Automatically Detecting Boolean Operations Supported by Search Engines, towards Search Engine Query Language Discovery

Zonghuan Wu, Dheerendranath Mundluru, Vijay V. Raghavan
The Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA 70504-4330, USA
{zwu, dnm8925, raghavan}@cacs.louisiana.edu

Abstract

Each Web search engine provides query language through which it can communicate with its users and retrieve corresponding results to user queries. Supporting Boolean operations is a major characteristic of the query language. In this paper, we propose a novel, fully automatic, query probing based approach to identify what Boolean operations that are supported by a search engines and their corresponding syntaxes. Experiments show high effectiveness and efficiency. Along with this, we also provide a Web application called SE-BOSS (Search Engine Boolean Operation Scanning System) for interested users.

1. Introduction

There are hundreds of thousands of **s**earch **e**ngines (SEs) existing on the Web, most of which are *Deep Web* SEs [13] and contain high quality content that are not crawl-able by SEs like Google.

MetaSearch Engine (MSE) is a system that provides unified access to several SEs that it knows how to communicate with. When an MSE receives a user query, it dispatches the query to selected SEs; results returned from SEs are then reorganized, merged and displayed to the user by the MSE [9]. The MSE approach provides convenient concurrent access to multiple SEs. More importantly, an MSE built on multiple *Deep Web* SEs provides a platform for users to search on tremendous amount of Web content that are not searchable through crawler based SEs such as Google. The state-of-the-art MSEs, such as Dogpile [1], Mamma [2], Kartoo [3], Profusion [4], Search.com [5], turbo10 [6] and others, are built on top of tens to up to *3,000* SEs.

Each SE provides an interface through which its users can input their queries to retrieve results. In most cases, as a Web information retrieval system, each SE has its query language model with operators and syntaxes (we will call them query patterns in the rest of this paper) through which a user can submit a more complex query than just keywords. The SE can understand queries in these patterns so that corresponding operations will be executed. Using Google as an example, it supports the Boolean operation of disjunction by using the operator "OR" between two keywords. However, due to the heterogeneity of SEs, different SEs may support different operations and/or use different symbols and syntaxes (i.e. different query patterns) to represent same operators. For example, the SE www.scrubtheweb.com supports disjunction by using the operator "|" between two keywords while "OR" is regarded as a stopword.

Knowing the query language model of SEs is important to SE users as well as to MSEs. By understanding SE query language models, users may resolve their confusions like, when they submit a query "Computer Science", whether the search engine explain the query as "Computer *AND* Science" or "Computer *OR* Science" or the phrase "Computer Science". Also, by applying the language model to their queries, users can send complex queries and use SEs more effectively. Similarly, when an MSE has knowledge of the query language model of its underlying SEs, it can effectively translate complex queries, in the MSE query language, into the language that its underlying SE uses and get more accurate results back. Moreover, by applying a customized combination of probe queries, especially by using Boolean operators, that an autonomous search engine supports, an MSE will have the capability to collect special representative information about SEs that would not otherwise be possible. For example, this additional information will be helpful to better determine rank position of documents in the retrieval output, compared to using only term distribution statistics and hyperlink-based popularity characteristics of documents in the retrieval output [17, 18].

However, not many present MSEs have the capabilities of discovering the query language model of its underlying SEs. There are a few MSEs that support complex queries such as Boolean operations and "*PHRASE* Search", either manual approaches or proprietary techniques are used to discover the operation syntaxes of SEs.

Manual and semi-automatic approaches are expensive and not scalable when the number of SEs that an MSE has increases. Toward solving the query language discovery problem, in this paper, we propose a query probing based, robust, highly effective and automatic approach, to detect the basic operations, including Boolean operations such as disjunction (OR), conjunction (AND) and negation (NOT) and a few other common operations that an SE may support, such as *PHRASE, FST and SND* operations, and discover their corresponding syntaxes.

In section 2, we introduce the background knowledge including query language model, query probing and SE connection. After we introduce and discuss prior relevant research on SE query language discovering in section 3, we describe our approach in section 4 and it is validated through experiments that are presented in section 5. Finally, we conclude in section 6.

2. Background

In this section, we briefly introduce basic terminology such as Customized MSE, Query-Probing and SE Connection that are needed for subsequent developments.

Customized MetaSearch Engine. Customized MSE systems such as SELEGO, Bright Planet's DQM and Turbo10 emerged recently [6, 8, 10, 12]. Their users are able to build own MSEs on demand by simply providing the url's of those SEs they wish to include in the MSE and the SE incorporation is automatic and instant. Users are then able to submit their queries to the new MSE right away.

If an MSE is able to detect the query language model of SEs in the process of automatic incorporation, MSE will also be able to handle complex queries such as Boolean operations, Phrase search and so on.

Query Probing. By sending queries with pre-selected terms to an autonomous SE and exploring the SE by analyzing the returned results, query-probing approach has been used to discover SE document language models [16], categorize SEs [15] and discover query language models [11, 14].

Search Engine Connection. When an MSE dispatches a query to an SE, it constructs and sends a query string in the format that the SE understands, and gets returned page from the SE. We call this the process of SE connection. In this paper, we need a program to conduct SE Connection process for the purpose of probing SEs. We use the SE Connection component of SELEGO [10, 12], which is a customized metasearch engine system that we built previously. Through this component, by giving the url of an SE and the query terms, a query string can be properly assembled and sent to the search engine and the corresponding result page can then be obtained.

3. Prior Research

To the best of our knowledge, the approach proposed in [11, 14] by Bergholz and B. Chidlovskii is the closest to our work. It uses query probing along with machine learning algorithms to identify query language features of Web data sources. The approach assumes that (1). SEs can be automatically connected (refer to section 2). (2). On the result page returned by an SE, *the number of returned documents that match the query* is reported by the SE and can be identified and extracted.

Authors defined a few query models such as 'A', 'A B', '"A B"', "+A +B", "A AND B", "A + B" and so on. When queries are submitted with an operator to be classified, by examining the matched numbers of documents in the result set with rules in Boolean algebra such as $|A \vee B| \geq |A|$, $|A \wedge B| \leq |A|$ and a few others, the corresponding operation could be derived. Authors defined features based on the matched document numbers and used a set of *22* predefined probe queries and a number of SEs to train the learner and generate classification rules that distinguish different semantics. A mean accuracy of *86%* for the set of most frequent operators has been reported [14]. However, the approach is not applicable to SEs that do not report the number of retrieved documents on result pages. In our survey (see section 4 for detail) on *182* SEs, it was found that 20% of them were not providing the match numbers. Moreover, it is not trivial to automatically identify and extract this number from result pages and the effectiveness was not discussed.

Among other approaches, one possible way to detect the operators is to analyze the "help page" of an SE, which normally provides information about how to use operators to construct complex queries. This process involves identifying links with captions like, "Help", "Tips" etc. at the SE interface page and, once such a link is found, an analysis of the page is done to find the information. This approach has apparent limitations. First, many SEs do not have such help pages; besides, it is found that the information on the help pages of some SEs are sometimes obsolete, incomplete, or incorrect [11]. Second, it is difficult to automatically locate the help pages effectively.

Another approach is to download and analyze the result documents returned by SEs. The drawback is that it is very time consuming to download documents and then to parse it to detect the presence of query terms. Also, distinguishing the valid document link to download from other links (such as links for service pages, advertisements) is a problem that may affect the effectiveness of the analysis.

To overcome the limitations that above approaches have, we propose an efficient approach based on query probing and link analysis for automatically discovering the query language features of an SE. As reported in section 5, our approach showed an accuracy of over 97% in correctly detecting the operators.

4. Proposed Methodology

Before we get into details, we first define few terms that we use for the purpose of simplifying the description of our approach:

Definitions 4.1: *Impossible Query Term* and *Valid Query Term*:

When a term t is submitted to an SE s as a query, δ results are returned,

t is an *Impossible Query Term* to s when $\delta = 0$.The query is an *impossible query*.

t is a *Valid Query Term* to s when $\delta > 0$. The query is a *valid query*.

Definitions 4.2: *Impossible Query Page* and *Valid Query Page*:

The result page (which displays 0 results for the query) that an SE returns for a given impossible query is an *impossible query page*.

The result page (which displays δ ($\delta > 0$) results for the query) that an SE returns for a given valid query is a *valid query page*.

Note that though most SEs return multiple pages of results for a valid query, in this paper, we only need the first result page and we refer it as the result page.

Definitions 4.3 *Static Links* and *Dynamic Links*:

On n $(n>0)$ different html pages, a link is a *Static Link* if it appears on each of the n pages with the same captions or urls. Otherwise, it is a *Dynamic Link*. An example for static link in SE result pages can be a link with caption like "Home", "Products", "Services", or "Help" etc. that usually points to a fixed url. However, sometimes, the url may contain a unique session id. An example of a dynamic link could be a link to a commercial advertisement displayed which change periodically or with every request. Table 1 gives all scenarios of static and dynamic links.

Table 1. Static and Dynamic Links

URL	Caption	Link Type
Same	Same	Static
Different	Same	Static
Same	Different	Static
Different	Different	Dynamic

Just like the approaches in [11, 14], we assume that the SEs can be programmatically connected. However, unlike their approach, instead of using pre-selected terms for the probe queries, we dynamically generate two special query terms. One is an *Impossible Query Term* and the other is a *Valid Query Term*. They are connected by different operators to formulate a set of probe queries to discover the supported operations of any SE with their corresponding query patterns. Figure 1 illustrates our three-step approach to automatically detect the supported Boolean operations of an SE.

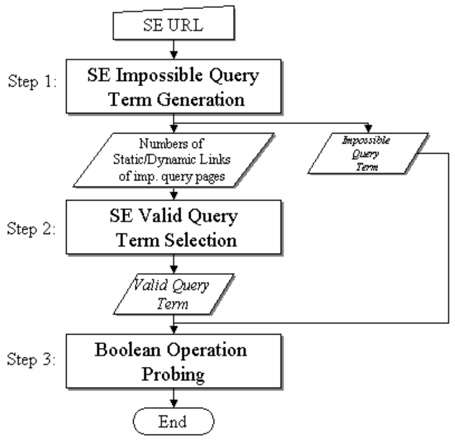

SE URL
Step 1:
SE Impossible Query Term Generation
Numbers of Static/Dynamic Links of imp. query pages
Impossible Query Term
Step 2:
SE Valid Query Term Selection
Valid Query Term
Step 3:
Boolean Operation Probing
End

Figure 1. Steps for query language detection

In the rest of this section, we explain the three steps from Figure 1 in detail.

Step 1: Impossible Query Generation

As specified above, when an impossible query term is submitted, an SE always returns *0* results. In our approach, we simply create terms like 'AnIm2345possibleQuery' that are extremely unlikely to be indexed by any SE in practice. When probed with such a query, a given SE would usually return a page with some statement saying that no results were found for the query. In addition, such a page usually has a few links, which can be either static (appear in all impossible query pages) or dynamic (appear in the specific impossible query page). The numbers of static and dynamic links are used as a feature to identify valid query terms in step 2 and to detect query operations in step 3.

By probing an SE twice with two different impossible query terms, we can get the number of static links by extracting the links that have common urls or captions for both the queries. The links left on both pages are dynamic links. If the numbers of the dynamic links on both pages are slightly different, we use their average.

Step 2: Valid Query Generation

After collecting impossible query term and related information, the next step is to find a valid query term.

For a valid query term, an SE always returns more than *0* results on the returned page (valid query page). Usually, a valid query page has significant difference from an impossible query page. Assuming this is true; we generate a list of candidate valid query terms, submit them to the SE, and evaluate the returned pages by using the total number of links. For some candidate valid query term, if the difference between the total number of unique links in its result page and the numbers that we collected from impossible query pages in step 1 is above certain threshold, which means there is significant difference between the two pages, we select it as a valid query term. Otherwise, we just discard this term. This step can be further divided into two parts: in step 2.1, the candidate valid query terms are generated; in step 2.2, the heuristic logic for selecting a valid query term is provided. The candidate terms are checked by this logic one by one, until a valid query term is found.

2.1. Generating candidate valid query terms. We generate two lists of query terms to be candidate valid query terms. Usually, the SE interface page provides descriptions that relates to the content of the SE. For example, the description may include important terms such as the name of a company and its products/services that may frequently appear on the company's SE interface page as well as in many Web pages that this SE indexes. By querying the SE with such important terms, the SE is likely to return results; so these terms are likely to be valid query terms. Based on this observation, we made an assumption that the more frequent a term appears on the interface, the more likely that it is a valid query term; thus, we create the first candidate term list that contains 10 most frequent terms extracted from the SE's interface page (stop words are removed). However, a few SEs (e.g. http://www.metor.com) have extremely simple interface that do not have enough good terms, so we also collect a set of terms that are very generic to construct the second candidate term list. Our assumption is that these terms are so generic that some of them will always be found in document collections of most, if not all SEs. We collect candidate terms from the homepage of CompletePlanet's [7] SE directory that classify *70,000+* SEs into *42* categories. We used all *54* terms that appear in these categories as the other set of candidate valid query terms.
We start with trying to probe SE by using the terms from the first list; if none of them is identified as a valid query term, we then try the terms in the second list. The following section explains how a valid query term is selected.

2.2. Valid query term selection. Figure 2 shows the rules to check whether a candidate term is indeed a valid query term.

```
func isQueryValid ($t_{can}$, $d_{imp}$, $s_{imp}$) returns boolean
    if ($|t_{can}| - |d_{imp} + s_{imp}|$) > $d$ then              /*1*/
        return true;
    else if $\big||t_{can}| - |d_{imp} + s_{imp}|\big| \leq d$ then      /*2*/
        return false;
    else                                                           /*3*/
        remove links in $s_{imp}$ from the result page of $t_{can}$
        if number of remainining links $\geq e$ then
            return true;
        else
            return false;
        end-if
    end-if
endfunc
```

Figure 2. The heuristic logic for identifying a valid query term

In Figure 2, respectively, d_{imp} and s_{imp} indicate dynamic links and static links on the impossible query page*s* that we got previously (see step 1); $|d_{imp} + s_{imp}|$ is the number of all links on a impossible page. t_{can} is the candidate valid query term. For a given SE, when t_{can} is submitted, the returned result page, called *Candidate Query Page*, has $|t_{can}|$ unique links. Let d be a threshold used to determine whether the candidate query page is significantly different from an impossible query page. At /*1*/, the first if-condition tests whether the total number of links in the candidate query page is significantly greater (difference > d) than the sum of the total number of links in impossible query page (i.e., the sum of dynamic links and static links in the impossible query page) and threshold d. If yes, it implies the page is different from the impossible query page and therefore is a valid query page. Note that $d = 7$, was found to be good in our experiments. At /*2*/, when the first if-condition is not satisfied, but the difference between the total number of links in the candidate query page and the total number of links in the impossible query page is less than or equal to d (i.e. the number of results retrieved is between 0 and d), then we discard the term because of the insignificant difference between the candidate query page and impossible query page.
Otherwise, at /*3*/, there are two possibilities left: A) The candidate query is a valid query page if it includes 0 or only a subset of all the static links displayed. B) The candidate query page is an *error page* that generally has 0 or very few links unrelated to the user query. This might happen when the query contains characters that may not be recognizable by the SE. We still consider this as a special form of impossible query. In order to differentiate the two cases, from all the links extracted from the candidate query page, we remove all the static links that are found in s_{imp} and then check whether the

total number of the remaining links is greater than a threshold e. If yes, we regard t_{can} as a valid query term. Otherwise, we discard t_{can} and try with a new candidate term.

Step 3: Query Operation Detection

To conduct the detection, we've surveyed *182* SEs including both general purpose and specialty SEs that we randomly picked from CompletePlanet's [7] website. As shown in table 2, we found that *89%* of these SEs support *AND*, *71%* support OR and 60% support NOT.
Note that a few SEs provide "Advanced Search" interfaces at which a complex form is provided for users to customize queries (eg. Google's Advanced Search interface is located at: http://www.google.com/advanced_search?hl=en)However, in this paper, we only deal with query language models of SE simple interfaces at which there is only one text field for users to input queries.

Table 2. Search engines and their supported Boolean operations

Operation	Number of SEs	Percentage
AND	162	89%
OR	132	71%
NOT	110	60%

In this step, based on the survey, we automatically detect which of the above three operations are supported by an SE and how the query patterns are represented. Also, with slight extension, we also detect *FST* (first query term should appear in all retrieved documents) and *SND* (second query term should appear in all retrieved documents) and *PHRASE* (all retrieved documents should contain all the specified query terms in the given order) operations.
In this step, the valid query term and the impossible query term found in previous steps are used to formulate probe queries to discover the operations supported by a given SE. Based on our survey, we summarized 15 query patterns that are used to generate all the probe queries. Table 3 displays these query patterns and their possible semantics.

Table 3. Possible Operators and Query Patterns

Operations / Query Patterns	AND	OR	NOT	FST	SND	PHRASE
t_1 t_2 (default op)	Y	Y		Y	Y	Y
$+t_1$ $+t_2$,	Y					
t_1 ***AND*** t_2, t_1 ***and*** t_2 t_1 ***And*** t_2, $t_1 + t_2$	Y				Y	
t_1 ***OR*** t_2, t_1 ***or*** t_2, $t_1 \mid t_2$, $t_1 \mid\mid t_2$		Y				
t_1 ***AND NOT*** t_2, t_1 ***ANDNOT*** t_2, $t_1 - t_2$, t_1 ***NOT*** t_2			Y			
"t1 t2"						Y

For example, for the conjunction operation, possible patterns are 't_1 t_2', '$+t_1$ $+t_2$', 't_1 **and** t_2', 't_1 **And** t_2', 't_1 **AND** t_2' and '$t_1 + t_2$'. Also for simplification, we omitted the possible semantics when any operator specified can be considered as a stopword or a literal.
In our method, we first detect the operation of the pattern 't_1 t_2', which we call it as the "*default operation*" of an SE. Based on the default operation, which can be *AND, OR, FST SND* or *PHRASE,* we then detect other operations through probing as described in Figure 3.

```
if default operation is OR then
        Detect the support for AND, SND, NOT
else if default operation is AND then
        Detect the support for OR, NOT, SND
else if default operation is FST then
        Detect the support for OR, AND, SND
else if  default operation is SND then
        Detect the support for OR, AND, NOT
end-if
```

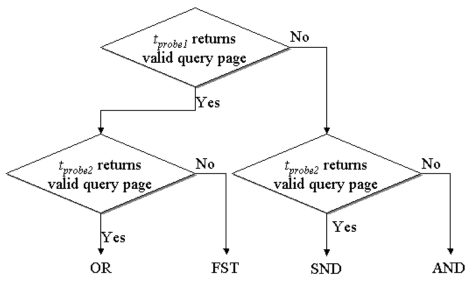
Figure 3. Boolean operation detection for search engines

Please note that the *FST* operation can only appear as a default operation (See Table 3). Also note that one flaw of the query probing using valid query term and impossible query term is that it is not able to detect the support for *NOT* operation when the default operation is *FST*. Another issue that needs to be clarified is that, in this algorithm, we consider *PHRASE* as a special form of *AND.* To further differentiate it from the *AND* operation, several result document samples needed to be taken. We will not discuss it in this paper since we found that, in our survey, it is very rare (One out of *182* SEs) that the default operation of a SE is *PHRASE*.
The rest of the section describes this step in detail.

3.1. Detecting the Default Operation. As it can be seen from table 3, 't_1 t_2' can mean five kinds of operations for different SEs (*AND, OR*, *FST*, *SND*, or *PHRASE*).

Figure 4 shows the rules to detect the default operator of an SE.

As shown in Table 3, other than *default* query pattern, only *AND* query patterns can be semantically equivalent to *SND*. For example, if 't_{vld} **AND** t_{imp}' returns an impossible query page but if 't_{imp} **AND** t_{vld}' returns a valid query page, then we conclude that query pattern t_1 **AND** t_2 is for *SND* operation. Thus, let t_{probe1}, t_{probe2} be the default probe queries obtained by binding t_1 to t_{vld} and t_2 to t_{imp} where t_{probe1} = '$t_{vld}\, t_{imp}$' and t_{probe2} = '$t_{imp}\, t_{vld}$'. If both t_{probe1} and t_{probe2} return valid query pages, then the default operation is *OR*. If at least one result page returned by t_{probe1} and t_{probe2} is a valid query page, the default operation should be either *FST* or *SND*. If both t_{probe1} and t_{probe2} return impossible query pages, then the default operation should be *AND*. The algorithm flowchart is shown in Figure 4. In Figure 4, to find if a result page is valid query page or not, we re-use the function defined in Figure 2.



t_{probe1} returns valid query page
No
Yes
t_{probe2} returns valid query page
No
t_{probe2} returns valid query page
No
Yes
Yes
OR
FST
SND
AND

Figure 4. Default operation detection

3.2. Detecting the support for OR, AND operations. As shown in Figure 3, we detect the support for *OR* operation with its possible patterns only if the default operator is either *AND* or *FST* or *SND* search. Similarly, we detect the support for *AND* operation and its entire different syntaxes only if the default operation is either *OR* or *FST* or *SND*. Different patterns of *OR* queries used are: t_{vld} **OR** t_{imp}, t_{vld} **or** t_{imp}, t_{vld} | t_{imp}, and t_{vld} || t_{imp}. Similarly the different patterns of *AND* queries used are: $+t_{vld}$ $+t_{imp}$, t_{vld} **AND** t_{imp}, t_{vld} **and** t_{imp}, t_{vld} + t_{imp}, and t_{vld} **And** t_{imp}. The rules to find the support for the both *OR, AND* operation patterns is similar to the rules shown in Figure 4. The only difference is that we need to apply the query patterns corresponding to AND and OR operations.

3.3. Detecting the support for NOT operation. We form different probe queries to detect the *NOT* operation support based on the supported default operation. For example, if the default operator is *AND*, different patterns of *NOT* queries used are: t_{vld} - t_{imp}, t_{vld} **AND NOT** t_{imp}, t_{vld}**NOT** t_{imp}, and t_{vld} **ANDNOT** t_{imp}. Note that t_{vld} always appears in front of t_{imp}. However, if the default operation is *OR*, different patterns of *NOT* operation used are: t_{imp} $-t_{vld}$, t_{imp} **AND NOT** t_{vld}, t_{imp} **NOT** t_{vld}, and t_{imp} **ANDNOT** t_{vld}. Note the order of appearances of t_{vld} and t_{imp} are reversed. For example, if the default operation is known as to be *AND*, we send probe query 't_{vld} $-t_{imp}$' and if a valid query page is returned, it indicates that *NOT* is supported by pattern t_1 $-t_2$. If the default operation is known to be *OR*, we send probe query 't_{imp} $-t_{vld}$' and if an impossible query page is returned, it implies the *NOT* operation has been executed.

5. Experimental Results

We have tested our algorithm on *128* SEs, which includes both general purpose and specialty SEs from various domains. Most of the specialty SE's have been taken from CompletePlanet's SE directory [7]. We also randomly collected a few SEs from other sources, including SEs incorporated by profusion.com, search.com, turbo10.com. Following are the different domains from which SE's are included in our test collection:

General Purpose SE's:	*21*
Sports/Basketball:	*10*
Business/Small Business:	*22*
Health/Cancer:	*16*
MetaSearch Engines:	*5*
SE's randomly taken from other sources:	*54*
Total:	*128*

Please note that, to avoid bias of experimental results, these SEs are selected independently of the search engines used in our survey.

5.1. Impossible Query Term Generation

Our impossible query term generator uses very simple method to generate dynamic impossible query terms such as 'AnIm2376possibleQuery' in which the value *2376* is randomly generated. We found that the approach is so effective that, in all cases of our experiment, it successfully generates impossible queries.

5.2. Valid Query Term Selection

The success rate for finding a valid query term from the two lists of candidate valid query terms has been over *99.21%*. The only failed case in our experiments is an SE that only returned at most 5 results (which is < *d*) for all the queries on its result page (for most of other SEs, the number is usually *10* or more).

Another important question is how efficiently a valid query term can be generated from the two lists of candidate valid query terms. For this we tracked the generation of valid query terms in our experiment and found that, on average, only *1.132* probe queries were used to select a valid query term. Therefore it indicates that the approach to automatically select a valid query term is not only accurate, but is also very efficient.

5.3. Boolean Operation and Query Pattern Detection

Since we failed to generate valid query term for one of the *128* SEs, we detected the Boolean operations and their corresponding query patterns on the remaining *127* SEs. To validate our results, we manually inspected all SEs and compared the results with programmatically generated results.

Overall, our system showed an accuracy of *97.63%* i.e. out of *127* SE's, *124* SE's were correctly classified. This accuracy is specified based on consideration of all the operation patterns used in the system i.e. if there is at least one operator which was wrongly detected for a particular SE, we considered that the system failed to classify this particular SE as a whole. For the 3 failed SE's, the default operation was wrongly classified. It also results in the failure of detecting other operations since they rely on the detection of default operation.

5.4. Efficiency Analysis of the approach

The number of probe queries used for an SE is the key factor for the time involved in detecting an SE's query language features. As shown in section 5.2, the number of probe queries used in selecting a valid query term is on average only 1.132. 2 probe queries are used to get 2 impossible query pages for detecting the static and dynamic links. 2 probe queries are needed for detecting default operation and 4 probe queries are needed for *NOT* operation detection. In addition, if default operation is *OR*, we need 5 *AND* probe queries whereas we need 4 *OR* queries when default operation is *AND*. Therefore, in total, 13.132 or 14.132 probe queries are needed to detect the query patterns of all three basic Boolean operations (*AND, OR, NOT*), depending on whether the default operation is *AND* or OR. 4 more probe queries are needed if *SND* is needed to be detected. In other cases of default operations (FST and SND), the number of probe queries used is just slightly different from what has been used in the case when default is either AND or OR.

6. Conclusions and Future Work

In this paper, we have proposed a novel approach to detect the basic Boolean operations that an SE supports. This highly effective and efficient approach is based on a series of simple and robust techniques such as impossible query generation, valid query generation and link analysis. By a comprehensive survey, followed by experiments on 128 SE's with a set of most commonly used operators, we achieved a very high overall accuracy of over 97%. With SE Boolean operation detection, MSEs will be capable of dispatching more accurate queries to search engines; it also provides researchers a tool for analyzing search engines in large scale.

We have set up a Web application called SE-BOSS, which detects the various operations and their query patterns supported by a SE given its URL, for experiment/demo purposes at: http://lincstaff2.cacs.louisiana.edu:8080/metasearch/SubmitURL.

Finally, we list a few directions for future work:

1. **Large Scale Test:** Current testing on 128 SE's showed high effectiveness of the algorithm heuristics. However, due to the highly dynamic Web environment, we plan to enlarge the test bed and validate our method.
2. **Improve Valid Query Generation and Valid Query Page Identification:** The four cases of false detection were caused due to the inability of correctly identifying valid query pages as the current approach of using the numbers of dynamic and static links is not able to handle these particular SEs, though the approach is simple and effective in most of the cases. More robust and sophisticated features, such as the structure of the result page, may need to be studied and applied to further improve the effectiveness.
3. **Detecting the support of more complex Boolean operations.** In this paper, we dealt with the simple Boolean operations. It is still an open question to find out how heterogeneous SEs support complex operations such as ((t_1 AND t_2) OR t_3 NOT t_4). We plan to extend our work to address this issue too.
4. **Automatically discover operations of advanced search interfaces**. Our current work deals with only basic search interfaces. However, many SEs have advanced interfaces on which a complex html form is provided, which can be customized to organize complex queries. It would be interesting to be able to automatically discover the query language features of such SE advanced interfaces.

Acknowledgements: This work is supported in part by the IT Initiative of the State of Louisiana to Lafayette.

7. References

[1] Dogpile. http://www.dogpile.com/

[2] Mamma. http://www.mamma.com/

[3] KartOO. http://www.kartoo.com/

[4] Profusion. http://www.profusion.com/

[5] Search.com. http://www.search.com/

[6] Turbo 10. http://www.turbo10.com/.

[7] CompletePlanet, http://www.completeplanet.com.

[8] BrightPlanet, http://www.brightplanet.com/.

[9] W. Meng, C. Yu, K. Liu. Building Efficient and Effective Metasearch Engines. ACM Computing Surveys, Vol. 34, No. 1, March 2002, pp.48-89.

[10] http://www.selego.com, Creating MetaSearch Engines On-Demand.

[11] Bergholz, B. Chidlovskii. Using query probing to identify query language features on the Web. In Proceedings of the SIGIR 2003 Workshop on Distributed Information Retrieval, Toronto, Canada, August 2003.

[12] Zonghuan Wu, Vijay Raghavan, Weiyi Meng, Hai He, Clement Yu, and Chun Du. Creating Customized Metasearch Engines on Demand Using SE-LEGO. In Proceedings of Fourth International Conference on Web-Age Information Management (WAIM'03), Demo paper, pp.503-505, Chengdu, China, August 2003.

[13] Bergman, M. The Deep Web: Surfacing Hidden Value. Journal of Electronic Publishing, 7(1), 2001.

[14] Bergholz, B. Chidlovskii. Learning Query Languages of Web Interfaces. In Proceedings of the 2004 ACM Symposium on Applied Computing: 1114 – 1121.

[15] P. G. Ipeirotis, L. Gravano, and M. Sahami. Probe, count, and classify: Categorizing hidden-web databases. In Proc. ACM SIGMOD Conf., pp. 67-78, Santa Barbara, CA, USA, May 2001.

[16] J. P. Callan, M. Connell, and A. Du. Automatic discovery of language models for text databases. In *Proc.* ACM SIGMOD Conf., pp. 479-490, June 1999.

[17] M. Kim, V. V. Raghavan, and J. S. Deogun. Concept based retrieval using generalized retrieval functions. Fundamenta Informaticae, 47(1-2):119--135, 2001.

[18] A. H. Alsaffar, J. S. Deogun, V. V. Raghavan, and H. Sever. Enhancing concept-based retrieval based on minimal term sets. J. of Intelligent Information Systems, 14(2-3):155--173, 2000.