Design and Development of a Programmable Meta Search Engine

Manoj M*
Senior Research Fellow
CSIR - National Institute for Interdisciplinary Science and Technology,
Thiruvananthapuram - 695019, India.

Elizabeth Jacob
Scientist
CSIR - National Institute for Interdisciplinary Science and Technology,
Thiruvananthapuram - 695019, India.

ABSTRACT
To the web user, a Meta Search Engine (MSE) appears much like a regular search engine (SE). MSE, unlike an SE does not have an index. Instead, it dynamically queries multiple search engines; extracts, fuses and re-ranks results and presents to users. Generally, an MSE is developed from scratch even if the focus is on improving fusion ranking, query modification or domain specific search. This paper proposes a programmable MSE with the help of a LISP-like language called T! for interfacing search sources and for modifying fused results. This enables easy construction and deployment of an MSE service. The Programmable-Meta Search Engine (P-MSE) provides, but is not limited to the following new features: construction of a domain specific vertical or general MSE by addition of dynamic web pages as front-end; a tool for fully automated test-run of various fusion algorithms and query modification schemes using a program or script capable of http-querying as front-end on real-time web; a programmable service where account holders can upload scripts and deploy meta search engines. The concept of P-MSE developed here, is showcased by the MSE called SSIR (**S**SIR for the **S**avvy **I**nformation **R**etriever).

General Terms
Information Retrieval, Internet Search, Meta Search Engine, IR research tool.

Keywords
Information Retrieval, Programmable Meta Search Engine, Embedded Script, T! language, Programmable software.

1. INTRODUCTION
In recent years, excessive use of Internet is posing the challenge of an information explosion both in stored form as well as in real-time. Research in Information Retrieval (IR) needs to find new improved ways to mine data with user targeted query modification, improved fusion, ranking techniques and IR spam filtering. Meta Search Engines (MSE) were proposed and built as data mining tools. A Meta Search Engine appears much like a regular search engine (SE). MSE, unlike an SE does not have an index. Instead it dynamically queries multiple search engines; extracts, fuses and re-ranks results and presents it to users.

Vertical MSEs were developed for domain specific Information Retrieval (Nano-Spider [1], Cancer-Spider [2]). Developing a vertical MSE from scratch requires collaboration between computational scientists and the domain expert. A customizable MSE can save much time spent for such specialized engines to be designed. It can also be used for conducting experiments on the real web, while focusing on a specific area of IR like ranking. On the other hand, web experiments with collections like Text REtrieval Conference (TREC) [3] or Forum for Information Retrieval Evaluation (FIRE) [4] that create a controlled environment and differ from the real dynamic web can also benefit from such a tool.

2. RELATED WORK
There are numerous Meta Search Engines available on the web. Basically, they operate by collection fusion [5]. Due to their commercial nature, the exact implementation details of most engines are often not available. Details of search engines implemented as part of academic research like MetaCrawler [6], Inquirus-2 [7], and vertical MSE for Human Resource domain [8] are some of the publicly available documented MSE implementations.

Studies on Meta-Searching or fusion [9 10, 11, and 12] are more often done with data collection like TREC or FIRE data. It is shown that automatic IR frameworks on real web can be used as an alternative to data collections [13, 14, 15, and 16].

Open source engines available on the net have many limitations. Myriad and Chalipa [17, 18] are PHP scripts without any parallelism. An earlier version of Scroogle (a self advertised privacy aware third party proxy for Google) source code which is publicly available is a CGI-C script [19]. Helios [20] though an impressive MSE, and implemented by its developers as SnakeT [21], has only limited flexibility.

A flexible MSE calls for clean, simple easily re-programmable software. A programmable search engine system [22] designed by Google, uses context files to fine tune various search parameters.

However, it was observed that these systems have many limitations.

- MSE cannot handle consecutive page URLs for more results or for HTML pages having frames.
- Pages with inconsistent format need patterns with conditional processing.
- Primitive pattern extraction languages used in these MSEs [20] are inadequate to handle JSON, SOAP etc.
- Most of these MSEs run as a CGI-process (has associated process creating overhead) and use plain text files for storage.

- A configuration file even with a web-page interface never offers the power and flexibility of a program script.

P-MSE was designed to overcome these limitations and make a developer friendly tool for IR applications.

Section 3 briefly describes the T! scripting language and gives a detailed description of the architecture of the Programmable-Meta Search Engine (P-MSE) and how the programming interface is implemented. Section 4 shows how P-MSE has been used to implement a typical MSE. Section 5 concludes this work on meta search and its prospects as an IR tool.

3. DESIGN OF P-MSE

The Programmable - Meta Search Engine is a MSE which can be programmed with scripts to create custom fusion (search) services.

The P-MSE is designed to be a standalone web server. It can handle HTTP request from any other program and work as a typical Unix daemon. So, it does not need C++/CGI based web servers like apache. It uses an embedded scripting language interpreter called T!, to provide programming capability. Multiple independent T! scripts, one corresponding to each search source, running in parallel queries and collect the results.

3.1 Architecture of P-MSE

Figure 1 shows the logical architecture of a MSE implementation using the P-MSE server.

The server module listens in an HTTP port (8080) or other ports as pre-configured. Upon receiving a request (based on the virtual file path in the request), either an html file is served or an instance of Query handler is created. The ASIO with BOOST C++ library seemed to be easy to use with C++ streams, but they are unable to handle complete page output (~ > 80 KB). So a custom-made printf style function, capable of writing to sockets was used. Unlike their C function counterparts, extra facility to handle C++ strings via pointers were also added. When a new search request is received, an additional thread of execution with a new instance of query handler object is created.

For adding scripting capability, a LISP-like language called T! is developed. Its source code is based on symbolic expression (s-expressions) which have parenthesized syntax. At source level, it is exactly like LISP but with a different internal representation.

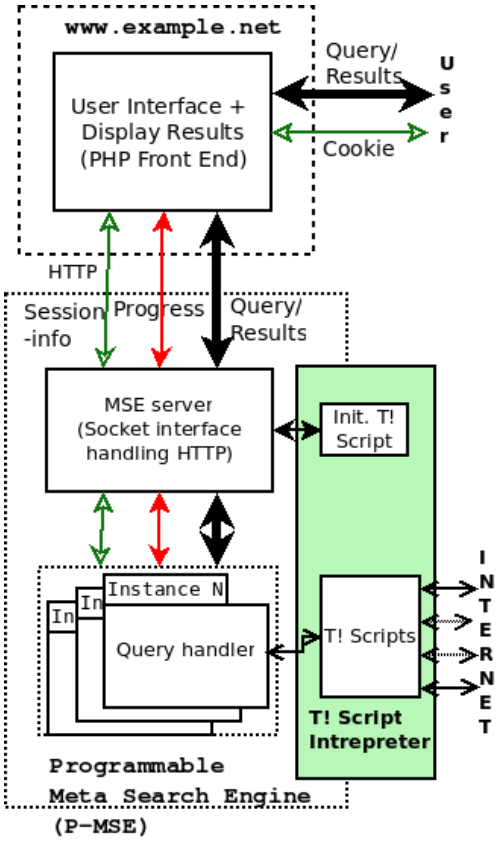

Fig 1: Logical architecture of the implementation of Programmable - Meta Search Engine

Though any language like Lua can be used, T! is selected because it is :

- ⇨ Light weight, minimal, Turing complete and extendable comparable only to LISP among popular embeddable languages.
- ⇨ Uses an easy to create and manage data structure instead of link list.

3.2 Scripting with T! language

The T! language is designed around a data-structure called T named after the arrangement of tea bags in a teabox. Though designed for P-MSE, it can be embedded as a scripting language in any application software.

In T!, s-expressions are converted into the array-based T data structure instead of link list as shown in Figure 2. T-bags and T-boxes form basic homo-iconic data types. T-bags are strings and T-boxes contain any number of T-bags or T-boxes recursively. Names of T-boxes are the variable names. A set of t-boxes is called a t-table.

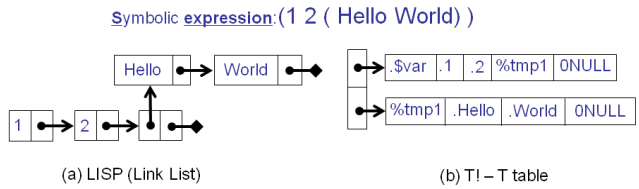

Fig 2: Representation of a typical s-expression in LISP and T!

In T!, a user defined function is just another variable whose contents must be in a specific format :

(FUNCTION::- parameter-type (formal-arguments) (single-function)).

e.g. SQR <- (FUNCTION::- EVAL_PARAM (x) (* x x)).

The function is invoked just as any built-in primitive function or operator. All functions are interpreted by a single subroutine which forms the T! interpreter. Functions support recursion as in LISP. In addition, a special predefined primitive function proc is available to support iteration.

T! scripts have very good pattern extraction capabilities along the lines of SNOBOL. The meta-information can be either stored as plain text file as done in this implementation or can be stored in a database like MySQL/Hypertable in s-expression format.

The use of T! language allows easy sandboxing of scripts in the P-MSE. Interaction of scripts with the code in external environment is restricted to the APIs provided by the script engine. The security can be enhanced, for example, remote file (or URL) access can be restricted to a black-list or white-list. T! language details and P-MSE APIs implemented as callback functions by the P-MSE are available at http://ssir.in/doc/pmse-sm.pdf.

3.3 Implementation of P-MSE

When the program is executed, the core T! libraries are loaded and initialization T! script is run.

Listing 1. Initialization T! script

```
(
( (// msed.t - MSED main configuration )
msed-main
( FUNCTION::- EVAL_PARAM ()
( proc ( ($res_tmp 0) $a $b $tmp_buff )
( - ( load-script 4ONQRY-DOPARALLEL google-handler
      google.t ) (- : - ) )
( - ( load-script 4ONQRY-DOPARALLEL yahoo-handler
      yahoo.t ) (- : - ) )
( - ( load-script 4ONQRY-DOPARALLEL bing-handler
      bing.t ) (- : - ) )
( - ( load-script 4ONQRY-DOPARALLEL dmoz-handler
      dmoz.t ) (- : - ) )
...
( - ( set-reset-global-value CACHE_ENABLE TRUE ) (- : - )
      )
( - ( set-reset-global-value DO_PARALLEL_SERACH TRUE
      ) (- : - ) )
...
( - ( load-script PRE-PROCESSOR pre-proc-handler pre-
      proc.t )(- : - ) )
...
( - ( set-reset-global-value PROXY_ADDR NO-PROXY ) (- :
      - ) )
)
))
)
```

The T! script snippet shown in Listing one calls the P-MSE API 'load-script' to define the script name (e.g. google.t) and the callback function (e.g. google-handler) to be called in parallel, when a query arrives.

Listing 2. Initialization of the service in C++ (in P-MSE).

```
T_TABLE t_table1;
LoadTLib(&t_table1,FLAVOR_LIB_FILE,"msed-flvrlib-
      bag");
...
t_table1.exec_t(t_table1.t_parsed_readr("(  msed-main
      )","tmp_main_var") ) ;
...
listen(msed.srv_socket,50);
do{
   int client_socket;
   if ((client_socket = accept (msed.srv_socket, NULL,
            NULL)) > 0){
      QUERY_HANDLER *qh = new
      QUERY_HANDLER();
      ...
      qh->serve_request_in_thread(); // Thread from here;
   }else MSED::error_log("Accept Failed");
}while(!is_exit);
...
```

Listing two C++ code snippet loads the 'flavor libraries' and executes the T! script for initialization. Then the P-MSE server listens for new HTTP connection. Upon receiving a search request, a new instance of query handler object is created. Rest of query handling is done in a new thread of execution.

Figure 3 shows a typical instance of the query handler. The query handler does pre-processing (e.g. query modification) by calling the appropriate T! script through the script interpreter library. Then cache hit verification is done. If a cache hit is found, its query id (Qid) is passed along with rest of the data for the scripts to process.

A T-table object is created as a local object and the corresponding T! script is loaded. The pre-defined callback function corresponding to each script is executed using the interpreter function. This is done in parallel using host thread mechanism. In the implementation, the C language pthread is used for P-MSE.

The P-MSE creates t-variables BROWSER-VARIABLE and SEARCH-RESULT which function as a virtual t-table. A virtual t-table is a T! variable which acts as a t-table. BROWSER-VARIABLE is populated by the P-MSE using HTTP request parameters. Each script is supposed to populate SEARCH-RESULT using results (title, url, search result snippet, rank position and engine name) parsed from collected search engine pages.

Algorithm 1 is a typical search script that checks for cache hit and loads data from cache if a hit is found in the cache hit variable passed by the P-MSE. Upon a cache miss, it constructs typical http request and collects the result in a t-bag (which is a string buffer). It then extracts the results using

string pattern recognition function and populates a virtual t-table named RESULTS.

Algorithm 1. Routine for interfacing with a typical SE (Implemented via T! script).

SubRoutine: Generate-Query
$SE-query-url ← 'Absolute URL'
for each $query-phrase, $connect-operator **in** BROWSER-VARIABLE
do
    $SE-query-url ← **concatenate** $SE-query-url $query-phrase '*Engine specific Boolean operator*'
End-for

SubRoutine: Parse-Results
Input-parameter:: $result-page
$Text-body ← **Call Library-Function** *Extact-String-Pattern* ($result-page,'*page pattern*')
Repeat
    {$url,$title,$stub} ← **Call Library-Function** *Extract-String-Pattern* (Text-body, 'pattern')
    Append {$url,$title,$stub} **to** *global variable* $RESULTS
Until ($url is **Empty**)

MainRoutne: Query-Handler
Global-variable: $BROWSER-VARIABLE, $RESULTS
$query ← **Call** Generate-Query
$result-page ← **Call *API*** http-get-file query
Call Parse-Results $result-page
Call *API* save-file $result-page **to** '*Cache Fle*'

In the given implementation, the Open Directory Project (also called DMOZ) is used as one of the search sources. A T! script of Algorithm 1 is used. When a query arrives, 'dmoz-handler' in the script is called. It can use its copy of BROWSER-VARIABLE to create the query. It then uses the P-MSE provided API 'http-get-file' to get the search result page in a t-bag. Then the script uses the function 'parse-results' to extract results. Using the library function 'tbag-find-replace', the contents of t-bag (the HTML result page) are parsed; and URL, title and stub are extracted. It then populates the virtual t-table RESULTS.

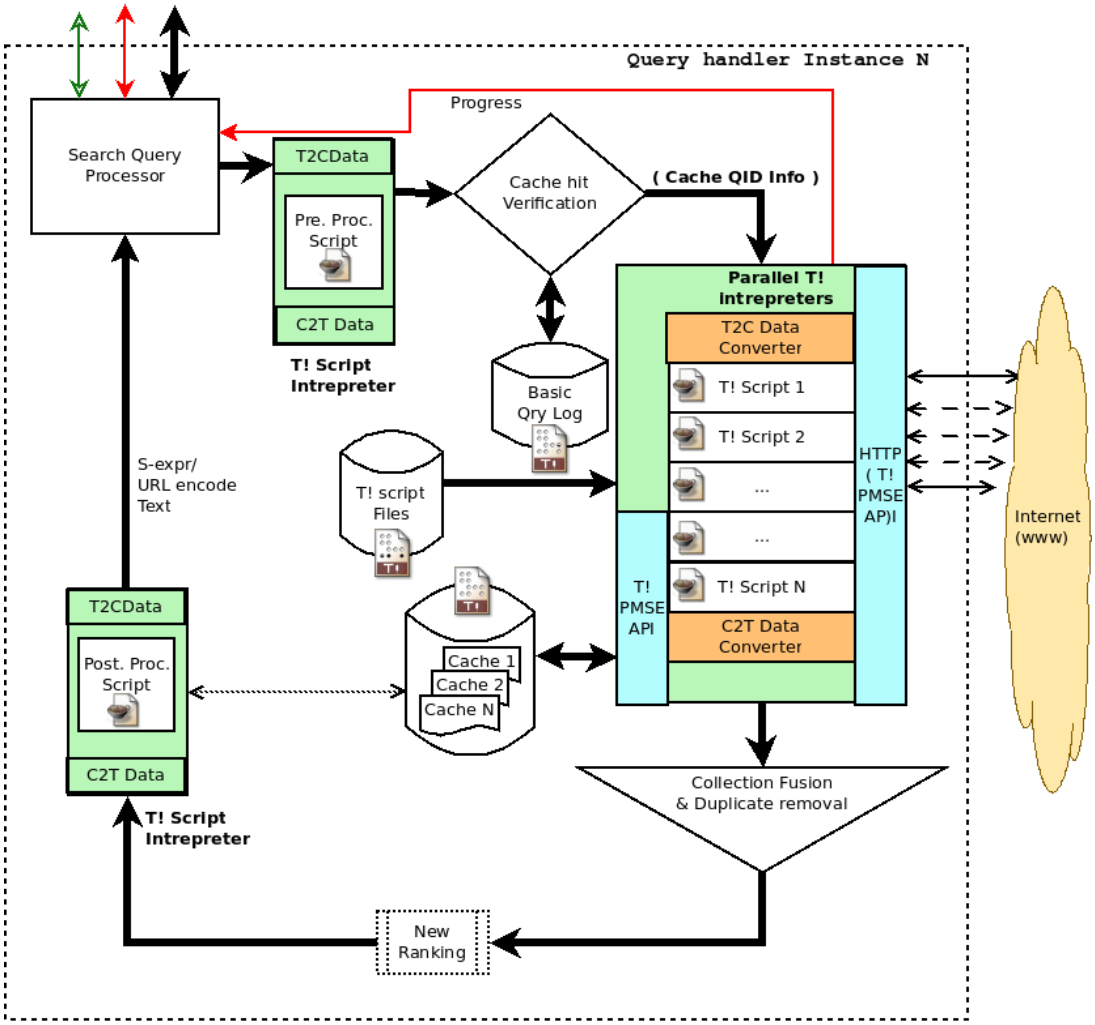

Fig 3: The logical architecture of a single instance of query handler

The results from all scripts using respective t-table object's RESULTS variable are fused with removal of duplicates. After simple re-ranking, it is passed to an optional post-processor T! script. The post-processor can do further processing like downloading actual documents and supplementing URL information. The result is passed to the program which issued the search query in the form of table of text encoded as s-expression or URL-Encoding which in turn passes to the user interface PHP script.

The host P-MSE adds a predefined callback script interpreter function, which is called when the 'call-extern' function is executed. It is used to implement various API functions used by the script. Any host program can use this call-back function to implement necessary APIs.

Algorithm 2. Running the T! scripts from P-MSE.

```
SubRoutine Run_T_Script:
Input-parameter:: $engine-N
create T-Table object
T-Table ← load T! script for $engine-N
create variable BROWSER-VARIABLE and RESULTS in T-
        Table
for each $Variable in 'list of Browser/HTTP variables'
do
    create variable in virtual t-table BROWSER-VARIABLE
end-do
call script interpreter function with T-Table object and script
        for $engine-N
for each $Variable in ' virtual t-table RESULTS '
do
    lock search-result array
    add variable in search-result array  with $Variable
    unlock search-result array
end-do

MainRoutine Do_Search
for each $Engine in 'list of engine'
do in parallel
     Run_T_Script $Engine
end-do
```

The main routine, for each SE registered in the initialization, calls the subroutine to execute the corresponding T! script. All scripts are executed in parallel. The subroutine initializes T! environment variables, loads and parses the script and calls the interpreter to run the script. On completion of the script, the environment variables are read and the search result array is updated (Algorithm 2).

4. APPLICATION OF P-MSE

The P-MSE developed in C++, with T! scripts for communication with SEs has been used to implement a general MSE called SSIR. Figure 4 illustrates the architecture of SSIR with a user interface developed in PHP running on an Apache web server.

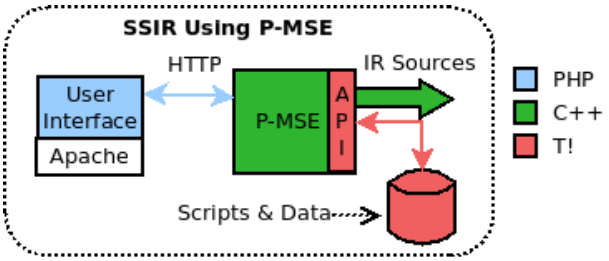

Fig 4: SSIR Meta search engine using P-MSE

The Meta Search Engine named SSIR, recursive acronym: SSIR for the Savvy Information Retriever, searches the popular SEs. Specific scripts for each search source page have been written. Engines like MS-Bing provide free API to get results in the form of XML. For other sources, raw HTML result pages are parsed. The live implementation can be accessed at http://ssir.in. Figure 5 is a screenshot of SSIR.

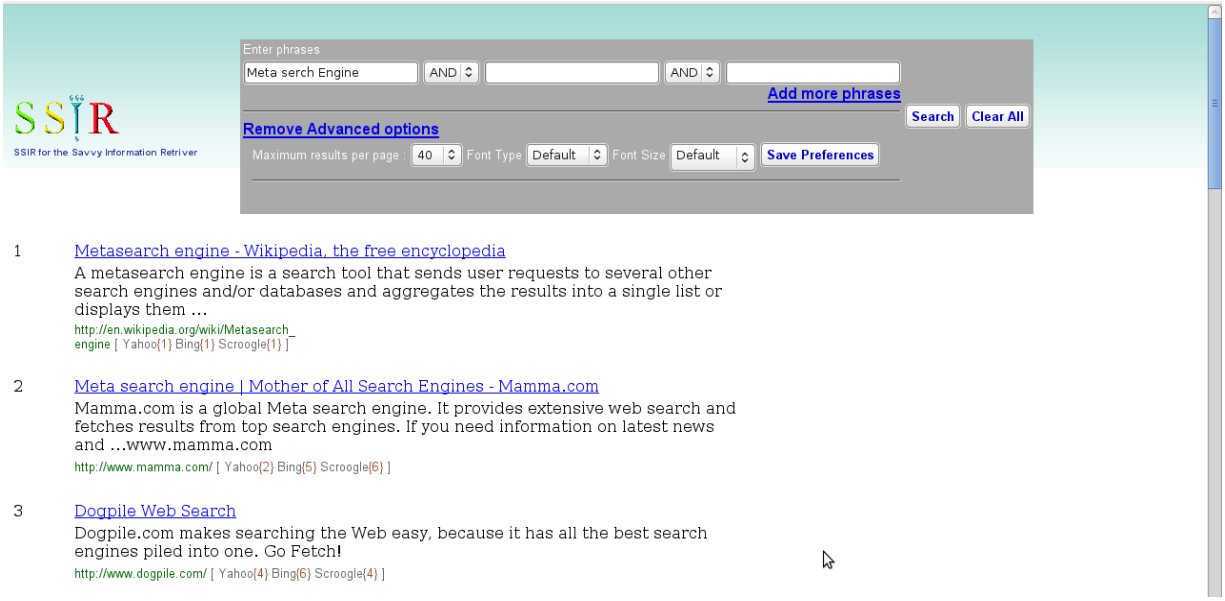

Fig 5: SSIR web site

It searches Yahoo, Bing, Google, Yandex and ODP taking a maximum of 10 phrases connected by boolean operators AND, OR or NOT. After fusion, duplicate URLs are merged and re-ranked using a ranking algorithm that fine tunes various fusion effects. The original position of the URL in the respective engines is also displayed with each result.

P-MSE can be used to create vertical MSEs and as an IR tool for studies like overlapping of Search Engines [23]. The query log and result cache from P-MSE based system can provide real web search data for IR studies, either as complementary or as an alternative to traditional data sets like TREC or FIRE.

Interested readers can contact the authors to get a copy of P-MSE for non commercial purpose.

5. CONCLUSION

P-MSE addresses the limitations highlighted in Section 2 by adding the programmability concept through lightweight concurrent embedded scripting, making it a powerful tool for MSE building. P-MSE has been created with a homoiconic, minimalistic LISP- like language called T!. This can make the creation of vertical MSEs much easier, helping the area specific information seeker to focus on his domain without having to surf through the general web.

P-MSE can be used as an IR tool to study new techniques by testing hypothesis on the real web in a fully automated way. Custom MSE/portal can be developed for institutions to search both their intranet as well as the web with features like censoring filters for educational institutions.

User querying can be simulated from the researcher's desktop using P-MSE with a front-end program for IR studies even without a live MSE implementation.

Apart from the MSE domain, the concepts described here can be used to embed scripting language in application software or web services to create customized, value added redistributable software products for secondary developers.

6. ACKNOWLEDGMENTS

The authors acknowledge the use of facilities at National Institute for Interdisciplinary Science and Technology, Thiruvananthapuram, Kerala, a constituent laboratory of CSIR (Council of Scientific and Industrial Research), India where this research has been carried out.

7. REFERENCES

[1] Chau, M., Chen, H., Qin, J., Zhou, Y., Qin, Y., Sung W., and McDonald, D. 2002. Comparison of Two Approaches to Building a Vertical Search Tool: A Case Study in the Nanotechnology Domain. In Proceedings of JCDL'02, Portland, OR, USA (July 2002)

[2] Chen, H., Fan, H., Chau M., and Zeng, D. 2003. Testing a Cancer Meta Spider; International Journal of Human-Computer Studies (IJHCS). 59(5), 755-776.

[3] http://trec.nist.gov/, Text REtrieval Conference Home Page, (Accessed 20 Feb. 2013).

[4] Majumder, P., Mitra, M., Pal, D., Bandyopadhyay, A., Maiti, S., Mitra, S., Sen A., and Pal, S. 2008. Text collections for FIRE. In Proceedings of the 31st annual international ACM SIGIR conference on Research and development in information retrieval (SIGIR '08). ACM, New York, NY, USA, 699-700.

[5] Manoj, M., and Jacob, E. 2008. Information retrieval on Internet using meta-search engines: A review, Journal of Scientific & Industrial Research. 67(10), 739-746.

[6] Selberg, E. W. 1999. Towards Comprehensive Web Search. Ph.D. thesis, University of Washington.

[7] Glover, E. J. 2001. Using Extra-Topical User Preferences to Improve Web-Based Metasearch. Ph.D. thesis, University of Michigan.

[8] Naz, T. 2009. Configurable Meta-search in the Human Resource Domain. PhD thesis, Vienna University of Technology.

[9] Voorhees, E. M., Gupta, N. K. and Johnson-Laird, B. 1995. The Collection Fusion Problem. In Proceedings of the Third Text Retrieval Conference (TREC-3); 95-104.

[10] Vogt, C. C. 1999. Adaptive Combination of Evidence for Information Retrieval. Ph.D. Thesis, University of California, San Diego.

[11] Montague, M. 2002. Metasearch: Data Fusion for Document Retrieval. Ph.D. thesis, Dartmouth College.

[12] Nassar M. O., and Kanaan, G. 2009. fCombMNZ: an Improved Data Fusion Algorithm. In Proceedings of the International Conference on Information Management and Engineering, Kuala Lumpur, Malaysia April 03-April 05, 461-464.

[13] Chowdhury, A. and Soboroff, I. 2002. Automatic evaluation of World Wide Web search services, In Proceedings of SIGIR-2002. 421-422.

[14] Jensen, E. C. 2006. Repeatable Evaluation of Information Retrieval Effectiveness in Dynamic Environments. Ph.D Thesis, Illinois Institute of Technology, Chicago, Illinois.

[15] Jensen, E. C., Beitzel, S. M., Chowdhury A., and Frieder, O. 2007. Repeatable evaluation of search services in dynamic environments, ACM Transactions on Information Systems (TOIS). 26(1), 1-38

[16] Nuray R., and Can, F. 2006. Automatic ranking of information retrieval systems using data fusion, Information Processing and Management. 42(3), 595-614.

[17] http://tools.seobook.com/authority-finder/myriad.txt, Myriad Meta Search Engine Source Code, (Accessed 20 Feb. 2013).

[18] http://sourceforge.net/projects/chalipa/, Chalipa Meta Search Engine - Open Source Project, (Accessed 20 Feb. 2013).

[19] http://web.archive.org/web/20070927230954/http://fravia.com/nbbw.c, Scroogle Source Code, (Accessed 20 Feb. 2013).

[20] Gulli A., and Signorini A. 2005. Building an Open Source Meta Search Engine. In Proceedings of 14th International World Wide Web Conference, (Chiba, Japan), 1004–1005.

[21] Ferragina P., and Gulli, A. 2008. A personalized search engine based on Web-snippet hierarchical clustering. Software: Practice and Experience, 38 (2), 189-225.

[22] Guha, R. V. 2005. Programmable search engine. U.S. patent (10 Aug 2005).

[23] Manoj, M., and Jacob. E. 2010. Analysis of Meta-Search engines using the Meta-Meta-Search tool SSIR. International Journal of Computer Applications. 1(6), 10–16