QuadSearch: A Novel Metasearch Engine

Leonidas Akritidis[1] **George Voutsakelis**[2]
Dimitrios Katsaros[1,2] **Panayiotis Bozanis**[2]

[1]Dept. of Informatics
Aristotle University
Thessaloniki, Greeee
[2]Dept. of Computer & Communication Engineering
University of Thessaly Volos, Greece
lakritid@mywork.gr, {gevoutsa, dkatsar, pbozanis}@inf.uth.gr

Abstract

Metasearch engines are increasingly becoming a very useful tool for Web information retrieval. In this paper we describe *QuadSearch*, an experimental metasearch engine that provides simultaneous access in four major conventional, crawler-based search engines. The heart of the new metasearch engine is based on two novel rank-based aggregation algorithms. The *QuadSearch* engine aims to combine speed, reliable rank aggregation method, "spam" free results, and detailed and enriched information. A publicly accessible interface for the new engine can be found at http://cheetah.csd.auth.gr/~lakritid/.

Keywords: Rank aggregation, top-k lists, Borda count, spam, bombing, metasearching, Web ranking, search engines, information retrieval.

1. Introduction

Nowadays, the most popular systems for digging for information in the Web are the search engines, either general purpose search engines, like Google (Page at al., 1999), or special purpose engines, like Medical World Search (http://www.mwsearch.com/). Although search engines are extremely popular among Web users, they can not achieve large coverage and high scalability.

The tool which rapidly gains acceptance by the users are the *metasearch engines* (Meng et al., 2002). Metasearch engines run simultaneously a user query across multiple *component search engines*, take the returned results and then aggregate them. The advantages of the metasearch engines are the following: a) they increase the search coverage of the Web, b) they solve the scalability problem of Web searching, c) they facilitate the exploitation of multiple search engines, and d) they improve the retrieval effectiveness.

The heart of a metasearch engine is the rank aggregation algorithm, which defines the final ranked result list from the individual results. Further process can be done in order to filter the results and allow the final result list of the metasearch engine to be relieved from unwanted, devious and undeservedly highly ranked Web pages. In a world which is frequently motivated by commercial interests, the user does not have a clear form of protection against the interests of individual search engines. Therefore, the metasearch engine should provide results to the user that are as free as they can be from paid listings and paid links.

1.1 Motivation and contributions

During the last years, the problem of paid listing within the retrieved results of search or metasearch engines has received a lot of attention. Generally, search engines resent less paid listings than metasearch engines. In addition, paid links at the major search engines are, in some way, separated from the main result list. In contrast, the metasearch engines do not have such delineation, making unclear which links are paid. Related to this is the problem of "spam" by authors of Web pages who attempt to achieve undeservedly high rank for their Web pages by exploiting defects of the ranking functions of search engines.

Secondly, there is no rank aggregation algorithm that bears a wide variety of parameters like the number of the search engines where a particular item appeared, the total number of exploited search engines or the size of the top-k list returned from each search engine. What we expect from a metasearch engine is to:

- consider as many parameters of these as possible,
- have anti-spam and anti-paid list properties,
- support personalization properties,
- refrain from using any training data in order to perform the rank aggregation, because, there is usually no evidence about the underlying data properties and their distributions, and
- do not count upon the *scores* of the individual search engine rankings in order to perform the rank aggregation, because, most of the search engines do not provide such scores.

Motivated from these requirements, we developed the *QuadSearch* metasearch engine (named after the fact that it currently capitalizes on four most popular search engines), which satisfies the above criteria. Firstly, it has two fast rank aggregation algorithms; a default algorithm and an improved version of it, enhanced with more antispam properties. Secondly, it allows the user to exploit whichever version of the rank algorithm he desires, and adjust a lot of the interface and the appearance parameters. We have implemented an experimental version of this metasearch engine, which although can be accessed at http://cheetah.csd.auth.gr/~lakritid/.

The rest of this article is organized as follows: in Section 2, we briefly review the relevant work on metasearch engines; in Section 3, which presents the main article ideas, we describe the new rank aggregation methods and in Section 4, we present the implementation issues behind the developed metasearch engine. Finally, in Section 5, we highlight the main features of the new metasearch engine and in Section 6, we conclude the paper.

2. Existing metasearch engines

The first metasearch engines were established back in 1996. The interested reader can find out the most popular metasearch engines of that period at (Baeza-Yates & Ribeiro-Neto, 1999, page 388). However, many problems occurred such as a lot of paid links inside organic results, the refusal of Google to cooperate with them and some fraud problems of pay-per-click search engines that led the metasearch engines to decline.

Now metasearch engines are coming back and a significant part of work is conducted for them (Fagin et al., 2006; Fagin et al., 2003). Researchers and developers work hard to prove that the results returned are defined by search algorithms and not by advertisers. They use classification and personalization techniques that conventional search engines do not have. In the sequel, we will mention a few metasearch engines[1] (Gulli & Signorini, 2005; Wu et al., 2001). A complete list of metasearch engines can be found at www.searchenginewatch.com/showPage?html.page=2156241.

Vivisimo (http://vivisimo.com/) and *Jux2* are clustering engines, which automatically organize the retrieved pages on-the-fly into categories (groups). *IxQuick* ranks the results based on the top-10 rankings a site receives from various search engines, *iBoogie* creates a list of categories related to search terms and *InfoGrid* provides direct links to major search engines and topical Web sites in different categories. *SearchOnline* offers a highly customizable interface, while *Kartoo* presents the results within a map that shows the most important sites and the linkage relationship between the results.

[1] The best and most popular meta search engines, www.searchenginewatch.com/showPage?html.page=2160791

3. The heart of the proposed novel result merging

The ideal scenario for result merging is when each search engine gives a complete result list of all the alternative items, related to the keyword terms of a given query, in the universe of alternatives. This can not be done and it is far too unrealistic for two main reasons: (i) search engines' coverage is different, and (ii) search engines limit access only to a portion of the complete result list. The worst scenario is when the result lists of component search engines don't have overlapping elements between them. In this case there is nothing that a rank aggregation algorithm can do.

Several rank aggregation methods have been used by metasearch engines (Lu et al., 2005; Meng et al., 2002, Renda & Straccia, 2003). In the 1990s most of the metasearch engines used *score-based* ranking methods to produce their results, i.e., they utilized the scores (weights) returned by the component search engines in order to fuse the component rankings. Moreover, many metasearch techniques applied normalization on these ranking scores in order to make them comparable.

Although score-based methods appear to be more effective for rank fusion, the absence of scores (or denial to reveal) from many search engines' rankings turned these methods problematic (Renda & Straccia, 2003); thus the *rank-based* fusion became the mainstream in present metasearch engines (Renda & Straccia, 2003). For instance, the Borda Count (Dwork et al., 2001; Renda & Straccia, 2003), which is a *voting-based* fusion method, is very popular among metasearch engines. Each result is a candidate and each search engine is the voter. Each candidate receives points from each voter according to its rank in the voter's list. For example, the top ranked candidate will receive *n* points, where *n* is the number of candidates. If a candidate is not in the top-k list of some voter then it will receive a portion of the remaining points of the voter (each voter has a fixed number of points available for distribution) or a constant number (0 or 1), depending on the variation of the method. The Borda Count method can be found in different versions, like the weighted Borda Count method (Souldatos et al., 2005), where each voter also takes a score and therefore his opinion for a candidate is not treated equally against other voters. Improved methods for ranking comparison and merging in the case of ties can be found at (Fagin et al., 2004, Fagin et al., 2006).

3.1 The ke method

In the sequel, we will present the rank fusion method of *QuadSearch* using only four component search engines (*Google, Yahoo!, Live Search, Ask Jeeves/Teoma*) since, for the present, our *QuadSearch* engine incorporates only these engines. The consideration of more engines is straightforward though.

In *QuadSearch*, we treat all four component search engines equally. The reason we do this is due to the following observations: (i) all of them are considered by experts as “major” search engines, (ii) during their lifetimes they have been proved reliable and (iii) most users and metasearch engines prefer them. The default rank aggregation method of *QuadSearch* is *rank-based*. Each returned ranked item is assigned a score based on the following formula:

$$ke = \frac{S}{n^m\left(\frac{k}{10}+1\right)^n} \qquad (1)$$

where *S* 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, *k* is the total number of ranked items that *QuadSearch* uses from each search engine. We named this weight as *ke*. The less the *ke* value for an item, the larger the final rank this item will take is. For example, consider the following listings of two search engines (see Table 1) for a particular query:

Table 1. Results of two search engines for a particular query.

Rank	**SE_1**	**SE_2**
1	U_1	U_{11}
2	U_2	U_{12}
3	U_3	U_{13}
4	U_4	U_{14}
5	U_5	U_4
6	U_6	U_{15}
7	U_7	U_{16}
8	U_8	U_{17}
9	U_9	U_{18}
10	U_{10}	U_{10}

Let us elaborate a bit more on this table. Firstly, we presume that we deal with the top-10 lists (*k*=10) from each conventional search engine (SE: search engine). Also we name the URLs of each result as U_i, in order to demonstrate the overlapping URLs more easily. As we can see, there are two overlapping URLs in the above listings, the U_4 which was ranked 4-th by SE_1 and 5-th by SE_2 and the U_{10} which was ranked 10-th by both search engines. All the others are found only in one of the two search engine top-10 lists. In Table 2 we can see the ranking scores of *ke* and Borda Count methods for each URL. In this point we should mention a compact. We assume that when two URLs have the same score, then the URL that is in both top-10 lists will be ranked first, otherwise the URL of the first search engine will be ranked first.

Table 2. Ranking scores of ke and Borda Count methods.

URL	**ke**	**ke rank**	**BC_{18}**	**BC rank**
U_1	0.5	1	18	3
U_2	1	4	17	5
U_3	1.5	7	16	7
U_4	0.5625	3	29	1
U_5	2.5	10	14	10
U_6	3	11	13	11
U_7	3.5	13	12	13
U_8	4	15	11	15
U_9	4.5	17	10	17
U_{10}	1.25	6	18	2
U_{11}	0.5	2	18	4
U_{12}	1	5	17	6
U_{13}	1.5	8	16	8
U_{14}	2	9	15	9
U_{15}	3	12	13	12
U_{16}	3.5	14	12	14
U_{17}	4	16	11	16
U_{18}	4.5	18	10	18

Finally in Table 3 we can see the final top-10 lists of the two methods.

Table 3. Final top-10 lists of *ke* and BC methods.

Rank	**SE_1**	**SE_2**	**ke result list (top10)**	**BC result list (top10)**
1	U_1	U_{11}	U_1	U_4
2	U_2	U_{12}	U_{11}	U_{10}
3	U_3	U_{13}	U_4	U_1
4	U_4	U_{14}	U_2	U_{11}
5	U_5	U_4	U_{12}	U_2
6	U_6	U_{15}	U_{10}	U_{12}
7	U_7	U_{16}	U_3	U_3
8	U_8	U_{17}	U_{13}	U_{13}

9	U_9	U_{18}	U_{14}	U_{14}
10	U_{10}	U_{10}	U_5	U_5

We must stress some differences between Borda Count and the *ke* method:

- The Borda Count method takes into consideration the total number of candidates, while *ke* takes into consideration the number of voters.
- Some Borda Count variations assign scores to each and every candidate; a candidate which is not included in the top-k list of a particular search engine takes a part of the remaining points. This does not hold for the *ke* method. In the *ke* method, a candidate will be assigned a score only when it is contained in the top-k list of a particular search engine, otherwise its score is zero.
- The *ke* method takes into consideration the total number of exploited search engines, the number of search engines where a candidate has been appeared and the size of the top-k list.
- The *ke* method has better "resolution", in the sense that the possibility of two scores being the same is less than that of the Borda Count. For example, in Table 2 we can see that Borda Count assigned three URLs with the same score (U_1, U_{10}, U_{11} with score 18) while the *ke* method has given U_{10} a different score.
- The lower the *ke* weight an item has the higher will be ranked in the final result list. In Borda Count holds the opposite.

3.2 Antispam version of ke method

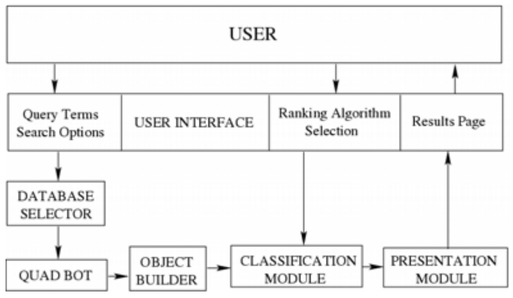
Informaly, we say that a search engine has been spammed by a page in its result list when it ranks the page too highly with respect to the other pages, according to the view of a "typical" (average) user. This is unavoidable for search engines, because their ranking algorithms have "defects" that can be exploited by Web page developers in order to achieve an undeservedly high page rank. Thus, if a page spams all or even most of the search engines, then the metasearch engine could not defeat this problem as well, because the aggregation function would work with bad data.

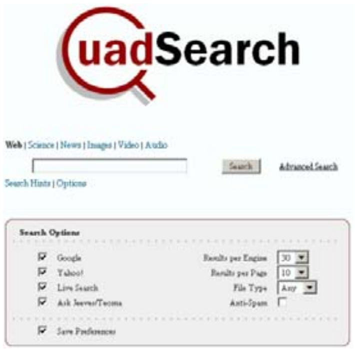
In *QuadSearch* we gave to the users the option to use an antispam version of *ke* method. This method takes into consideration the *Condorcet Criteria* (Francis, 2005). In the context of metasearching, these criteria tell us, in a few words, that an item which is enlisted in the top-k lists of some search engines should be ranked above an item that is ranked in the top-k lists of fewer search engines. The *QuadSearch* engine attempts to satisfy the intuition that if a page spams fewer than half of the search engines, then the majority of search engines will prefer a relatively good page to a spam page. The following pseudocode describes the antispam version:

1. *Find which items appear in more than half pages (let the number of these items be c).*
2. *Apply the ke method for these items.*
3. *Position them in QuadSearch result list, starting at rank 1.*
4. *Apply the ke method for the rest of the items.*
5. *Position them in QuadSearch result list, starting at rank 1+c.*

4. System Implementation

In the following paragraphs, we describe the technical issues regarding the implementation of the new metasearch engine. The most significant modules of *QuadSearch* are the Quad Bot, the Object Builder, the Classification Module and the Presentation Module. These modules are described in the next subsections. A schematic diagram of the architecture is depicted in the left part of Figure 1.





Search Options

- [✓] Google
- [✓] Yahoo!
- [✓] Live Search
- [✓] Ask Jeeves/Teoma

Results per Engine: 30
Results per Page: 10
File Type: Any
Anti-Spam: [ ]

- [✓] Save Preferences

Figure 1. (Left) Architecture of QuadSearch. (Right) QuadSearch's homepage.

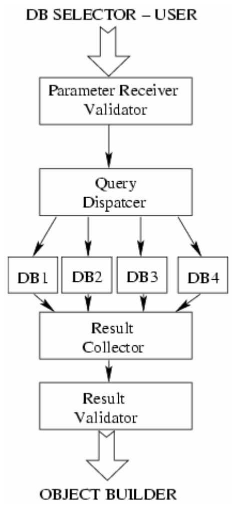
User interface and database selector. The user can switch among these features from either the home page or the results page. Regarding the database selector, the user has the option to choose which search engines will be exploited (see the right part of Figure 1. Also, the user can select the search engines that will participate in the search process, the number of results to be retrieved per resource, the number of results that will be displayed per page etc. The interface includes an extra option to prevent spam records from entering the *ke* list. Finally, in the options page the user can select the ranking algorithm (*ke* or Borda Count).

Quad bot. The Quad Bot receives its inputs from both the database selector and the user interface. It is responsible for validating the input data and parameters, passing the query to the selected databases and collecting the results. Its internal structure is depicted in the left part of Figure 2.

Parameter Receiver/Validator. It accepts all the data coming from the database selector and the user. The validation process includes transformation of the inputs in a way that can be sent to the search engines.

Query Dispatcher. The Query Dispatcher is the Quad Bot's heart. It gets the validated data and creates http requests to the selected search engines. This is the slowest procedure of the whole system; its speed depends on the number of the invoked search engines, the requested results, the server's Internet connection, etc. We have accelerated this procedure by submitting all the requests to the search engines simultaneously. To achieve that, we had to employ the *libcurl* library with cURL (client URL) extensions 7.16.0 for PHP 5.1 that support multiple connections at a time. By building the Query Dispatcher this way, we managed to shrink the idle time to no more than 1 or 2 seconds.

Result Collector. The Result Collector embraces the http responses transmitted by the search engines. Each involved search engine must respond to the Query Dispatcher's request, by sending the source code of its result page. The module retrieves the Rank, the URL, the Title and the Abstract for each candidate. When it receives all the information, it stores it temporary and sends it to the next module for validation.



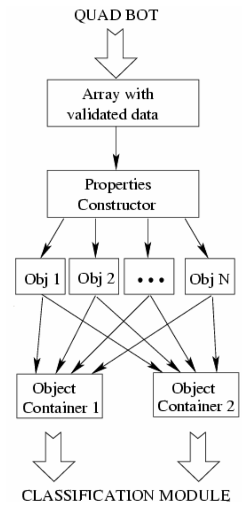


Figure 2. (Left) Quad Bot's structure. (Right) Object Builder's architecture.

Result Validator. The Result Validator is the most complex compartment of this module, as it performs multiple conversions to the collected data. The URL validation part is responsible for the appropriate formatting of the collected URLs, so that the overlapping candidates could be correctly detected later.

Object Builder. The Object Builder is the bridge between the Quad Bot and the Classification module; its architecture is depicted in the right part of Figure 2.

Array with validated data. The Object Builder's input is the array that the Quad Bot produces. It contains all the collected results that passed the Result Validator's checks.

Property Constructor. This module implements a class that describes the properties of our objects. The properties that are being assigned to the objects are the URL, the Title, the Abstract and one to four Rankings (one for each selected search engine).

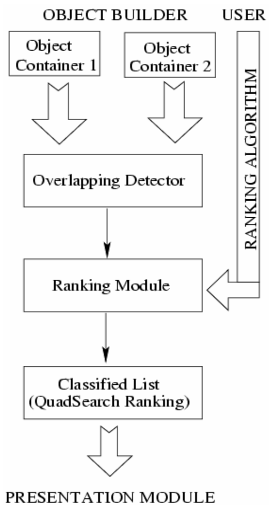
Object Containers 1 and 2. In this compartment, all the objects (the results) are being transferred to two new, identical object containers. The results enter the containers in groups. The first group consists of the results that the first search engine returns, the second group consists of the results that the second engine returns, etc. These containers will be the main tool in our effort to compare the search engine rankings and generate the final ranked list.

Classification Module. The Classification Module accepts the two result containers from the Object Builder and performs the result ranking according to the selected ranking algorithm. Its architecture is illustrated in the left part of Figure 3.

Overlapping Detector. This section is responsible for detecting the overlapping candidates and for creating the final candidate list. It receives input from the two object containers and compares each object from the first container, to all objects from the second container. Finally, the procedure constructs one container that holds all candidates, overlapping or not.

Ranking Module. The Ranking Module accepts the candidate container that the Overlapping Detector constructs, and the ranking algorithm that the user selected. The Ranking Module will apply the *ke* algorithm by default, unless the user selects another supported algorithm. Next, it computes the weight factors and/or the Borda Scores. Finally, it sorts the candidate list on ascending (for weight factors) or descending (for Borda Scores) order and passes this list to the Presentation Module.

Presentation Module. The task of this module is to construct the result page that will be presented to the user. In comparison to the other system compartments, this one has the simplest architecture. In the right part of Figure 3, we illustrate a schematic diagram of its internal structure.



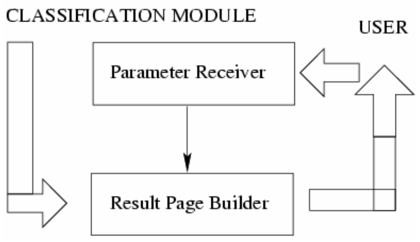


Figure 3. (Left) Classification Module. (Right) Presentation Module.

5. Innovative Features

In this section, a quick walkthrough of Quad Search's innovative features is presented.

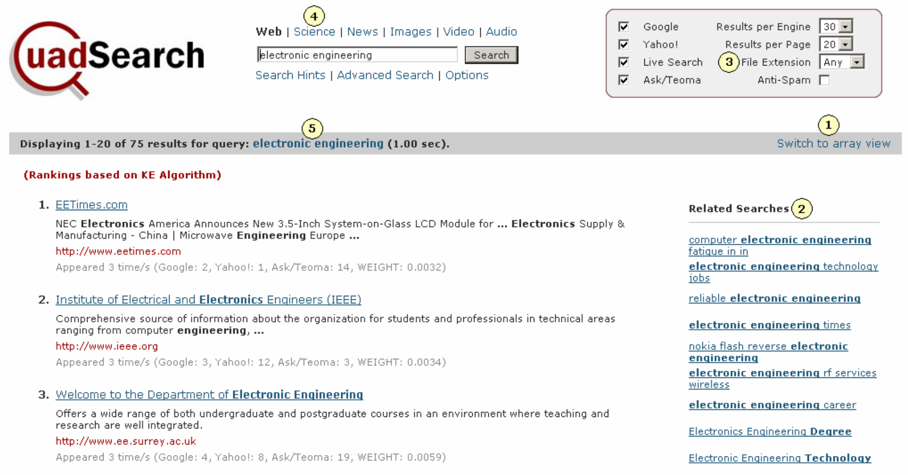
1. Classic/Array View Switch. The user is able to view the results in the classic way, but can also select the array view that provides an easier way of comparing the collected results.

2. Related Searches. Apart from the desired results, the Quad Bot is capable of grabbing almost everything from the results' pages that the exploited search engines transmit. In order to provide more specific results, the search engines prompt their users to submit the queries that they propose. The Quad Bot can fetch these query strings and present it to the result page through the Presentation Module.

3. File Type Filter. Many users tend to search the Web for specific file types (e.g., Adobe Acrobat or Microsoft Word files) and QuadSearch includes a similar feature. The user can select one of the most popular file extensions and perform a Web search. At this time, the QuadSearch engine supports searches for the following file formats: PDF, DOC, XLS, PS, RTF and PPT.

4. Search for Scientific Articles. QuadSearch supports searches for scientific articles in the richest scientific databases. Google Scholar is also included in these databases. This type of search can be accessed from the “Science” link and will return papers, technical reports and books related to the query terms.

5. Query String Explosion Feature. This feature (see Figure 4) splits the query string to its search terms and gives the user the ability to perform `single term' searches. For example, the query string `electronic engineering' is being split to the terms `electronic' and `engineering'. By clicking on any of these words QuadSearch will perform a Web search.


uadSearch
4
Web | Science | News | Images | Video | Audio
Search
Search Hints | Advanced Search | Options
3
5
1
Switch to array view
2

electronic engineering

✓ Google — Results per Engine: 30
✓ Yahoo! — Results per Page: 20
✓ Live Search — File Extension: Any
✓ Ask/Teoma — Anti-Spam: ☐

Displaying 1-20 of 75 results for query: electronic engineering (1.00 sec).

(Rankings based on KE Algorithm)

1. EETimes.com
   NEC **Electronics** America Announces New 3.5-Inch System-on-Glass LCD Module for **...** **Electronics** Supply & Manufacturing - China | Microwave **Engineering** Europe **...**
   http://www.eetimes.com
   Appeared 3 time/s (Google: 2, Yahoo!: 1, Ask/Teoma: 14, WEIGHT: 0.0032)

2. Institute of Electrical and **Electronics** Engineers (IEEE)
   Comprehensive source of information about the organization for students and professionals in technical areas ranging from computer **engineering**, **...**
   http://www.ieee.org
   Appeared 3 time/s (Google: 3, Yahoo!: 12, Ask/Teoma: 3, WEIGHT: 0.0034)

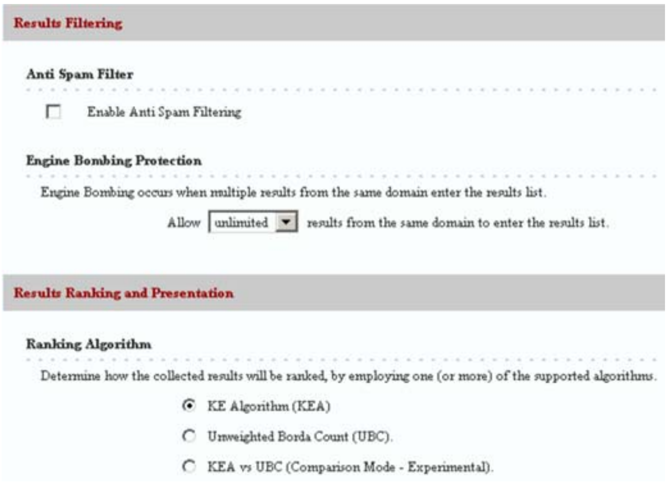
3. Welcome to the Department of **Electronic Engineering**
   Offers a wide range of both undergraduate and postgraduate courses in an environment where teaching and research are well integrated.
   http://www.ee.surrey.ac.uk
   Appeared 3 time/s (Google: 4, Yahoo!: 8, Ask/Teoma: 19, WEIGHT: 0.0059)

Related Searches

computer **electronic engineering** fatigue in in
electronic engineering technology jobs
reliable **electronic engineering**
electronic engineering times
nokia flash reverse **electronic engineering**
electronic engineering rf services wireless
electronic engineering career
Electronics Engineering **Degree**
Electronic Engineering **Technology**

Figure 4. The results' page with QuadSearch's innovative features.

6. Ranking Algorithm Selector. This feature (see Figure 5) is only accessible from the options page and provides the user with the facility to determine how the collected results will be ranked, by employing one (or more) of the supported algorithms. At this time, QuadSearch supports our *ke* Algorithm and the Borda Count method. It also provides a third option that utilizes both algorithms and presents the results in array view (comparison mode). It is in our intentions to include more ranking algorithms in the system (e.g., Markov Chains).


Results Filtering

Anti Spam Filter

☐ Enable Anti Spam Filtering

Engine Bombing Protection

Engine Bombing occurs when multiple results from the same domain enter the results list.

Allow [unlimited] results from the same domain to enter the results list.

Results Ranking and Presentation

Ranking Algorithm

Determine how the collected results will be ranked, by employing one (or more) of the supported algorithms.

● KE Algorithm (KEA)

○ Unweighted Borda Count (UBC).

○ KEA vs UBC (Comparison Mode - Experimental).

Figure 5. Options with the Ranking Selector and the Engine Bombing Protection.

7. Engine Bombing Protection. When various search resources are being exploited, a possibility that many similar results will enter the result's list always exists. This phenomenon is called *engine bombing*. For example, it is not very informative and useful for a user to submit a query and receive five or more results from the same domain in the top, say, twenty listing. Thus, we developed a feature (which can be enabled or disabled) to prevent multiple results coming from the same domain to enter into the result list; alternatively the user can select the maximum number of such results.

6. Concluding remarks and future work

In this article, we considered the issue of developing a new metasearch engine to assist in the process of Web information retrieval. The main motivation to develop this novel metasearch engine was the common intuition that a rank aggregation algorithm should: a) be related to the comparison of the top-k lists of each conventional search engine, and b) deal with the problem of the spam into metasearch result lists. Thus, we came up with a pair of new methods for rank aggregation, i.e., the *ke* method and its antispam version. We injected some new parameters, like the number of the top-k lists that a page appears, the total number of exploited search engines and the size of the top-k lists. The new metasearch engine is named *QuadSearch*, and it is publicly available at http://cheetah.csd.auth.gr/~lakritid/. For the near future, we are going to implement anonymous personalization techniques, further result filtering thorough search hints. Furthermore, we are orientated towards making *QuadSearch* a scientific tool by implementing most of the bibliometric indexes based on the h-index (Katsaros et al., 2007) and developing some new variations of it.

References

Baeza-Yates, R., Ribeiro-Neto, B. (1999), *Modern Information Retrieval*, Addison-Wesley and ACM Press.

Dwork, C., Kumar, R., Naor, M., Sivakumar, D. (2001), *Rank aggregation methods for the Web*, Proceedings of ACM Conference on World Wide Web (WWW), pp. 613–622.

Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., Vee, E. (2004), *Comparing partial rankings*, Proceedings of ACM Symposium on Principles Of Database Systems (PODS), pp. 47–58.

Fagin, R., Kumar, R., Mahdian, M., Sivakumar, D., Vee, E. (2006), *Comparing partial rankings*, SIAM Journal on Discrete Mathematics, vol. 20, pp. 628–648.

Fagin, R., Kumar, R., Sivakumar, D. (2003), *Comparing top k lists*, SIAM Journal on Discrete Mathematics, vol. 17, no. 1, pp. 134–160.

Francis, N. (2005), *Voting as a method for rank aggregation and spam reduction on the Web*, Undergraduate senior thesis (CPSC 490). Department of Computer Science, Yale University.

Gulli, A., Signorini, A. (2005), *Building an open source meta-search engine*, Proceedings of ACM Conference on World Wide Web (WWW), pp. 1004–1005.

Katsaros, D., Sidiropoulos, A., Manolopoulos, Y. (2007), *Age-decaying h-index for Social Networks of Citations*, Proceedings of Workshop on Social Aspects of the Web (SAW).

Lu, Y., Meng, W., Shu, L., Yu, C., Liu, K.-L. (2005), *Evaluation of result merging strategies for metasearch engines*, Proceedings of IEEE International Conference on Web Information Systems Engineering (WISE), pp. 53–66.

Meng, W., Yu, C., Liu, K.-L. (2002), *Building efficient and effective metasearch engines*, ACM Computing Surveys, vol. 34, no. 1, pp. 48–89.

Page, L., Brin, S., Motwani, R., Winograd, T. (1999), *The PageRank citation ranking: Bringing order to the Web*, Stanford University Technical Report, TR-1999-66.

Renda, M.E., Straccia, U. (2003), *Web metasearch: Rank vs score based rank aggregation methods*, Proceedings of ACM International Symposium on Applied Computing (SAC), pp. 841–846.

Souldatos, S., Dalamagas, T., Sellis, T. (2005), *Sailing the Web with Captain Nemo: A personalized metasearch engine*, Proceedings of the ICML workshop: Learning in Web Search (LWS).

Sugiura, A., Etzioni, O. (2000), *Query routing for Web search engines: Architecture and experiments*, Computer Networks, vo. 33, no. 1–6, pp. 417–429.

Wu, Z., Meng, W., Yu, C., Li, Z. (2001), *Towards a highly-scalable and effective metasearch engine*, Proceedings of ACM Conference on World Wide Web (WWW), pp. 386–395.