# Search Result Merging and Ranking Strategies in Meta-Search Engines: A Survey

**Hossein Jadidoleslamy**
**Department of Information Technology, Anzali International Branch, The University of Guilan**
**Zahedan, Sistan and Balouchestan, Zip: 9815687389, Iran**

**Abstract**
MetaSearch is utilizing multiple other search systems to perform simultaneous search. A MetaSearch Engine (MSE) is a search system that enables MetaSearch. To perform a MetaSearch, user query is sent to multiple search engines; once the search results returned, they are received by the MSE, then merged into a single ranked list and the ranked list is presented to the user. When a query is submitted to a MSE, decisions are made with respect to the underlying search engines to be used, what modifications will be made to the query and how to score the results. These decisions are typically made by considering only the user's keyword query, neglecting the larger information need. The cornerstone of their technology is their rank aggregation method. In other words, Result merging is a key component in a MSE. The effectiveness of a MSE is closely related to the result merging algorithm it employs. In this paper, we want to investigate a variety of result merging methods based on a wide range of available information about the retrieved results, from their local ranks, their titles and snippets, to the full documents of these results.

***Keywords: Search, Merge, Web, Meta-Search, MetaSearch Engine, Ranking.***

## 1. Introduction

MetaSearch Engines (MSEs) are tools that help the user identify such relevant information. Search engines retrieve web pages that contain information relevant to a specific subject described with a set of keywords given by the user. MSEs work at a higher level. They retrieve web pages relevant to a set of keywords, exploiting other already existing search engines. The earliest MSE is the MetaCrawler system that became operational since June 1995 [5,16]. Over the last years, many MSEs have been developed and deployed on the web. Most of them are built on top of a small number of popular general-purpose search engines but there are also MSEs that are connected to more specialized search engines and some are connected to over one thousand search engines.

MSE is a system that provides unified access to multiple existing search engines [5]. After the results returned from all used component search engines are collected, the MetaSearch system merged the results into a single ranked list. The major benefits of MSEs are their capabilities to combine the coverage of multiple search engines and to reach deep web. To increase the precision of the results, some MSEs do not always send the user's query to the same search engines. The existing methods assign scores according to objective criteria; but most current methods lack personalization [1,10].

In this paper, we investigate different result merging algorithms; The rest of the paper is organized as: In Section 2 motivation, In Section 3 overview of MSE, Section 4 provides scientific principles of MSE, Section 5 discusses about why MSE, Section 6 discusses architecture of MSE, Section 7 describes ranking aggregation methods, In Section 8 we express key parameters to evaluating the ranking strategies, Section 9 gives conclusions and Section 10 present future works.

## 2. Motivation

There are some primarily factors behind developing a MSE, are:

- The World Wide Web (WWW) is a huge unstructured corpus of information; MSE covers a larger portion of WWW;
- By MSE we can have the latest updated information and it increases the web coverage;
- Improved convenience for users;
- MSE provides fast and easy access to the desired search [5]; better retrieval effectiveness [2];
- MSE provides a broader overview of a topic [12];
- MSE has ability to search the invisible Web, thus increasing the precision, recall and quality of result;
- MSE makes the user task much easier by searching and ranking the results from multiple search engine;

- MSE provides a quick way to determine which search engines are retrieving the best match for user's information need [4].

## 3. Overview of a MetaSearch Engine

MSE search several engines at once; it does not crawl the web or maintain a database of web pages; instead, they act as a middle agent, passing the user's query simultaneously to other search engines or web directories or deep web, returning the results, collecting them, remove the duplicate links, merge and rank them into a single list and display it to the user [5,8]. Some samples of MSEs are Vivisimo, MetaCrawler, Dogpile, Mamma, and Turbo10.

### 3.1 Differences (Search vs. MetaSearch)

- MSE does not crawl the Web [2,4];
- MSE does not have a Database [4,10];
- MSE sends search queries to several search engines at once [2,5];
- MSE increased search coverage (but is limited by the engines they use with respect to the number and quality of results) and a consistent interface [6,12];
- MSE is an effective mechanism to reach deep web.

### 3.2 MetaSearch Engine Definitions

- Dictionary meaning for Meta: more comprehensive, transcending;
- The term MetaSearch is a list of search engines, but it is also used to describe the paradigm of searching multiple data sources in real time;
- A MSE allows you to search multiple search engines at once, returning more comprehensive and relevant results, fast [5,9];
- A search engine which does not gather its own information directly from web sites but rather passes the queries that it receives onto other search engines. It then compiles, summarizes and displays the found information;
- MSE is a hub of search engines/databases accessible by a common interface providing the user with results which may/may not be ranked independently of the original search engine/source ranking [6,10].

### 3.3 The Types of MetaSearch Engine

- MSEs which present results without aggregating them;
- Searches multiple search engines, aggregates the results obtained from them and returns a single list of results [1,3], often with duplicate removed;
- MSEs for serious deep digging.

### 3.4 MetaSearch Engine Issues

- Performing search engine/database selection [5,6];
- How to pass user queries to other search engines;
- How to identify correct search results returned from search engines; an optimal algorithm for implementing minimum cost bipartite matching;
- How to search results extraction, requiring a connection program and an extraction program (wrapper) for each component search engine [14];
- Expensive and time-consuming to produce/maintain wrapper programs [14];
- merging the results from different search sources;
- Different search engines produce result pages in different formats [6,8].

### 3.5 Principles of MetaSearch Engine

- Accept the User query;
- Convert the query into the correct syntax for underlying search engines, launch the multiple queries, wait for the result;
- Analyze, eliminate duplicates and merge results;
- Deliver the post processed result to the users.

## 4. Scientific Fundamentals

### 4.1 Search Engine Selection

To enable search engine selection, some information that can represent the contents of the documents of each component search engine needs to be collected first. Such information for a search engine is called the representative of the search engine [5,17]. The representatives of all search engines used by the MSE are collected in advance and are stored with the MSE. During search engine selection for a given query, search engines are ranked based on how well their representatives match with the query. Different search engine selection techniques often use different types of representatives. A simple representative of a search engine may contain only a few selected key words or a short description. This type of representative is usually produced manually but it can also be automatically generated [5]. As this type of representatives provides only a general description of the contents of search engines, the accuracy of using such representatives for search engine selection is usually low. More elaborate representatives consist of detailed statistical information [5] for each term in each search engine; include,

- The document frequency and collection frequency of each term in each search engine is used to compute the cue validity variance of each query term, which measures the skew of the distribution of the query term across all component search engines, to help rank search engines for each query [6,17];
- The adjusted maximum normalized weight of each term across all documents in a search engine is used to represent a search engine [5];
- The notion of optimal search engine ranking is proposed based on the objective of retrieving the m most similar documents with respect to a given query q from across all component search engines: n search engines are said to be optimally ranked with order [S1, S2, . . ., Sn] if for any integer m, an integer k can be found such that the m most similar documents are contained in [S1, . . ., Sk] and each of these k search engines contain at least one of the m most similar documents [5,9,17]. It is shown that a necessary and sufficient condition for the component search engines to be optimally ranked is to order the search engines in descending order of the similarity of the most similar document with respect to q in each search engine.

There are also techniques that create search engine representatives by learning from the search results of past queries. Essentially such type of representatives is the knowledge indicating the past performance of a search engine with respect to different queries.

## 4.2 Automatic Search Engine Connection

In most cases, the HTML form tag of a MSE contains all information needed to make the connection to the search engines. The form tag of each search engine interface is usually pre-processed to extract the information needed for program connection and the extracted information is saved at the MSE [5,17]. After the MSE receives a query and a particular search engine, among possibly other search engines, is selected to evaluate this query, the query is assigned to the name of the query textbox of the search engine and sent to the server of the search engine using the HTTP request method. After the query is evaluated by the search engine, one or more result pages containing the search results are returned to the MSE for further processing.

## 4.3 Automatic Search Result Extraction

A result page returned by a search engine is a dynamically generated HTML page. In addition to the search result records (SRRs) for a query, a result page usually also contains some unwanted information/links [5]. It is important to correctly extract the SRRs on each result page. A typical SRR corresponds to a retrieved document and it usually contains the URL, title and a snippet of the document. Since different search engines produce result pages in different format, a separate wrapper program needs to be generated for each search engine [5,14]. Most of them analyze the source HTML files of the result pages as text strings or tag trees to find the repeating patterns of the SRRs.

## 4.4 Results Merging

Result merging is to combine the search results returned from multiple search engines into a single ranked list. Early search engines often assigned a numerical matching score to each retrieved search result and the result merging algorithms were designed to normalize the scores returned from different search engines into values within a common range with the goal to make them more comparable [16]; Normalized scores will then be used to ranking all the search results [1,6]. Score normalization and rank aggregation may also take into consideration the estimated usefulness of each selected search engine with respect to the query [3,7], which is obtained during the search engine selection step. The normalized score of a result can be weighted by the usefulness score of the search engine that returned the result. This increases the chance for the results from more useful search engines to be ranked higher. When matching scores are not available, the ranks of the search results from component search engines can be aggregated using voting-based techniques.

Another result merging technique is to download all returned documents from their local servers and compute their matching scores using a common similarity function employed by the MSE; The results will then be ranked based on these scores [1,6]. The advantage of this approach is that it provides a uniform way to compute ranking scores. Its main drawback is the longer response time due to the delay caused by downloading the documents and analyzing them [17]. Most modern search engines display the title and snippet of each retrieved result [16]. These features can often provide good clues on whether or not the result is relevant to a query. As a result, result merging algorithms that rely on titles and snippets have been proposed recently. When titles and snippets are used to perform the merging, a matching score of each result with the query can be computed based on several factors such as the number of unique query terms that appear in the title/snippet and the proximity of the query terms in the title/snippet [1].

It is possible that the same result is retrieved from multiple search engines; such results are more likely to be relevant to the query; To help rank these results higher in the merged list, the ranking scores of these results from different search engines can be added up to produce the final score for the result. Then search results are ranked in descending order of the final scores [1,5].

## 5. Why Are MetaSearch Engines Useful?

### 5.1 Why MetaSearch?

- Individual Search engines do not cover all the web;
- Individual Search Engines are prone to spamming [5];
- Difficulty in deciding and obtaining results with combined searches on different search engines [6];
- Data Fusion (multiple formats supported) and take less effort of user.

### 5.2 Why MetaSearch Engines?

- General search engines have difference in search syntax, frequency of updating, display results/search interface and incomplete database [5,16];
- MSE improves the search quality with comprehensive, efficient and one query queries all;
- MSE is good for quick search results overview with 1 or 2 keywords;
- MSE convenient to search different content sources from one page.

### 5.3 Key Applications of MetaSearch Engine

- Effective mechanism to search surface/deep web;
- MSE provides a common search interface over multiple search engines [5,10];
- MSE can support interesting special applications.

### 5.4 General Features of MetaSearch Engine

- Unifies the search interface and provides a consistent user interface;
- Standardizes the query structure [5];
- May make use of an independent ranking method for the results [6];
- May have an independent ranking system for each search engine/database;
- MetaSearch is not a search for Meta data.

## 6. MetaSearch Engine Architecture

A MSE is a search tool that sends user requests to several other search engines/databases, aggregates the results, merges/renks them into a single list and displays them to user [5]. MSEs enable users to enter search criteria once and access several search engines simultaneously. This also may save (a lot of time) the user from having to use multiple search engines separately (by initiating the search at a single point).

MSEs create a virtual database; They do not compile a physical web database. Instead, they take a user's request, pass it to several heterogeneous databases and then compile the results in a homogeneous manner. No two MSEs are alike; some search only the most popular search engines while others also search lesser-known engines, newsgroups, and other databases [10]. They also differ in how the results are presented and the quantity of engines that are used. Some will list results according to search engine/database. Others return results according to relevance, often concealing which search engine returned which results. This benefits the user by eliminating duplicate hits and grouping the most relevant ones at the top of the list.

### 6.1 Standard Architecture

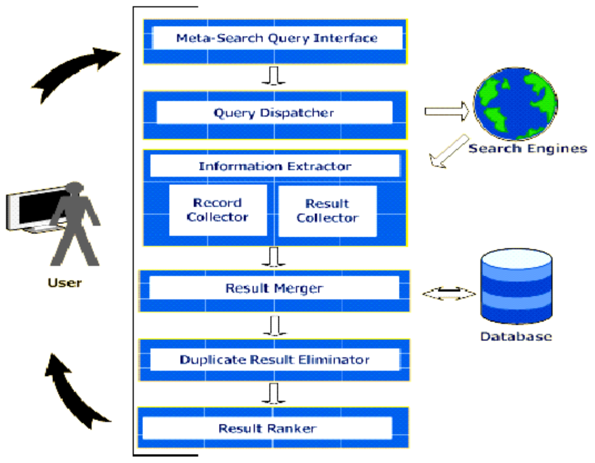

Fig. 1 How works a MSE?

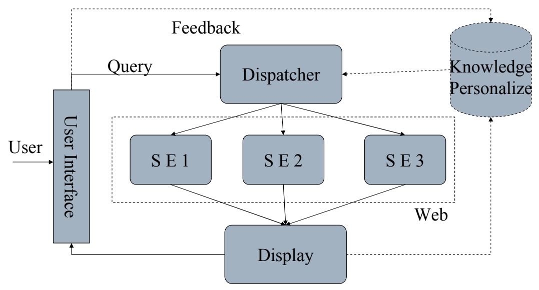

Fig. 2 Block diagram and components (block representation)

- User Interface: similar search engine interfaces with options for types of search and search engines to use;

- Dispatcher: generates actual queries to the search engines by using the user query; may involve choosing/expanding search engines to use;
- Display: generates results page from the replies received; May involve ranking, parsing and clustering of the search results or just plain stitching;
- Personalization/Knowledge: may contain either or both. Personalization may involve weighting of search results/query/engine for each user.

## 6.2 The Architecture of a MetaSearch Engine with Concerns User Preferences

Current MSEs make several decisions on be-half of the user, but do not consider the user's complete information need. A MSE must decide which sources to query, how to modify the submitted query to best utilize the underlying search engines, and how to order the results. Some MSEs allow users to influence one of these decisions, but not all three [4,5].

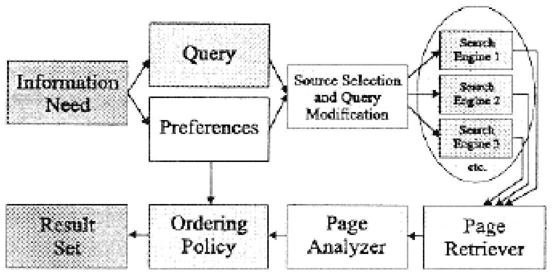

Fig. 3 The architecture of a MSE with user needs

A typical MSE, submits a user's query (with minor modifications) to a set of search engines, and returns the results in the order returned by the search engines. This order might not make sense if the user has a specific need. User's information needs are not sufficiently represented by a keyword query alone [4,10]. For example we can use of decision theory to ranking results from a single search engine while capturing more of a user's information need than a text query alone.

This architecture has an explicit notion of user preferences. These preferences or a search strategy, are used to choose the appropriate search engines (source selection), query modifications and influence the order the results (result scoring). Allowing the user to control the search strategy can provide relevant results for several specific needs, with a single consistent interface [4]. The current user interface provides the user with a list of choices. The specification of preferences allows users with different needs, but the same query, to not only search different search engines (or the same search engines with different "modified" queries), but also have results ordered differently [4]. Sometimes Even though users have different information needs, they might type the same keyword query, and even search some of the same search engines. This architecture guarantees consistent scoring of results by downloading page contents and analyzing the pages on the server, as opposed to relying on the reported scores and short summaries from the original search engines [1,4]. By downloading pages locally, we are assured to have the most recent version of any page and eliminating dead links and old content.

Table 1: Information needs categories

| **Name** | **Description** |
|---|---|
| Research papers | Detailed pages, preferably an actual article |
| Individual homepages | The homepages of the individual listed in the query |
| Organizational homepage | The homepages of the organization listed in the query |
| Current events, news recent | Recent articles, or content about the given query, with significant content |
| General introductory about | Getting started, references, "what is", etc |

### *6.2.1 Choosing a Search Engine (Source Selection)*

Currently, the search strategies are human generated. In future work includes use of learning techniques to improve the source selection decision. MSEs, allow the user some control of which search engines are chosen. Each of these engines has had experimental methods for automatic source selection based on the user query. A difficulty of automatic source selection is the metric used to compare sources. One metric is the average number of results returned for a given query. This metric is similar to the recall measure, if one assumes all returned results are relevant. Unfortunately, there is not necessarily a correlation between number of results returned and their usefulness.

Some MSE attempted to consider user satisfaction with results for some given query based on user feedback. Although this approach considers user statements about result values, it assumes that the user's valuations are mapped to the query, which is not true if there is more than a one-to-one mapping of needs to queries [4].

Other MSE contains a function for predicting the value of documents. This function is specific to the individual user. Given a reasonable model of user value, it is possible to determine how good any given source is for a given need (on average), as opposed to associating the "worth" of a given source to a query. By evaluating the worth of a source based on the need, not the query, it is possible to make reasonable judgments for previously unseen queries.

### *6.2.2 Modifying the Query (Query Modification)*

Query modification is one method of causing the underlying search engines to provide more valuable results for given information need [4]. One of the problems of a MSE is its dependence on the underlying search engines to provide a reasonable set of results. Just because a search engine contains very good results for the current user's need, there is no guarantee that those results will be returned for any given query [16]. To enhance the precision of the results and deal with the problems caused by search engine result limits, some MSEs allow query modification. Some types of modification performed: use of the search engine specific options; pre-pending terms; or appending terms.
One method for modification is using search engine specific options. Most search engines provide the ability to influence their result ordering or to add constraints. The choice of options depends on the user's information need, not only his keyword query [4].

The second modification adds keywords. Depending on the user's information need, it might be desirable to locate pages by type. By adding some keywords to a query, the precision can be significantly increased. It is important to note that query modification does not affect the scoring function; documents are scored based on the user-provided query and preferences, not modification [1,4].

Some MSEs allow query terms to be added before or after the provided query. For the information need category of "general resources", both types of modification are used. One modification is pre-pending "what is" to the query, while another adds the keywords "resources links" [4].

Query modifications can increase coverage. However, with a constraint, it is possible to get many valuable (both topically relevant and recent) results. The use of such options depends on the user's need, and is applied differently to different search engines.

### *6.2.3 Information Extractor and Result Merger/Ranker*

The information extractor component is responsible for extraction of results from the result pages. It consists of a Record collector component and a Result Collector component. The Record Collector component is responsible for identification of the record section from the result page and Result Collector component is responsible to extract the exact field from the identified record section. Information extraction often performed by using wrappers. Wrappers can be constructed manually, semi-automatically and automatically for record section identification [14]. For identification of record section, different approaches (include automatic, supervised or data mining wrapper generation) can be utilized. After identification of record section, a MetaSearch result identifier component is utilized for the extraction of exact results by using a domain ontology or information extraction tool [14]. Result extracted from different search engines need to be merged and then stored in a database or in XML format for future use. For schema matching and merging it is required to normalize the terms. Stemming can be utilized for term normalization process. The stemming process is useful to find similar terms by only considering the word stem in search engines, natural language and text processing.

The most important decision made by a MSE is results ordering. A typical search engine scores results based on the keywords in the query and the terms in the document [16]. Typical MSEs score documents based on the original scores returned from the search engines queried, running the risk that the actual pages are no longer relevant, or that the page scored high as a result of keyword spamming. Some MSEs download web pages and order them based on the full content. Other MSEs using an ordering policy defined by the user's preferences [4]; Different users, even with the same query and the same set of documents, will have results presented in an order meaningful to their individual need. MSEs also can use utility theory for evaluating the results; the ordering policy is "sort by value" where utility theory provides the mechanism for predicting the value. Each user-selected information need category has an associated additive value function of the form [17],
$U(dj) = \sum kWkVk(x_{jk})$;

Where Wk is the weight of the kth attribute and by convention totals one. Vk is the value function for the kth attribute; xjk is the level of the kth attribute for the jth document, and by convention [17]: any k, d: Vk (d) is member of [0, 1]; The page analyzer extracts the attributes (xjk) for every page. Table 2 lists several page specific attributes. Each information need category is described as a value function allowing a balance between several attributes [4,17].

Table 2: List of some of the page specific attributes and their description

| Name | Description |
|---|---|
| Word count | Number of words per pages |
| Homepage | A measure of the number of homepage like features present |
| Genscore | A measure of features indicative of a general page |
| researchpaper | A measure of features indicative of a researchpaper page |
| Imagecount | The number of unique images present on a page |
| Numkeywords | The number of keywords in the query matched on a page |

| | |
|---|---|
| Sectioncount | The number of sections on a page |
| Summary | An automatically generated summarization of the document |

Some MSEs that downloading all web pages can be time consuming, and presents an interface issue. If the ordering policy is sort by value, the standard approach is to wait until all results are downloaded, scored and then sort. An alternative is a dynamic interface that inserts each result as it is scored. If by coincidence the very first downloaded web page is perfect, it will be immediately available and the user can stop the search. Likewise, if there are mixtures of good and bad results, the better ones will be displayed on top. As a new result is downloaded and scored, it is immediately available for the user to see, thus reducing the effect of the latency, and improving over many MSEs that force the user to wait before seeing results. These MSEs provide the user with an optional JAVA applet that provides this dynamic-sorting and display functionality, plus the ability to change the ordering policy during or after searches.

## 6.3 Helios Architecture

In this section we describe the architecture of Helios. The Web Interface allows users to submit their queries and select the desired search engines among those supported by the system. This information is interpreted by the Local Query Parser & Emitter that re-writes queries in the appropriate format for the chosen engines. The Engines Builder maintains all the settings necessary to communicate with the remote search engines. The HTTP Retrievers modules handle the network communications. Once search results are available, the Search Results Collector & Parser extracts the relevant information and returns it using XML. Users can adopt the standard Merger & Ranker module for search results or integrate their customized one [12]. To achieve its high-performance, Helios utilizes a-sync I/O and parallel TCP connections, with the remote search engines. This is useful for two reasons: (i) the system is not overloaded with hundreds of threads; (ii) the connection cost is reduced to a few μsec, since parallel connections allow to retrieve data from one server while starting the connection to a second one, sending data to a third one. We remark that for a given query, it is possible to exploit both parallelisms among different search engines and within a single engine [12].

The integration of a new engine is simple: a configuration file is used to specify the engine parameters and a parser script provides the parser engine the necessary information to extract the relevant data. This MSE include a simple but efficient parsing language which allows searching strings, maintaining a cursor over a string, extracting substring, deleting or rewriting them. The language also provides some constructs such as if, until, jump and processes the search results in a fast and efficient way [12].

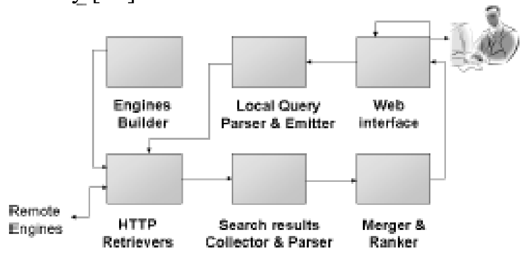

Fig. 4 The architecture of HELIOS MSE

## 6.4 A Tadpole Architecture

In this architecture, when a user issues a search request, multiple threads are created in order to fetch the results from various search engines. Each of these threads is given a time limit to return the results, failing which a time out occurs and the thread is terminated [5,11].

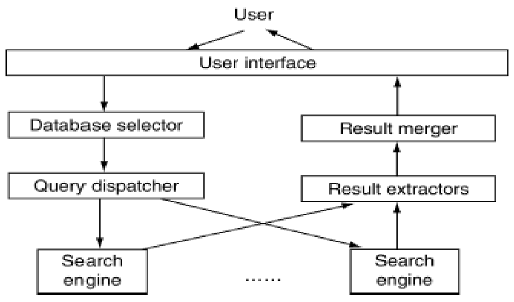

Fig. 5 Basic component architecture of a typical MSE

MSEs are web services that receive user queries and dispatch them to multiple crawl-based search engines. Then, they collect the returned results, reorder them and present the ranked result list to the end user [11]. The ranking fusion algorithms that MSEs utilize are based on a variety of parameters, such as the ranking a result receives and the number of its appearances in the component engine's result lists [15]. These parameters are being exploited to compute a weight (score) for each collected result. Better results classification can be achieved by employing ranking fusion methods that take into consideration additional information about a web page. Another core step is to implicitly/explicitly collect some data concerning the user that submits the query. This will assist the engine to decide which results suit better to his informational needs [4,11,15].

### 6.5 A TreeMap Architecture

Each process converts the given query to the format specific to the search engine it is dealing with. This request is sent to the search engine via the java URL object and the results are obtained in the form of a HTML page [11,16]. This HTML results page is parsed by the process and for each result the URL, title, description, rank and search source are stored; creating a result object. These results are entered into a TreeMap data structure with the key as the URL and the item as the result object. The GUI provides advanced search options for entering Boolean queries, Phrase searches, selecting the number of results per search engine and the selection of search engines to be queried [11].

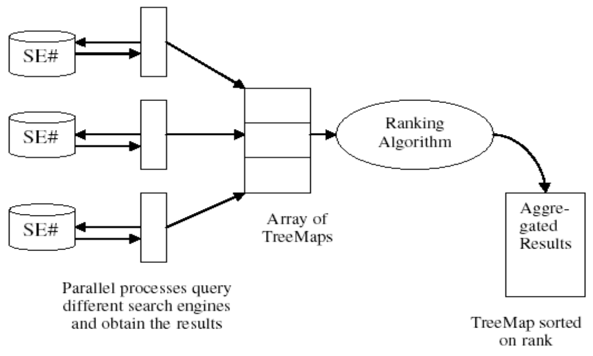

Fig. 6 The Architecture of TreeMap

## 7. Results Merging and Ranking Strategies

Some important techniques for ranking search results from different search engines in MSEs are:

- Normalizing the scores of search results and uniformly ranked them by the normalized scores [1,6];
- Some ways do not presuppose any information about these scores [1,7];
- The similarity measure in several search engines may be different. Therefore, normalization is required to achieve a common measure of comparison. Moreover, the reliability of each search engine must be incorporated in the ranking algorithm through a weight factor [6];
- Stress that the scores of various search engines are not compatible and comparable even when normalized. For example, notes that the same document receives different scores in various search engines and concludes that the score depends on the document collection used by a search engine;
- Some ranking algorithms are proposed which completely ignore the scores assigned by the search engines to the retrieved web pages [1]: bayes-fuse uses probabilistic theory to calculate the probability of a result to be relevant to the query, while borda-fuse is based on democratic voting [6,7];
- The contents associated with the SRRs can be used to rank/merge retrieved results. In other words, Merging based on titles, snippets, local rank and different similarity functions of retrieved results [6];
- Some of algorithms also consider the frequencies of query terms in each SRR, the order and the closeness of these terms;
- Approaches based on full document analysis.

Attention to this point that the comparison is not feasible even among engines using the same ranking algorithm and claims that search engines should provide statistical elements together with the results [7]. We want to investigate result merging algorithms for MSEs, to merge results from different search engines into a single ranked list. Most of the current generation search engines present more informative search result records (SRRs) of retrieved results to the user. A typical SRR consists of the URL, title and snippet of the retrieved document [6]. We are aware of only two works that utilize such SRRs. if a SRR contains enough information for merging so that the corresponding full document need not be fetched rather than a merging technique. The available evidences that can be used for result merging are identified; such as the document title, snippet, local rank, search engine usefulness. In follow, some algorithms based on different combinations of these evidences are proposed [6,7].

### 7.1 Take the Best Rank

In this algorithm, we try to place a URL at the best rank it gets in any of the search engine rankings [13]. That is [17],

- MetaRank (x) = Min (Rank1(x), Rank2(x), …, Rankn(x));

Clashes are avoided by an ordering of the search engines based on popularity. That means, if two results claim the same position in the MetaRank list, the result from a more popular search engine, is preferred to the result from a less popular one.

### 7.2 Borda's Positional Method

In this algorithm, the MetaRank of a URL is obtained by computing the Lp-Norm of the ranks in different search engines. That is [8,17],

- MetaRank(x) $=\sum$ (Rank1(x) $^{p}$, Rank2(x) $^{p}$, …, Rankn(x) $^{p}$) $^{1/p}$;

This algorithm has considered the L1-Norm which is the sum of all the ranks in different search engine result lists. Clashes are avoided by search engine popularity.

## 7.3 Weighted Borda-Fuse

In this algorithm, search engines are not treated equally, but their votes are considered with weights depending on the reliability of each search engine. These weights are set by the users in their profiles. Thus, the votes that the i result of the j search engine receive are [9,17],

- $V (r_{i,j}) = w_j * (\max_k (r_k)-i+1)$;

Where wj is the weight of the j search engine and rk is the numbers of results rendered by search engine k. Retrieved pages that appear in more than one search engines receive the sum of their votes.

## 7.4 The Original KE Algorithm

KE Algorithm on its original form is a score-based method [1]. It exploits the ranking that a result receives by the component engines and the number of its appearances in the component engines' lists. All component engines are treated equally, as all of them are considered to be reliable. Each returned ranked item is assigned a score based on the following formula [10],

- $W_{ke} = \sum^{m}_{i=1}(r (i)) / ((n)^{m} * (k/10 + 1)^{n})$;

Where ∑mi=1(r(i)) is the sum of all rankings that the item has taken, n is the number of search engine top-k lists the item is listed in, m is the total number of search engines exploited and k is the total number of ranked items that the KE Algorithm uses from each search engine. Therefore, it is clear that the less weight a result scores the better ranking it receives.

## 7.5 Fetch Retrieved Documents

A straightforward way to perform result merging is to fetch the retrieved documents to the MSE and compute their similarities with the query using a global similarity function. The main problem of this approach is that the user has to wait a long time before the results can be fully displayed. Therefore, most result merging techniques utilize the information associated with the search results as returned by component search engines to perform merging. The difficulty lies in the heterogeneities among the component search engines.

## 7.6 Borda Count

Borda Count is a voting-based data fusion method [15]. The returned results are considered as the candidates and each component search engine is a voter. For each voter, the top ranked candidate is assigned n points (n candidates), the second top ranked candidate is given n–1 points, and so on. For candidates that are not ranked by a voter (i.e., they are not retrieved by the corresponding search engine), the remaining points of the voter will be divided evenly among them. The candidates are then ranked on their received total points in descending order [13,15,17].

## 7.7 D-WISE Method

In D-WISE, the local rank of a document (ri) returned from search engine j is converted to a ranking score (rsij) by using the formula [6],

- $rs_{ij} = 1 – (r_i - 1) * S_{min} / (m * S_j)$ ;

Where Sj is the usefulness score of the search engine j, Smin is the smallest search engine score among all component search engines selected for this query, and m is the number of documents desired across all search engines. This function generates a smaller difference between the ranking scores of two consecutively ranked results retrieved from a search engine with a higher search engine score. This has the effect of ranking more results from higher quality search engines (with respect to the given query) higher. One problem of this method is that the highest ranked documents returned from all the local systems will have the same ranking score 1.

## 7.8 Merging Based on Combination Documents Records (SRRs)

Among all the proposed merging methods, the most effective one is based on the combination of the evidences of document such as title, snippet, and the search engine usefulness. These methods work as follows [1,2]:

At first, for each document, the similarity between the query and its title, and the similarity between the query and its snippet are computed. Then the two similarities are linearly aggregated as this document's estimated global similarity. For each query term, its weight in every component search engine is computed based on the Okapi probabilistic model [6]. The Okapi model requires the information of document frequency (df) of each term. Since the df information cannot be obtained in a MSE context, the df of the term t in search engine j is approximated by the number of documents in the top 10 documents returned by search engine j containing term t within their titles and snippets. The search engine score is the sum of all the query term weights of this search engine. Finally, the estimated global similarity of each result is adjusted by multiplying the relative deviation of its source search engine's score to the mean of all the search engine scores. Major general purpose search engines have a certain amount of overlaps between them. It is very possible that for a given query, the same document is returned from multiple component search engines. In this case, their (normalized) ranking scores need to be combined [1]. A number of linear combination fusion functions have been proposed to solve this problem include min, max, sum, average and etc [15].

## 7.9 Use Top Document to Compute Search Engine Score (TopD)

Let Sj denote the score of search engine j with respect to q. The TopD algorithm uses the similarity between q and the top ranked document returned from search engine j (denoted dij) to estimate Sj [6,7]. In general, the highest ranked document is the most relevant to the user query based on the search engine's ranking criteria. Its content can reflect how "good" the search engine is with respect to the user query. Fetching the top ranked document from its local server will introduce some extra network delay to the merging process, but we believe that this delay is tolerable since only one document is fetched from each used search engine for a query. For the similarity function, we tried both the Cosine function and the Okapi function. In Cosine function, the weight associated with each term in q and dij is the tf weight (we also tried tf*idf weight and the results are similar). The similarity between query q and dij using Okapi function is the sum of the Okapi weight of each query term T. The formula is [6],

- $\sum_{TEq} W * (((K_1 + 1) * tf) / (K + tf)) * (((K_3 + 1) * qtf) / (K_3 + qtf))$ ;
- With $W = Log ((N-n+0.5) /(n+0.5))$ and $K = K_1 * ((1-b)+b*(dl/avgdl))$ ;

Where tf is the frequency of the query term T within the processed document, qtf is the frequency of T within the query, N is the number of documents in the collection, n is the number of documents containing T, dl is the length of the document, and avgdl is the average length of all the documents in the collection. K1, k3 and b are the constants with values 1.2, 1,000 and 0.75, respectively [6]. Since N, n, and avgdl are unknown, we use some approximations to estimate them. The ranking scores of the top ranked results from all used search engines will be 1[1,6]. We remedy this problem by computing an adjusted ranking score arsij by multiplying the ranking score computed by above formula, namely rsij, by Sj [6], arsij = ∑ (rsij * Sj);

If a document is retrieved from multiple search engines, we compute its final ranking score by summing up all the adjusted ranking scores.

## 7.10 Use Top Search Result Records (SRRs) to Compute Search Engine Score (TopSRR)

This algorithm is the same as the TopD algorithm except that a different method is used to compute the search engine score. When a query q is submitted to a search engine j, the search engine returns the SRRs of a certain number of top ranked documents on a dynamically generated result page. In the TopSRR algorithm, the SRRs of the top n returned results from each search engine, instead of the top ranked document, are used to estimate its search engine score [6]. Intuitively, this is reasonable as a more useful search engine for a given query is more likely to retrieve better results which are usually reflected in the SRRs of these results. Specifically, all the titles of the top n SRRs from search engine j are merged together to form a title vector TVj, and all the snippets are also merged into a snippet vector SVj. The similarities between query q and TVj, and between q and SVj are computed separately and then aggregated into the score of search engine j [6],

- $S_j = C_1 * Similarity (q, TV_j) + (1 – C_1) * Similarity (q, SV_j)$;

Where for example C1 = 0.5 and n = 10. Again, both the Cosine similarity function with tf weight and the Okapi function are used. In the Okapi function, the average document lengths (avgdl) of the title vector TVj and the snippet vector SVj are estimated by the average length of the titles and the snippets of the top 10 results on the result page [6,7].

## 7.11 Compute Simple Similarities between SRRs and Query (SRRsim)

Since each SRR can be considered as the representative of the corresponding full document, we may rank SRRs returned from different search engines based on their similarities with the query directly using an appropriate similarity function. In the SRRsim algorithm, the similarity between a SRR (R) and a query q is defined as a weighted sum of the similarity between the title (T) of R and q and the similarity between the snippet (S) of R and q [6,7],

- $Sim(R , q) = C_2 * Similarity (q, T) + (1 – C_2) * Similarity (q , S)$ ;

Where, C2 is constant (C2 = 0.5). Again both the Cosine similarity function with tf weight and the Okapi function are tried. If a document is retrieved from multiple search engines with different SRRs (different search engines usually employ different ways to generate SRRs), then the similarity between the query and each such SRR will be computed and the largest one will be used as the final similarity of this document with the query for result merging.

## 7.12 Rank SRRs Using More Features (SRRRank)

The similarity function used in the SRRsim algorithm, no matter it is the Cosine function or the Okapi function, may not be sufficiently powerful in reflecting the true matches of the SRRs with respect to a given query [6]. For example, these functions do not take proximity information such as how close the query terms occur in the title and snippet of a SRR into consideration, nor does it consider the order of appearances of the query terms in the title and snippet. Intuitively, if a query contains one or more phrases, the order and proximity information has a significant impact on the match of phrases versus just individual terms. To better rank SRRs, this algorithm defines five features with respect to the query terms; that are [6,7],

- NDT: The number of distinct query terms appearing in title and snippet;
- TNT: total number occurrences of the query terms in the title and snippet;
- TLoc: The locations of the occurred query terms;
- ADJ: whether the occurred query terms appear in the same order as they are in the query and whether they occur adjacently;
- WS: the window size containing distinct occurred query terms.

For each SRR of the returned result, the above pieces of information are collected. The SRRRank algorithm works as follows [6]:

- All the SRRs are grouped based on the number of distinct query terms (NDT) in their title and snippet fields. The groups having more distinct terms are ranked higher;
- Within each group, the SRRs are further put into three subgroups based on the location of the occurred distinct query terms (TLoc). The subgroup with these terms in the title ranks highest, the subgroup with the distinct terms in the snippet and the subgroup with the terms scattered in both title and snippet;
- Finally, within each subgroup, the SRRs that have more occurrences of query terms (TNT) appearing in the title and the snippet are ranked higher. If two SRRs have the same number of occurrences of query terms, first the one with distinct query terms appearing in the same order and adjacently (ADJ) as they are in the query is ranked higher, and then, the one with smaller window size is ranked higher.

If there is any tie, it is broken by the local ranks. The result with the higher local rank will have a higher global rank in the merged list. If a result is retrieved from multiple search engines, we only keep the one with the highest global rank [3,6].

### 7.13 Compute Similarities between SRRs and Query Using More Features (SRRSimMF)

This algorithm is similar to SRRRank except that it quantifies the matches based on each feature identified in SRRRank so that the matching scores based on different features can be aggregated into a numeric value [1,3]. Consider a given field of a SRR, say title (the same methods apply to snippet). For the number of distinct query terms (NDT), its matching score is the ratio of NDT over the total number of distinct terms in the query (QLEN), denoted SNDT=NDT/QLEN. For the total number of query terms (TNT), its matching score is the ratio of TNT over the length of title, denoted STNT=TDT/TITLEN. For the query terms order and adjacency information (ADJ), the matching score SADJ is set to 1 if the distinct query terms appear in the same order and adjacently in the title; otherwise the value is 0. The window size (WS) of the distinct query terms in the processed title is converted into score SWS= (TITLEN–WS)/TITLEN. All the matching scores of these features are aggregated into a single value, which is the similarity between the processed title T and q, using this formula [6],

- $\mathrm{Sim}(T , q) = S_{NDT} + (1/QLEN) * (W_1 * S_{ADJ} + W_2 * S_{WS} + W_3 * S_{TNT})$ ;

This formula guarantees that titles containing more distinct query terms will have larger similarities. For each SRR, the similarity between the title and the query (Sim (T, q)) and the similarity between the snippet S and the query (Sim(S, q)) are computed separately first and then merged into one value as,

- $\mathrm{Similarity} = (TNDT/QLEN) * (C_3 * \mathrm{Sim}(T , q) + (1 - C_3) * \mathrm{Sim}(S , q))$ ;

Where TNDT is the total number of distinct query terms appeared in title and snippet. By multiplying by TNDT/QLEN, we guarantee that the SRR containing more distinct query terms will be ranked higher [6,7]. A genetic algorithm based training method is used to determine the values of the parameters involved in this method.

## 8. Evaluation Key Parameters for Ranking Strategies

### 8.1 Algorithmic Complexity (Time Complexity)

The positional methods take linear time [11,16].

### 8.2 Rank Aggregation Time

This parameter was measured with respect to each other and with normal search engines [3,11].

### 8.3 Overlap across Search Engines (Relative Search Engine Performance)

Among the top 10 results obtained for each query, there is overlap across multiple search engines' results. An interesting observation would be to find which search engines rank the overlapping results better. An intuition behind such a measure is that a search engine, which ranks the overlapping results, better, can be regarded as a better search engine considering that the overlapping results are more relevant [5].

### 8.4 Performance of the Various Rank Aggregation Methods

In evaluating the performance of the ranking strategies for all the queries, some ways have chosen precision as a good measure of relative performance; because all the ranking strategies work on the same set of results and try to get the most relevant ones to the top [11]. Hence, a strategy that has a higher precision at the top can be rated better from the user's perspective. These ways have plotted the precision of the ranking strategies with respect to both the number of search results and the recall [7,8]. In considering the recall, these ways have taken the total number of relevant documents based on user evaluation of all the top 10 results retrieved by each search engine [11].

#### *8.4.1 Precision with Respect to Number of Results Returned*

#### *8.4.2 Precision vs. Recall*

## 9. Conclusion

In this paper, we have presented an overview and some ranking strategies in MSEs. We also reported our study on how to merge the search results returned from multiple component search engines into a single ranked list; this is an important issue in MSE research. An effective and efficient result merging strategy is essential for developing effective MetaSearch systems [2]. We investigated merging algorithms that utilize a wide range of information available for merging, from local ranks by component search engines, search engine scores, titles and snippets of search result records to the full documents. We discuss methods for improving answer relevance in MSEs; propose several strategies for combining the ranked results returned from multiple search engines. Our study has several interesting results; that are:

- A simple, efficient and easily result merging algorithm can help a MSE significantly outperform the best single search engine in effectiveness [2];
- Merging based on the titles and snippets of returned search result records can be more effective than using the full documents of these results. This implies that a MSE can achieve better performance than a centralized retrieval system that contains all the documents from the component search engines;
- We have observed that the computational complexity of ranking algorithms used and performance of the MSE are conflicting parameters;
- A simply result merging algorithm can perform as well as more sophisticated ones;
- MSEs are useful, because
    - Integration of search results provided by different engines;
    - Comparison of rank positions;
    - Advanced search features on top of commodity engines;
    - A complete MSE can be used for retrieving, parsing, merging and reporting results provided by many search engines.

## 10. Future Works

Component search engines employed by a MSE may change their connection parameters and result display format anytime. These changes can make the affected search engines unusable in the MSE unless the corresponding connection programs and result extraction wrappers are changed accordingly [14]. How to monitor the changes of search engines and make the corresponding changes in the MSE automatically and timely is an area that needs urgent attention from MSE researchers and developers.

Most of today's MSEs employ only a small number of general purpose search engines. Building large-scale MSEs using numerous specialized search engines is another area that deserves more attention. Challenges arising from building very large-scale MSEs include automatic generation and maintenance of high quality search engine representatives needed for efficient and effective search engine selection, and highly automated techniques to add search engines into MSEs and to adapt to changes of search engines.

## References

[1] M. E. Renda, and U. Straccia, "Web Metasearch: Rank vs. Score based Rank Aggregation Methods", 2003.
[2] W. Meng, C. Yu, and K. Liu, "Building Efficient and Effective Metasearch Engines", In ACM Computing Surveys, 2002.
[3] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing and Aggregating Rankings with Ties", In PODS, 2004.
[4] J. E. Glover, S. Lawrence, P. W. Birmingham, and C. L. Giles, "Architecture of a Metasearch Engine that Supports User Information Needs", NEC Research Institute, Artificial Intelligence Laboratory, University of Michigan, In ACM, 1999.
[5] W. MENG, "Metasearch Engines", Department of Computer Science, State University of New York at Binghamton, 2008.

[6] Y. Lu, W. Meng, L. Shu, C. Yu, and K. Liu, "Evaluation of Result Merging Strategies for Metasearch Engines", 6th International Conference on Web Information Systems Engineering (WISE Conference), New York, 2005.
[7] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar, "Rank Aggregation Methods for the Web", Proceedings of ACM Conference on World Wide Web (WWW), 2001.
[8] R. Fagin, R. Kumar, M. Mahdian, D. Sivakumar, and E. Vee, "Comparing partial rankings", Proceedings of ACM Symposium on Principles of Database Systems (PODS), 2004.
[9] R. Fagin, R. Kumar, and D. Sivakumar, "Comparing Top k Lists", SIAM Journal on Discrete Mathematics, 2003.
[10] S. Souldatos, T. Dalamagas, and T. Sellis, "Captain Nemo: A Metasearch Engine with Personalized Hierarchical Search Space", School of Electrical and Computer Engineering, National Technical University of Athens, November, 2005.
[11] S. M. Mahabhashyam, and P. Singitham, "Tadpole: A Metasearch Engine Evaluation of Meta Search ranking Strategies", University of Stanford, 2004.
[12] A. Gulli, and A. Signorini, "Building an Open Source Meta Search Engine", University of Pisa, Informatica, May, 2005.
[13] J. Aslam, and M. Montague, "Models for Metasearch", In Proceedings of the ACM SIGIR Conference, 2001.
[14] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu, "Fully Automatic Wrapper Generation for Search Engines", World Wide Web Conference, Chiba, Japan, 2005.
[15] L. Akritidis, D. Katsaros, and P. Bozanis, "Effective Ranking Fusion Methods for Personalized Metasearch Engines", Panhellenic Conference on Informatics (IEEE), 2008.
[16] C. D. Manning, P. Raghavan, and H. Schutze, "Introduction to Information Retrieval", Cambridge University Press, 2008.
[17] J. Dorn, and T. Naz, "Structuring Meta-search Research by Design Patterns", International Computer Science and Technology Conference, San Diego, April, 2008.

## Author Biography

**H. Jadidoleslamy** is a Master of Science student at the Guilan University in Iran. He received his Engineering Degree in Information Technology (IT) engineering from the University of Sistan and Balouchestan (USB), Iran, in September 2009. He will receive his Master of Science degree from the University of Guilan, Rasht, Iran, in March 2011. His research interests include Computer Networks (especially Wireless Sensor Network), Information Security, and E-Commerce.