is to become a metasearch engine that connects all special-purpose search engines (not general-purpose search engines such as Google and Yahoo) on the Web.

One of the issues in building a large-scale metasearch engine is to discover useful special-purpose search engines on the Web automatically. In [4, 13], techniques that can be used to determine whether or not a web page crawled by a web crawler contains the interface of a search engine were presented. Among found search engines, many of them may be redundant in the sense that their searchable contents are contained in other search engines. In this paper, we study the problem of identifying all redundant search

WIDM'06, November 10, 2006, Arlington, Virginia, USA.

engines in a very large-scale metasearch engine context. Redundant search engines will not be used in the constructed metasearch engine as their inclusion would incur unnecessary cost for processing the query at redundant sites and for removing redundant search results.

Redundant search engines may occur in different situations and the following are some examples. (1) Some mirror sites may have identical search engines. (2) The contents of one search engine are contained in another search engine. For example, a large organization may have search engines at different levels of the organization and the contents of a search engine for a division may be contained in the search engine for the entire organization. As another example, the papers of the search engine for a database journal may also be entirely searchable from a search engine for all computer science papers. (3) The contents of one search engine are contained in several other search engines. For example, the contents of a search engine for all the publications of a department may be contained in the search engines of many publishers. As will be shown later that the general redundant search engine identification problem is an NP hard problem. Therefore, we aim to develop approximate and practical solutions to this problem. To the best of our knowledge, the redundant search engine identification problem has not been studied before.

The main contribution of this paper is two-fold. First, we propose a three-step methodology to solve the redundant search engine identification problem. Second, we propose and evaluate several new heuristic algorithms to tackle the *set-covering problem*. These algorithms and an existing greedy algorithm given in [5] are compared experimentally.

We should note that existing metasearch engines on the Web (e.g., Mamma at www.mamma.com, Dogpile at www.dogpile.com, and Vivisimo at www.vivisimo.com) are very small in scale as they typically connect to dozens or so search engines only; furthermore, they usually connect to only popular Surface Web search engines. Therefore, the redundant search engine identification problem is unique to large-scale metasearch engines that connect to special-purpose search engines.

The rest of the paper is organized as follows. In section 2, we provide a formal statement for the problem of identifying redundant search engines. It can be shown that this problem is equivalent to the *set-covering problem*, which is an NP hard problem. In section 3, we outline our solution to the problem. This solution consists of multiple steps designed to reduce the huge problem to a much smaller problem. In section 4, we discuss the issue of finding sample documents from search engines. In section 5, we provide several heuristic solutions to the smaller problem (but it is still the *set-covering problem*). In section 6, experimental results are provided to show the efficiency and effectiveness of these heuristic solutions. We conclude the paper in section 7.

2. PROBLEM STATEMENT

Definition 1. For a given search engine E, the document database of E, denoted D(E), is the set of documents that are searchable through E by submitting queries to the search interface of E.

Definition 2. For a given set of search engines S, a search engine E is said to be redundant if D(E) is a subset of the union of all the document databases of the search engines in S – {E}.

Definition 3. For a given set of search engines S, a subset of search engines $T \subset S$ is said to be redundant if the union of all the document databases of the search engines in T is contained in the union of all the document databases of the search engines in S – T.

With the above definitions, the problem of identifying all redundant search engines from a given set of search engines S can be stated as follows: Find a subset of search engines P such that S – P is redundant and the number of search engines in P is the smallest. In other words, from a given set of search engines, we want to find the smallest number of search engines that contain all the documents in all search engines. Note that in general P may not be unique.

The problem of identifying all redundant search engines is equivalent to the *set-covering problem* [5]. This problem can be stated as follows. Given N sets, let X be the union of all the sets. An element of X is *covered* by a set if the element belongs to the set. A *cover* of X is a group of sets from the N sets such that every element of X is covered by a set in the group. The set-covering problem is to find a cover of X of the minimum size. The problem has been proved to be NP hard [5]. A greedy heuristic solution to this problem is described in [5] (also see section 5 of this paper).

3. SOLUTION OVERVIEW

Because (1) the problem is NP hard, (2) the number of search engines could be in the hundreds of thousands, and (3) the total number of documents from these search engines could be in the billions, it is not practical to find a complete solution to the given problem. In fact, with a problem of this scale, even a relatively efficient heuristic algorithm (like the ones to be discussed in section 5) will not be practical. In this paper, we propose an approximate and practical solution that aims to identify most of the redundant search engines in a cost-effective manner.

First, we sketch a methodology that can reduce the problem to much smaller problems. This methodology consists of the following three steps:

- *Search engine categorization*. This step is to categorize the input search engines based on their content. Basically, it assigns these search engines to appropriate concepts in a given concept hierarchy. For example, a search engine for information about NBA should be assigned to concept "Basket Ball." In general, the same search engine may be assigned to multiple concepts because it may cover contents related to multiple concepts. A concept hierarchy containing a wide variety of concepts should be utilized. For example, the first three levels of the ODPH (Open Directory Project Hierarchy, http://dmoz.org) can be used. Several techniques for automatically assigning search engines to concepts in a concept hierarchy have been reported in the literature [6, 9]. These techniques aim to assign each search engine to its most specific concept. For example, if a search engine searches only NBA related pages and there is a concept "NBA" in the hierarchy used, then this search engine will be assigned to "NBA". If "NBA" is not present but "Basket Ball" is, then the search engine will be assigned to "Basket Ball". As another example, if a search engine searches general sports news including basket ball, base ball, tennis, etc., then it will be assigned to concept "Sports" rather than assigned to each particular sports concept. We assume that one of these methods is used to accomplish this step and search engine categorization will not be discussed further in this paper.

Based on the initial categorization, we divide the search engines into many subsets, each subset contains the search engines assigned to a leaf concept LC as well as the search engines that are assigned to the ancestor concepts of LC in the concept hierarchy. This is because these search engines are likely to overlap based on the categorization techniques described above. The number of subsets is the same as the number of leaf concepts in the concept hierarchy used. So if the hierarchy consists of the first two levels of ODPH, then there will be slightly over 550 leaf concepts; if the first three levels are used, the number will be over 6,000.

The motivation for performing this step is to reduce the number of search engines that need to be compared against each other. Specifically, we can focus on identifying redundant search engines among those that are in the same subset. In other words, this step practically divides the input search engines into many subsets such that we can consider each subset independently. This would make the problem significantly more manageable. Suppose 500,000 search engines are given and they were evenly divided into 1,000 subsets with no overlaps, then each subset would contain only 500 search engines. Note that after each subset is independently considered, we need to identify those search engines that belong to multiple subsets and keep only one copy.

- *Document Sampling*. Since a search engine typically has a large number of documents and it is not practical to obtain all the documents accessible from each search engine, we propose to get a sample set of documents to represent the document set of each search engine. It is important to coordinate the process of obtaining the sample sets from different search engines such that certain properties are preserved. For example, if two search engines have identical document sets, then the sample sets should reflect this property. Document sampling will be discussed in section 4.
- *Redundant Sample Sets Identification*. This step is to identify redundant sample sets among all sample sets corresponding to the search engines under consideration according to Definition 3. If a sample set is found to be redundant, its corresponding search engine will be considered to be redundant. Solutions to this step will be discussed in detail in section 5.

4. DOCUMENT SAMPLING

Most special-purpose search engines search information within an organization (say a university) or related to a specific topic (e.g., sports). These search engines are much smaller than general-purpose search engines such as Google and Yahoo which index billions of pages, but they still typically cover from thousands to millions of documents. As a result, using all the documents in these search engines for redundant search engine identification can still be prohibitively expensive due to a large number of search engines involved. Furthermore, it may be difficult to obtain their complete document database from these autonomous search engines. Based on these considerations, we propose to use a subset of each search engine's document database to perform redundant search engine identification and we will call such a subset a *sample set*. In this section, we discuss how to obtain an appropriate sample set for each search engine. We call the process of obtaining such a sample set as *document sampling*.

For a given search engine E, let S(E) denote the sample set of the documents for E that we would like to obtain. Clearly S(E) $\subseteq$ D(E). Our method for obtaining S(E) consists of three steps:

1. Estimate the size of D(E), i.e., estimate how many documents are in the document database of E.
2. Obtain an initial sample set of D(E) that contains approximately K% of the documents in D(E), for some integer K (e.g. K = 5). Let IS(E) denote the initial sample set.
3. Adjust IS(E) to form the final sample set S(E). The initial sample sets are obtained independently but the final sample sets need to satisfy certain requirement across different search engines. This requirement will be discussed shortly.

Several techniques exist for estimating the size of a search engine (e.g., [8, 12]) and the method described in [8] will be used in this paper. There are also a number of approaches for obtaining a representative subset of a search engine's document database [2, 7]. These approaches all collecting sample documents by submitting selected queries to the search engine and they differ on how the queries are formed. Note that Step 1 and Step 2 above could be carried out together because a subset of a document database can be used to perform size estimation [12]. For the rest of this section, we focus on the third step.

The reason the initial sample sets obtained in Step 2 cannot be used for redundant search engine identification is because they may not maintain some important relationships that exist between the full document databases. For example, consider two search engines E1 and E2 that have identical document databases, i.e., D(E1) = D(E2). It is not difficult to see that in general IS(E1) $\neq$ IS(E2), which makes detecting redundant search engines impossible. To retain the ability to identify redundant search engines effectively, we require that the final sample sets satisfy the following inter-sample-set requirement:

Inter-Sample-Set Requirement: For any two sets of search engines S1 and S2, if $\bigcup_{E \in S1} D(E) \supseteq \bigcup_{E \in S2} D(E)$, then $\bigcup_{E \in S1} S(E) \supseteq \bigcup_{E \in S2} S(E)$.

In plain English, the above requirement says if the union of the document databases of the search engines in S1 covers the union of the document databases of the search engines in S2, then the union of the sample sets of the search engines in S1 must also cover the union of the sample sets of the search engines in S2.

One special case of the requirement is when both S1 and S2 contain just a single search engine, say E1 and E2, respectively. One sub-case of this case is if D(E1) = D(E2), then S(E1) = S(E2). Another special case is when S2 contains a single search engine while S1 has multiple search engines. This case will cover the situation when the document database of one search engine is covered by the union of the databases of multiple search engines.

Due to the large number of search engines involved, it is not practical to verify the above requirement for each pair of search engine sample sets. We propose a document adding process to adjust the IS(E_i) for each search engine E_i (see Figure 1).

A key step in Algorithm IS_Adjust is to determine if a document *d* in the database of one search engine, say E1, is in the database of another search engine, say E2 (line 5). In practice, this can be carried out by forming a query based on *d* and submitting the

query to E2 to see if *d* can be retrieved. This process is explained in detail below. For each document *d* in IS(E1), a special query q(*d*), called *strong page query* of *d*, is formed, which consists of the important content terms in *d*; then submit q(*d*) to E2. As q(*d*) is constructed from *d* based on the important terms in *d*, it is likely that if *d* is in D(E2), *d* will be retrieved by E2 as one of the top ranked documents. We have implemented and tested this method. In our implementation, query q(*d*) consists of the following three types of terms: (1) terms in q* which was used to retrieve *d* from E1 when generating IS(E1); (2) terms in the title of *d*; and (3) the K most frequent *content terms* in *d*, for some small integer K (K is about 10). Content words exclude so-called *stopwords* such as "of" and "the" which carry no or little meaning but occur frequently. Our experiments indicate that this method is very effective and essentially all such documents (i.e., *d*) can be retrieved and ranked in top 10 results. If *d* is retrieved from E2, *d* is added to IS(E2).

```
Algorithm IS_Adjust
1. repeat
2.    for each search engine E_i
3.        for each document d in IS(E_i)
4.            for each search engine E_j ≠ E_i
5.                if d is in D(E_j) – IS(E_j)
6.                    add d into IS(E_j);
7. until no new document can be added to any IS(E);
8. for each search engine E_i
9.     S(E_i) := IS(E_i); /* find the final sample set */
```

Figure 1. Algorithm IS_Adjust

We note that Algorithm IS_Adjust is still expensive to execute. We plan to investigate if a more efficient algorithm can be developed in the future.

The proposition below summarizes the relationship between the IS_Adjust algorithm and the Inter-Sample-Set Requirement:

Proposition 1. The sample sets produced by the IS_Adjust algorithm always satisfy the Inter-Sample-Set Requirement.

Sketch of proof: Assume the proposition were incorrect, then there would be two sets of search engines S1 and S2 such that the union of document databases for S2 was a subset of the union of document databases for S1, but the union of the sample sets for S2 was not a subset of the union of the sample sets for S1. As a result, there would a document *d* that was contained in the sample set (and the document database) for a search engine E of S2, but not contained in the sample set for any search engine of S1. When the search engine E was selected in line 2 of the IS_Adjust algorithm, *d* would be found not in the document database of any search engine of S1. Thus the union of the document databases for S2 was NOT a subset of the union of the document databases for S1. This contradiction means that Proposition 1 must be true. □

5. SOLVING SET-COVERING PROBLEM

Let's first restate the set-covering problem in the context of identifying redundant search engines. Let SE = { E_i : i = 1, 2, …, N} be the set of the search engines under consideration (i.e., SE can be any one of the subsets of search engines obtained in the search engine categorization step in section 3). As in section 4, $S(E_i)$ denotes the sample set of the documents for search engine E_i. Also let X be the union of all sample sets for search engines in SE and M the total number of distinct documents in all sample sets, i.e., $X = \cup S(E_i)$ and $M = |\cup S(E_i)|$. A document is said to be covered by a sample set $S(E_i)$ (and the corresponding search engine E_i) if the document is in $S(E_i)$. A *cover* of SE is a subset of SE such that each document in X is covered by at least one search engine in the subset. A *minimum cover* is a cover with the minimum size. The set-covering problem is to find such a minimum cover. A search engine E_i (and the corresponding sample set $S(E_i)$) is redundant with respect to a subset G of SE if each document of $S(E_i)$ is covered by a search engine in $G – E_i$.

As mentioned earlier, the set-covering problem is NP hard and we need heuristic solutions for this problem. In [5], a greedy heuristic algorithm is presented to solve the set-covering problem. Applying to our search engines problem, this greedy algorithm can be stated as in Figure 2. In this algorithm, ResultCover is the cover to be generated (i.e., it is an approximate solution of the minimum cover) and Uncovered is the set of documents that are not covered by any search engine in ResultCover. The time complexity of the algorithm is O(M * N * min(M, N)), where N is the number of search engines and M the total number of distinct documents. The step in line 4 takes time O(M * N) and the loop that starts in line 3 takes at most min(M, N) iterations. The theoretical upper bound of the ratio between the size of the generated ResultCover and the size of a minimum cover is max $\{|S(E_i)| : i = 1, 2, \ldots, N\}$.

```
Algorithm Greedy
1. ResultCover := ϕ;
2. Uncovered := X;
3. while Uncovered ≠ ϕ do
4.     select a search engine E that is not in ResultCover
         and covers the most documents not covered
         by ResultCover;
5.     add E to ResultCover;
6.     remove all documents covered by E from Uncovered;
```

Figure 2. Algorithm Greedy

The document set of a search engine E_i is normally very large, and hence the size of sample set $S(E_i)$ cannot be too small. In our experimental implementation, the size of the sample sets is 500. It is clear that the upper bound provided by the greedy algorithm won't be acceptable. One reason that the upper bound is so high for the greedy algorithm is that a search engine in ResultCover could become redundant after other search engines are added later, although it was not redundant when added to ResultCover initially. For example, if search engine SE1 has documents {A, B, C, D}, search engine SE2 has documents {A, B, E} and search engine SE3 has documents {C, D, F}. Then using the greedy algorithm, SE1 will be considered first and all three search engines will be included in ResultCover. However, SE1 becomes redundant after SE2 and SE3 are added. Our algorithm tries to improve the accuracy by checking and removing such redundant search engines after ResultCover has been generated.

No data structure is given in [5] for the implementation of the greedy algorithm. To present our algorithm, we use a matrix to represent the relationship between the search engines and the documents. Each row of the matrix represents a search engine and each column represents a distinct document among all search engines. Thus, the matrix is an N×M matrix. The fact that a document is covered by a search engine is indicated by a value of

"1" in the cell of the matrix corresponding to the document and the search engine. The matrix can be implemented using either arrays or linked lists. In the following example, there are three search engines and five distinct documents: Search engine E1 covers two documents B and C, search engine E2 covers three documents C, D and E, and search engine E3 covers four documents A, B, C and D.

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0

Add another row at the bottom of the matrix for ResultCover with a value of zero for each column.

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0
ResultCover	0	0	0	0	0

For each search engine and ResultCover, call the corresponding row as its *Coverage*. For example, the Coverage of E1 is

0	1	1	0	0

and the Coverage of ResultCover at the beginning is

0	0	0	0	0

Our basic algorithm is called **CAR** for Check And Remove as it checks if any search engine is redundant and removes such search engines after a cover of SE (the set of all search engines under consideration) is obtained. This algorithm has two phases. Phase 1 is called *Cover Building* as it builds a cover of SE and Phase 2 is called *Redundancy Removing* as it tries to remove redundant search engines. The detail of the algorithm is given in Figure 3.

Algorithm CAR(SE)
/* Phase 1: Cover Building */
1. select any search engine E;
2. ResultCover := {E};
3. Coverage(ResultCover) := Coverage(E);
4. while ResultCover is not a cover of SE
   /* i.e., at least one cell in Coverage(ResultCover) is zero */
5. select the next remaining search engine E_i;
6. if E_i is not redundant with respect to ResultCover
7. ResultCover := ResultCover $\cup$ {E_i};
8. Coverage(ResultCover) := Coverage(ResultCover) + Coverage(E_i);
/* Phase 2: Redundancy Removing */
9. for each search engine E_i in ResultCover
10. if E_i is redundant with respect to ResultCover
11. Coverage(ResultCover) := Coverage(ResultCover) – Coverage(E_i);
12. ResultCover := ResultCover – {E_i};

Figure 3. Algorithm CAR

From Figure 3, the *Cover Building* phase is straightforward as it simply keeps adding search engines to the ResultCover until a cover of SE is obtained. A cover of SE is obtained if the value in every cell in the *Coverage* of ResultCover is at least 1. In line 6, E_i is not redundant with respect to ResultCover if and only if there is a cell in the *Coverage* of E_i whose value is 1 while its corresponding cell in the *Coverage* of ResultCover has a value 0. The coverage addition of line 8 simply adds the values in the corresponding cells of each column of the two coverages. Phase 2 checks each search engine in the current ResultCover and see if it is redundant, and if so, removes the search engine. A search engine E_i in ResultCover is redundant if each non-zero value in its *Coverage* is less than the corresponding value of the *Coverage* of ResultCover.

We now illustrate Algorithm CAR using an example. Consider the three search engines mentioned earlier. After the first search engine E1 is added to ResultCover, we have the following initial matrix:

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0
ResultCover {E1}	0	1	1	0	0

At this time, E2 is not redundant with respect to ResultCover, since E2 covers document D and E and neither is covered by ResultCover. So add E2 to ResultCover and the matrix becomes:

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0
ResultCover {E1, E2}	0	1	2	1	1

ResultCover is not a cover yet as document A is not covered. E3 is not redundant with respect to ResultCover because its value under A is 1 while the corresponding value of ResultCover is 0. So E3 is added to ResultCover and the matrix becomes:

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0
ResultCover {E1, E2, E3}	1	2	3	2	1

Now ResultCover is a cover of all search engines and Phase 1 is completed. To do Phase 2, check each search engine in ResultCover. E1 is redundant since it covers B and C only, and both are covered by more than one search engine (the values under B and C for E1 are smaller than those for ResultCover, respectively). Remove E1 from ResultCover, we have the matrix:

	A	B	C	D	E
E1	0	1	1	0	0
E2	0	0	1	1	1
E3	1	1	1	1	0
ResultCover {E2, E3}	1	1	2	2	1

E2 and E3 are not redundant; the final ResultCover is {E2, E3}.

The time complexity of the CAR algorithm is O(N * M), where M is the total number of distinct documents in all the sample sets of the search engines in SE and N is the number of search engines. In fact, it is not difficult to see that both Phase 1 and Phase 2 take time O(N * M) as they each checks all search engines once in the worst case scenario.

Another aspect of the greedy algorithm that could be improved is to use document regions instead of individual documents. A *document region* for a group of search engines is the set of documents that are covered by every search engine in the group, but not covered by any search engine not in the group. The concept of document region comes from Venn Diagram in set theory [5]. In the following example (see Figure 4), there are three search engines E1, E2 and E3 with a total of 80 documents. However, the 80 documents are divided into 5 document regions A, B, C, D and E. Region A corresponds to search engine group {E3} as it only contains documents in E3 but not in E1 and E2. Similarly, region B corresponds to search engine group {E1, E3} as it contains only documents in E1 ∩ E3 but not in E2. Other regions can be understood similarly.

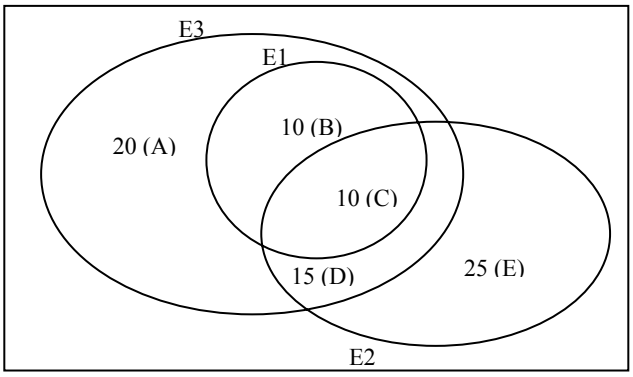

Figure 4. Illustration of Document Regions

When document regions are used, we need to consider the 3 search engines with the 5 document regions instead of the 80 documents. Notice that in theory for N search engines, there could be as many as $2^N - 1$ document regions. Since the regions form a partition of the set of all the documents, the number of regions will not exceed the number of documents. In practice, the number of document regions is likely to be much smaller than the total number of documents, as shown in the above example.

We still use a matrix to represent the relationship between the search engines and the documents regions. If a search engine covers a document region, the corresponding cell will have a positive value. In the following, we use the number of documents in the region as the value.

	A	B	C	D	E
E1	0	10	10	0	0
E2	0	0	10	15	25
E3	20	10	10	15	0

When document regions are used, our CAR algorithm will work the same way as before but we are going to call this algorithm in this case as **CARR** for Check And Removal with Regions. The time complexity of the CARR algorithm is O(N * R), where R is the number of document regions and N is the number of search engines. The time spent on constructing the document regions is not considered in this paper, because no data structure is presented for the greedy algorithm and the time for constructing the data structure is not considered for the greedy algorithm. Our experimental results in section 5 show that using document regions will speed up the process of identifying redundant search engines significantly.

It is possible to incorporate the basic idea of the greedy algorithm into the CARR algorithm by selecting the non-redundant search engine with the most number of uncovered documents in each iteration in the *Cover Building* phase of the CARR algorithm. We will call this version the **CARRG** algorithm.

6. EXPERIMENTS

In this section, we compare the four algorithms we presented in section 5, namely Greedy, CAR, CARR and CARRG, experimentally. All algorithms are implemented using the C language with the same data structure. We use two measures for comparison: the CPU time and the size of the ResultCover found by each algorithm. The former measures the efficiency of each algorithm while the latter measures the accuracy. The number of search engines in the ResultCover determines its size. In general, the smaller the ResultCover, the better as more redundant search engines have been removed.

Our experiments were carried out using simulated dataset where each search engine consists of a set of unique integers, each representing a document id. Our test data generator program takes the number of disjoint search engines (K), the total number of search engines (N) and the size of each search engine as arguments. It first generates K disjoint search engines of specified size. Then it generates N – K redundant search engines by randomly selecting document ids from the set of document ids of all the disjoint search engines. After generating the document ids for all search engines, it shuffles all search engines to produce a random order. All algorithms then use these search engines as input. Hence, they all process input search engines in the same order. In our experiments, N is fixed at 1,000 and each search engine has a fixed size of 500 documents. We let K take different values to test the algorithms with different degrees of redundancy. Specifically, K = 100, 200, …, 1,000 are used. For each K, 5 data sets are generated and used. Thus, the presented results are average results for 5 different data sets.

Table 1. Sizes of ResultCovers Produced by Different Algorithms

K	Greedy	CAR	CARR	CARRG
100	105.8	485.8	485.8	100
200	203.4	200	200	200
300	300.2	300	300	300
400	401.8	400	400	400
500	501	500	500	500
600	601.8	600	600	600
700	700.6	700	700	700
800	800.2	800	800	800
900	900.2	900	900	900
1000	1000	1000	1000	1000

Table 1 shows the sizes of the ResultCovers produced by different heuristic algorithms. The greedy algorithm performed surprisingly well despite the high theoretical upper bound. There are no significant differences among these algorithms in terms of accuracy except for the extremely high redundancy case (K = 100) where Algorithms CAR and CARR performed rather poorly. An explanation for this phenomenon is probably because Algorithms CAR and CARR are more sensitive to the input order of the search engines than the greedy algorithm, and when the degree of redundancy is extremely high, the redundancy removing phase of the CAR and CARR algorithms cannot function effectively. For example, consider the set of search engines shown in the Figure 5, where each "1" represents a document. When the input order of the search engines is (E1, E2, E3, E4, E5), Algorithm CAR (pay attention to Phase 2) will generate a non-optimal ResultCover {E2, E3, E4, E5}; however, if the input order is (E2, E1, E3, E4, E5), then it will generate an optimal ResultCover {E1, E4, E5}.

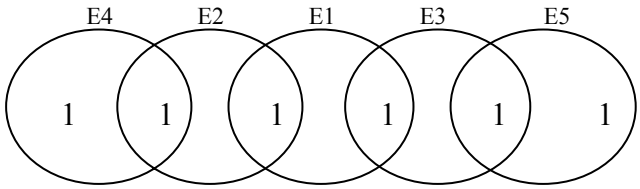

Figure 5. Illustration of Order Sensitivity

Figure 6 compares the efficiency of Algorithms CAR and Greedy. It is clear that Algorithm CAR takes significantly less CPU time compared to the greedy algorithm. While the CPU time of Algorithm Greedy increases quickly when the number of disjoint search engines (i.e, K) increases, the CPU time of Algorithm CAR increases much more slowly (also see Figure 7 for a more detailed CPU time of Algorithm CAR). This is expected because the complexity of Algorithm CAR is O(N*M), where M is the total number of distinct documents and N is the total number of search engines while the complexity of the Greedy algorithm is O(M * N * min(M, N)). For our data sets, N = 1,000 and M = 500*K. Intuitively, Algorithm CAR is more efficient than Algorithm Greedy because the former searches for any non-redundant search engine in its Cover Building phase while the latter searches for non-redundant search engine that has the highest number of uncovered documents in the same phase.

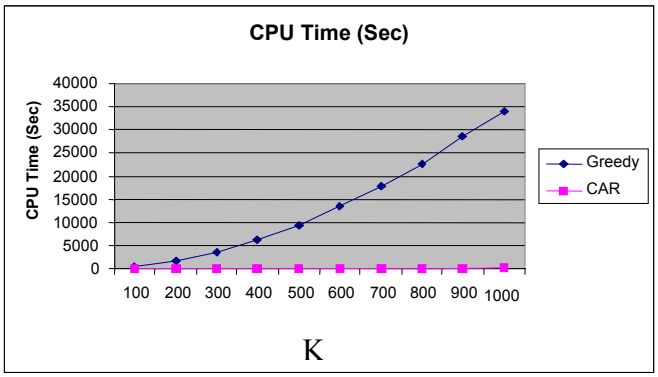

Figure 6. Comparison of Algorithms CAR and Greedy

Figure 7 compares the efficiency of Algorithms CAR and CARR. The figure shows that when the number of disjoint search engines K is small (less than 300), the two algorithms have similar performance in terms of the CPU time. However, after K exceeds 400, the CPU time of Algorithm CAR still keeps increasing while the CPU time of Algorithm CARR actually goes down.

To understand why the CPU time of Algorithm CARR deceases when K exceeds certain threshold, let's recall that the complexity of Algorithm CARR is O(N * R), where R is the number of document regions and N is the number of search engines. The value of R ranges from 1 to $2^N - 1$ and it is sensitive to the degree of redundancy among the search engines. In general, R increases initially with the decrease in redundancy, however, when the degree of redundancy goes below certain threshold, R starts to decrease. For the data set we generated, the value of R peaks when K is about 400. Intuitively, this can be explained as follows. Let KD denote the document sets of the K disjoint search engines and ND denote the document sets of the remaining N – K search engines. Since the document sets in KD are all disjoint, no non-empty intersections among these document sets are possible. As a result, no new document regions among these document sets will be formed other than those corresponding to the document sets in KD. On the one hand, when K gets larger, the document sets in ND are more likely to create more unique non-empty intersections with the document sets in KD, which leads to more document regions. On the other hand, when K gets larger, the number of remaining search engines (i.e., N – K) becomes smaller, which allows fewer unique non-empty intersections or document regions to be formed. As an extreme case example, when K = 1,000, N – K becomes 0. As a result, no new document regions can be formed and we have R = K = 1,000. In summary, the behavior of R is similar to K*(N–K) which has small values when either K or N – K is too small and large values when K and N – K are about the same. Figure 8 shows the change of R for one data set considered in our experiments. Notice that the curve of the CPU time of Algorithm CARR (see Figure 7) matches very well with the curve depicting the change of R.

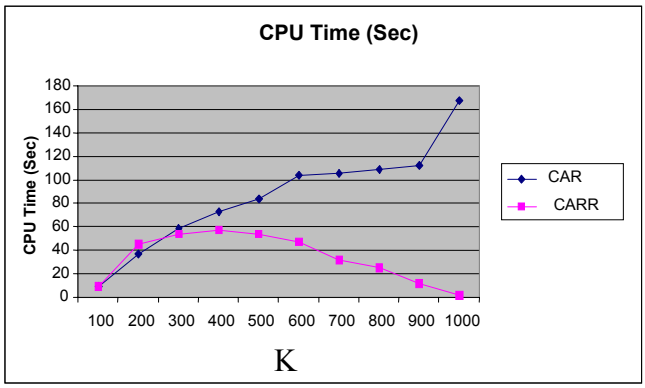

Figure 7. Comparison of Algorithms CAR and CARR

While Algorithms CAR and CARRG are much more efficient than Algorithm Greedy, they suffer from being much less accurate when the degree of redundancy is extremely high (i.e., when K is very small, e.g., when K is about 100 in our data sets). While we do not expect such a high degree of redundancy actually occurs in real special-purpose search engines on the Web, it is interesting to see if a good compromising solution can be found such that it can achieve good accuracy and at the same time has much better efficiency than the greedy algorithm. Algorithm CARRG has the potential to be such an algorithm. From Table 1, we can see that Algorithm CARRG has achieved optimal solutions for all cases.

Figure 9 compares the efficiency of Algorithms Greedy, CAR and CARRG. It can be observed that Algorithm CARRG is always more efficient than Algorithm Greedy and when K gets larger, the advantage of Algorithm CARRG over Algorithm Greedy becomes more and more substantial. Furthermore, the CPU time curve of Algorithm CARRG is similar to that of Algorithm CARR in that the time starts to decrease after K exceeds certain threshold.

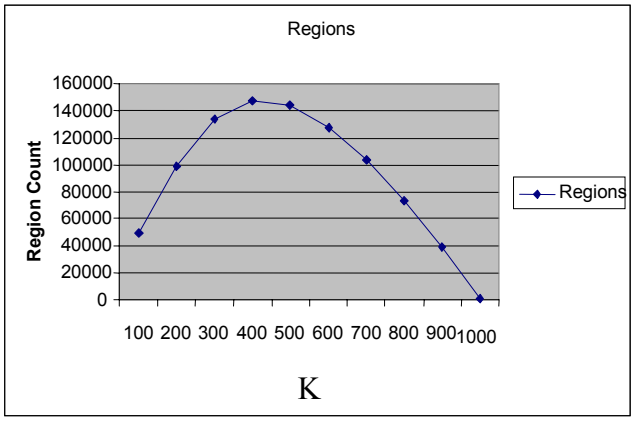

Figure 8. Behavior of R

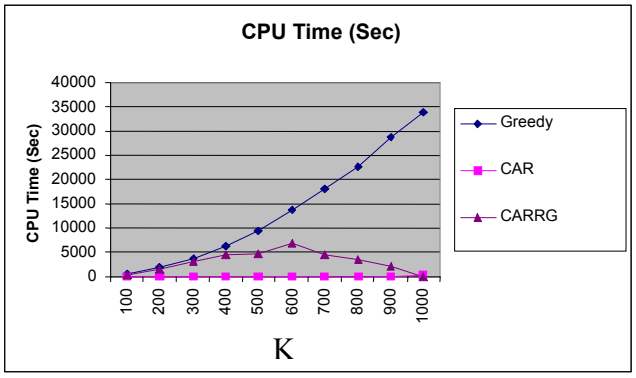

Figure 9. Comparison of Algorithms Greedy, CAR & CARRG

7. CONCLUSIONS

In a very large-scale metasearch engine context, using only non-redundant search engines can avoid sending search queries to un-needed search engines, which in turn can reduce network traffic for sending queries and receiving results and can reduce the time for identifying redundant search results. In this paper, we studied the problem of identifying redundant search engines from a given set of search engine. We showed that the essence of the problem is the set-covering problem – an NP hard problem. In consideration of the scale of the problem in the real world, we proposed a three-step methodology to reduce the huge problem to smaller problems. Then we proposed several new heuristic algorithms (CAR, CARR and CARRG) to tackle the set-covering problem and compared them with an existing heuristic algorithm (the Greedy algorithm) experimentally. Our experimental results indicate (1) Algorithm CARR is the most efficient but it may fail to identify many redundant search engines when the degree of redundancy among the input search engines is very high, and (2) Algorithm CARRG can identify all redundant search engines in our data sets while being significantly more efficient than the greedy algorithm.

In the future, we plan to investigate the following two issues. First, we plan to further study the document sampling problem in several directions such as determining the appropriate size of the sample set for each search engine and measuring the effectiveness of document sampling. Second, we would like to conduct experiments using real search engines instead of artificial data sets.

8. ACKNOWLEDEGMENTS

This work is supported in part by the following NSF grants: IIS-0208574, IIS-0208434, IIS-0414981, IIS-0414939, and CNS-0454298.

9. REFERENCES

[1] M. Bergman. The Deep Web: Surfing Hidden Values. White Paper of CompletePlanet. 2001. (Available at http://brightplanet.com/pdf/deepwebwhitepaper.pdf)

[2] J. Callan, M. Connell. Query-based Sampling of Text Databases. ACM TOIS, 19(2), April 2001.

[3] K. Chang, B. He, C. Li, M. Patel, and Z. Zhang, Structured Databases on the Web: Observations and Implications. SIGMOD Record, 33(3), September 2004.

[4] J. Cope, N. Craswell, D. Hawking. Automated Discovery of Search Interfaces on the Web. Australasian Database Conference, Adelaide, South Australia, 2003, pp.181-189.

[5] T. H. Cormen, C.E. Leiserson, R. L. Rivest. Introduction to Algorithms. The MIT Press, 1991.

[6] P. Ipeirotis, L. Gravano. Distributed Search over Hidden-Web: Hierarchical Database Sampling and Selection. VLDB Conference, 2002.

[7] P. Ipeirotis, L. Gravano, M. Sahami. Probe, Count, and Classify: Categorizing Hidden Web Databases. SIGMOD Conference, 2001.

[8] S. Karnatapu, K. Ramachandran, Z. Wu, B. Shah, V. Raghavan, R. Benton. Estimating Size of Search Engines in an Uncooperative Environment. Workshop on Web-based Support Systems, 2004.

[9] W. Meng, W. Wang, H. Sun, C. Yu. Concept Hierarchy Based Text Database Categorization. International Journal on Knowledge and Information Systems, 4(2), pp.132-150, March 2002.

[10] W. Meng, C. Yu, and K. Liu. Building Efficient and Effective Metasearch Engines. ACM Computing Surveys, 34(1), March 2002, pp.48-84.

[11] Research work related to WebScales can be found at site http://www.cs.binghamton.edu /~meng/metasearch.html.

[12] L. Si, J. Callan. Relevant Document Distribution Estimation Method for Resource Selection. ACM SIGIR Conference, 2003.

[13] Z. Wu, V. Raghavan, H. Qian, V. Rama K, W. Meng, H. He, and C. Yu. Towards Automatic Incorporation of Search Engines into a Large-Scale Metasearch Engine. Web Intelligence Conference, pp.658-661, 2003.