Buddy – Harnessing the Power of the Internet

Douglas Boulware[1], John Salerno[1], Nina Zumel[2], Michael Manno[1]
[1]Air Force Research Laboratory, Rome Research Site, 525 Brooks Rd., Rome, NY, 13441-4505
[2]Quimba Software, 4 West Fourth Avenue, Suite A, San Mateo, CA 94402

The Internet has become a way of life. In the past, when one was to perform research they would go to the library, proceed to the card catalogue to locate a book or to a set of periodicals for magazines. Today we sit in front of our computer, launch our Internet browser, bring up a search engine and perform various searches by entering a simple keyword, phrase or more complex Boolean expression. The Internet can be a great asset by significantly cutting the time consuming burden of finding relevant documents/papers; however, how do we know where and what we are searching? How many times do we perform a query and get irrelevant documents? In this paper we investigate today's search engines, what is meant by coverage, and what meta-search engines bring to the table. We also look at both their abilities and deficiencies and present a capability that attempts to put more power at their finger tips of the user. This capability is what we call "Buddy".

1. INTRODUCTION

The Internet has come a long way since its creation. No one ever imagined it would have grown so quickly and become so assimilated into everyone's daily lives. One of the original key objectives for its inception was to interconnect various research communities together, however not until the Web (to include browser and search engine technologies) came about that this objective was somewhat realized. Today, without these technologies the Internet would only be a collection of documents inaccessible to the general research public. Search engines like Google, Yahoo, AskJeeves, etc., have become basic tools for a plethora of scientists, researchers, students and general enthusiasts. If you went and asked many of these individuals questions like, "Which Search Engine is the best?", or "Why do you use the one you do?", in many cases the answers are very subjective. Is there one that is better than another? The answer to this question is "It depends". It depends on the application or purpose one has in mind. For typical searches where one is entering a single keyword on a current topic, today's engines are more than adequate, but for the researcher they can lack the power or capabilities required.

Typical researchers, whether it be a graduate student working on their thesis or dissertation, a professor working on a paper or an analyst working on an assessment, have developed an initial flow or outline that defines their scope. This scope can be easily described in terms of an outline. It would be nice if one could take this outline and use it to search the Internet with various search engines (will provide a discussion later why) and return the results binned in each of the topics identified in the researcher's outline. In essence the researcher is using their outline to build a "virtual" document. This "virtual" document or book can be updated automatically (by rerunning the request) and identifying new items from run to run. This results in a virtual or dynamic book that is being updated as new research is posted. Many of the search engines today are incapable of supporting this concept. So where do we begin? In section 2 we discuss the need for the use of multiple search engines and limitations in such techniques that use more than one engine. In section 3 we introduce a tool, "Buddy", which attempts to overcome a number of the discussed limitations and provide some concluding remarks and future directions in section 4.

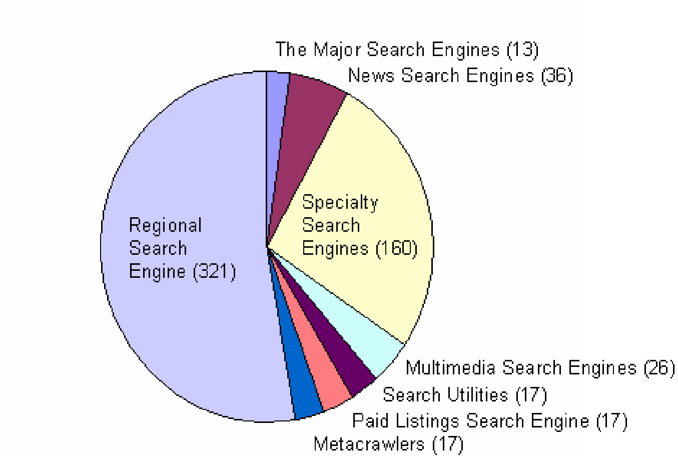
2. COLLECTION DEFICIENCIES

When using one of the many popular search engines on the Internet to search for a specific topic, many of the results returned are unrelated to what the user intended, or they do not contain the information desired. Typically the user ultimately looks through many Web pages before the information is found - if in fact, it's found at all. Different search engines search different parts of the Internet (alternately, the "Web" or "www") and thus give the user different results. The question often becomes: why is this, and which search engine to use?

There are three basic components of to a search engine: (1) the database of documents, (2) the indexing methodology and (3) the query/retrieval technique. All search engines use what are called web crawlers or spiders to traverse the Internet to find documents. They all begin with a starting set of links and transverse each link recursively until they have exhaustively visited each one. By doing so, a database of links and documents can be created. The next step is to index these documents so the database can be queried and quickly return results. Today, many search engines use some form of Boolean expressions (i.e, a list of one or more keywords connected either by "and" or "or" or a number of words within quotes to identify a specific phrase). A simple way of creating such an index is by tokenizing each document (building a list of words and in some cases saving only the stems or roots of the words), and removing common words (generally referred to as stop words, e.g., "a", "the", etc.). Once this list is generated for each document, an inverted list is created. This inverted list is a list of all keywords that were found in all the documents. As each word is added to this list, the documents in which the keywords appear are associated with the list. In this manner when one performs a simple keyword search the query mechanism returns all the documents associated with that keyword. If an "or" is used as part of the expression, the lists are appended. If an "and" is used, the intersection of the two lists would be returned. There have been many enhancements made over the years based on this approach to increase overall performance, but their basic mechanisms remain the same. So why are search engines different and return different results? Some have to do with storage and retrieval techniques, but much of it has to do with what portion of the Internet they cover. Recall earlier we discussed how search engines acquire their documents. Not everything in the Internet is interconnected. Therefore if different search engines start with a different set of sources and links, they will return a different set of documents and thus different returns when queried. So which one covers the set of data that you need? We performed a series of studies to look at the issue of coverage. In the first study we looked at the various types of search engines that are available on the Internet today; we next looked at the coverage problem by analyzing the results returned by a number of search engines. We then conclude this section with a look at meta-search engines and their coverage.

2.1 Searching the Internet

How do we access the Internet? What capabilities exist and what is available. To answer these questions we turn to Search Engine Watch. Search Engine Watch (dated 12/1/2005 and located at http://searchenginewatch.com) categorized various search strategies into 8 bins: (1) The major search engines, (2) News Search Engines, (3) Specialty Search Engines, (4) Multimedia Search Engines, (5) Search Utilities, (6) Paid Listings Search Engines, (7) Metacrawlers and (8) Regional Search Engines. We have taken the data presented on their web page and plotted it in the form of a pie chart (shown in Figure 1).



Category	Count
Major	13
Metacrawler	17
Multimedia	26
News	36
Specialty	160
Regional	321

Figure 1. Summary of search tools/techniques in the Internet

While most people simply rely on major search engines like Google or Yahoo, there are a staggering 321 regional search engines and 160 specialty search engines waiting to be exploited. Specialty search engines include those devoted to specialty subjects such as medicine, or law and could be a tremendous research aid. In addition, regional search engines could be of particular interest to travelers and vacation planners. Initially, one might argue that if the sources described above all return the same documents then it would be useless to query them. However, in section 2.2 we will see that these sources do actually produce significantly different results. Depending on the type of research it might be necessary to access sources from or about foreign activities. A second survey was conducted to identify specific foreign sources.

The results in Table I clearly show the pervasive presence of online news sources in countries around the world. We note here that the results presented in Table I were obtained from a very brief search and are only meant to be demonstrative. The abundance of information online becomes even more staggering when one examines the distribution of search engines available on the internet.

Table I. Online news sources per country

Country	# of Online News Sources
Afghanistan	3
Chechnya	3
China	31
Haiti	8
India	53
Indonesia	31
Iran	34
Iraq	36
North Korea	2
Somalia	10
South Korea	14
Sudan	6

2.2 Addressing the Coverage Issue

To address the coverage problem we conducted a series of queries using a number of major and regional search engines. The queries chosen were the top 7 queries at the time of the study. Each query was performed utilizing each search engine. All pages of the results were returned and each URL was stored in a database. A separate utility was used to resolve each alias and replaced each aliased URL with its absolute identifier. Also identified were broken links and server errors. Table II provides a list of query terms.

Table II. Queries

Paris Hilton
Christmas
KaZaA
Pamela Anderson
Britney Spears
Iraq
Java

After running each query and building the database, each results set for each query was examined for overlap. Table III provides the percent of overlap between two given search engine returns. For example if we look at “All The Web” and “AltaVista” we see a 27.12% and 27.34% overlap. What this means is that on average approximately ¼ of the results were the same when one performs the same search on “All The Web” and “AltaVista”. Table III, shown below, captures the overlap between major search engines.

Table III. Major search engine overlap

SOURCE	About	All The Web	Altavista	Google	Hot Bot	Lycos	Teoma	WiseNut	Yahoo
About	-	0%	0%	0%	0%	0%	0%	0%	0%
All The Web	0%	-	27.12%	28.22%	22.43%	89.93%	8.47%	13.86%	23.73%
Altavista	0%	27.34%	-	23.02%	21.41%	26.73%	7.24%	14.67%	18.89%
Google	0%	34.55%	27.96%	-	37.85%	34.68%	9.04%	18.32%	77.41%
Hot Bot	0%	22.57%	21.36%	31.09%	-	22.27%	7.72%	13.74%	26.68%
Lycos	0%	92.04%	27.14%	28.98%	22.65%	-	8.67%	13.47%	24.59%
Teoma	0%	46.70%	39.56%	40.66%	42.31%	46.70%	-	21.43%	32.97%
WiseNut	0%	47.28%	49.66%	51.02%	46.60%	44.90%	13.27%	-	42.18%
Yahoo	0%	33.81%	26.70%	90.06%	37.78%	34.23%	8.52%	17.61%	-

The mean overlap in the above table is 25%. However, the same analysis performed on foreign search engines produces shockingly different results. Table IV shows the same calculations for 9 foreign search engines.

Table IV. Foreign search engine overlap

SOURCE	Altavista India	Anzwers	Aport	SearchNZ	Yahoo France	Yahoo Mexico	Yahoo Sweden	Yahoo UK	search.ch
Altavista India	-	0%	0%	0%	0%	0%	0%	0%	0%
Anzwers	0%	-	0%	0%	0%	0%	0%	0%	0%
Aport	0%	0%	-	0%	0.39%	0%	0%	0%	0%
SearchNZ	0%	0%	0%	-	0%	0%	0%	0%	0%
Yahoo France	0%	0%	0.37%	0%	-	0%	0%	0%	1.31%
Yahoo Mexico	0%	0%	0%	0%	0%	-	0%	0%	0.21%
Yahoo Sweden	0%	0%	0%	0%	0%	0%	-	0%	0%
Yahoo UK	0%	0%	0%	0%	0%	0%	0%	-	0%
search.ch	0%	0%	0%	0%	0.70%	0.10%	0%	0%	-

The mean overlap in Table IV is 0% for the queries performed. These results clearly show that coverage can be increased by incorporating different types of search engines. Clearly, combining individual search engines drastically increases the coverage of a search. However, the benefit of reaching out to different types of sources remains questionable. To investigate this question the unique contributions (or distinct hits) of major search engines were compared to that of foreign search engines as shown below in Table V.

Table V. Unique contributions

Source Name	Number	Percent Total
About	485	4.7
All The Web	66	0.6
Altavista	575	5.6
Altavista India	169	1.6
Anzwers	350	3.4
Aport	503	4.9
Google	90	0.9
Hot Bot	561	5.5
Lycos	47	0.5
SearchNZ	2876	28.0
Teoma	56	0.5
WiseNut	73	0.7
Yahoo	53	0.5
Yahoo France	522	5.1
Yahoo Mexico	458	4.5
Yahoo Sweden	439	4.3
Yahoo UK	442	4.3
search.ch	989	9.6

The mean unique contribution for the major search engines is 2.2% while the mean for the foreign search engines reaches 7.3%. Analyzing the average overlap between pairs of sources suggests that regional search engines cover significantly distinct portions of the internet while in general the major search engines cover very similar regions.

It was hypothesized that combining individual search engines would help maximize the number of distinct hits for a query. The total number of distinct hits, as defined above, was computed for each individual engine as well as the collection of all engines. With these numbers, it is possible to determine by what factor the total number of distinct hits is increased by using multiple search engines. The total number of distinct hits was divided by the number of distinct hits for the engine with the largest number of distinct hits to calculate the minimum expansion factor. For the maximum expansion factor, the total number of distinct hits was divided by the number of distinct hits for the engine with the least number of distinct hits. The minimum and maximum expansion factors for each of the 7 queries are shown below in Table VI.

Table VI. Expansion factors for each query

Query	Minimum Expansion Factor	Maximum Expansion Factor
Paris Hilton	4.06	29.61
Christmas	3.45	17.41
KaZaA	4.41	16.16
Pamela Anderson	4.00	40.79
Britney Spears	4.23	23.13
Iraq	3.49	19.02
Java	3.65	19.24
Average	3.90	23.62

2.3 Meta-Search Engines

The problem with each of the methods of searching the Internet is that there are no perfect engines when it comes to finding the information a user may need. Additionally, there is no absolute basis for a comparison of the engines as each has its own unique features and databases. A site on the Internet that is clearly the best may not be found by querying only one individual search engine, forcing the user to use several search engines to perform an accurate search. In principal, a meta-search engine is a good alternative to individual search engines, but each of the meta-search engines uses a different algorithm or method of sorting the results. None of these algorithms stand out as being superior to the others. Meta-search engines are also as commercial as the individual search engines, selling a high return on its list of sites to the highest paying customer. This results in a poor representation of what is available on the Internet for the topic that they search for.

What search engines do meta-search engines rely on? We did a quick survey looking at 18 of the more popular free meta-search engines to look at the sources they access. Table VII provides a summary of the number of sources that are accessible from each engine. It seems that the typical number is around 12. There are three that are much lower (iSleauth, one2seek and Widow) and three much greater (Search,com, QuerySearch and Vivisimo).

Table VII. Summary of number of sources accessible per MetaSearch Engine

Name	Number of Sources
Debriefing	11
Dogpile	14
EXcite	12
iSleauth	5
IxQuick	12
mamma	14
metacrawler	11
metor	6
MyNetCrawler	12
one2seek	4
ProFusion	12
qbSearch	17
QueryServer	51
Search.com	28
SurfWax	11
Vivisimo	35
WebCrawler	12
Widow	14

Before we take a closer look at the sources themselves, we need to point out that even though QueryServer has access to 51 sources, the user only has access to a subset of them at any given time – based on the area of interest. That is, under the title of the "Web", one would have access to 11 of the 51 search engines. Likewise under the title "News", 10 sources are available; "Health" has 12; "Money" has 6; and "Government has 13. At no one time can you perform a single request against all 51 sources, however in Vivisimo and Search.com one can access all of their sources at once.

If we take a closer look at the sources themselves, we see that of the 18 meta-search engines, collectively they access around 115 different sources. If we looked at who accesses the source and how many of them rely on it, we see that if we take away Vivisimo with its 23 unique sources; Search.com with its 11 unique sources and QueryServer with its 49 unique sources we are left with 33 sources that are used by not only these three but the remaining 15. The next question is "What is the distribution of the 33 sources?" Table VIII provides a listing of the number of times that a given source is used and the percent of meta-search engines that use the source.

Based on the information provided in Table VIII, we can see that meta-search engines do in fact rely on the major search engines and a few of the news search engines. The only ones that go beyond these are Vivisimo, Search.com and QueryServer. Vivisimo provides access to two regional search engines: YahooUK and BBC; Search.com provides access to specialty engines such as mySimon and CNET Shopper. QueryServer supports a multitude of various specialty engines as: health, money and government. But for the most part both specialty and regional search engines have been overlooked.

Table VIII. Summary of use of individual search engines

No. of Times	Percent of Engines	Name
12	67	LookSmart
11	61	Open Directory, FindWhat, MSN
10	56	About
9	50	AltaVista, Ask Jeeves, Overture, Yahoo
8	44	Google
7	39	Fast, Sprinks
6	33	Teoma
5	28	DirectHit (HotBot), Lycos, WiseNut, Kanoodle, Excite
4	22	Ah-ha, SearchHippo, Netscape, CNN, AOL
3	17	Inktomi, EntireWeb, YahooNews
2	11	ePilot, Northern Lights, WebCrawler, GigaBlast, NYTimes, ODP, AllTheWeb

So why do the existing meta-search engines limit the number of sources that they access? Regardless of how meta-search engines get the information from the source, today's meta-search engines utilize a concept called Web Scraping. Web Scraping involves the process of querying a source, retrieving the results and parsing the page to obtain the results. At that point, the meta-search engine will then normalize the information, and in many cases, combine them with other results and present a single ranked list. The problem with this approach is that individual sources often change the format of their pages. Web scrapers break when this happens; therefore maintenance is critical. One approach taken by a majority of meta-search engines is to provide centralized service. That is, the meta-search engine is hosted on a centralized server and accessed through a web browser. A few have opted to provide a client application. In both cases, the user is at the mercy of the developer. If a web scraper breaks, the user must wait for the developer to fix it. Centralized approaches are easier to fix and require no software changes on the part of the user. On the other hand, client applications would require a patch to be downloaded. In both cases, the user has no control and cannot add additional sources at will.

Table IX. Number of sources per Metasearch Engine

Metasearch Engine	Number of Sources	Unique Sources
Dogpile	9	0
Excite	13	4
ixQuick	13	4
Mamma	12	3
MetaCrawler	8	0
ProFusion	12	3
Vivisimo	25	19
WebCrawler	9	0
Total	101	33

A closer examination of these metasearch engines reveals a number of key trends (refer to Table IX.) Merely adding up the number of sources accessed by each metasearch engine suggests that together they access 101 search engines, but this number includes duplicates. Counting each source only once reveals that in fact the metasearch engines only access 33 different sources. These numbers become even more telling when Vivisimo is removed from the study. After removing Vivisimo and recalculating the number of unique sources, there are only 15 search engines that are accessed by only one metasearch engine, and only 20 distinct search engines are used overall. In addition to not capitalizing on a large number of search engines, most metasearch engines also fail to utilize the two largest groups of search engines by ignoring specialty engines and regional engines. Table X lists the number of specialty and regional search engines used by each metasearch engine.

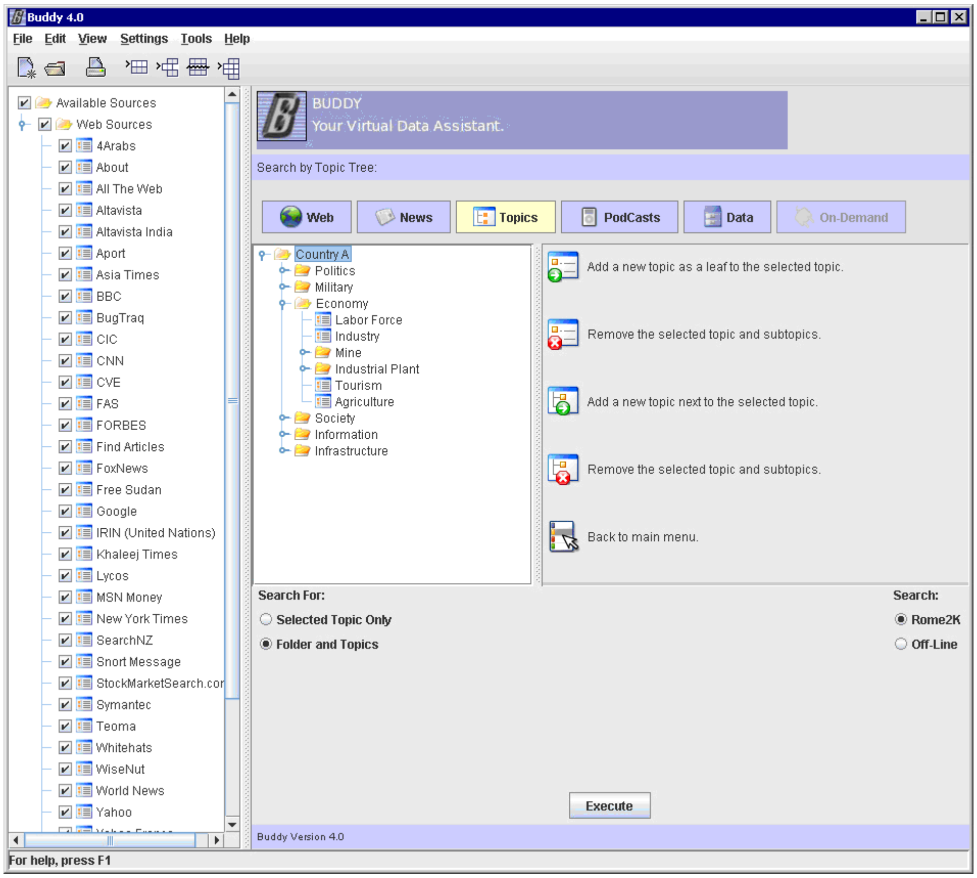
Table X. Specialty and regional search engine distribution

Metasearch Engine	Specialty Engines	Regional Engines
Dogpile	0	0
Excite	0	0
ixQuick	36	0
Mamma	1	0
MetaCrawler	0	0
ProFusion	0	0
Vivisimo	4	1
WebCrawler	0	0

Together, these tables demonstrate two shortcomings of major metasearch engines. First, in general, metasearch engines do not access many sources, especially when compared to the hundreds of search engines available. Even with Vivisimo, these metasearch engines only access 14 sources on average. Second, a look at the specific search engines reveals that metasearch engines concentrate on general purpose search engines. These limitations stem directly from the techniques metasearch engines use to search multiple sources. One technique relies on corporate agreements, while the other relies on developing custom parsers, or using publicly available Application Programming Interfaces (APIs) where available. Each of these suffers from its own limitations. Corporate agreements may be both cost prohibitive and too manually intensive to work on a very large scale and prohibit dynamically adding new sources on the fly. Developing custom parsers for each source alleviates the cost issue, but without automation it too fails to scale and prohibits any dynamic customization. Besides the problem of dealing with the dynamic world of web pages, many of the meta-search engines investigated were limited in a number of areas: a user cannot (1) perform multiple simultaneous searches using a tree or some form of hierarchical representation, (2) define their problem in terms of their own view (terms) and have the data returned back in that view, (3) add new sources at their will and (4) retrieve any/all the documents onto their computer. For these reasons, Buddy was developed.

3. BUDDY

Buddy, shown in Figure 2, is an advanced metasearch engine developed by the Air Force Research Laboratory to address the deficiencies outlined above. Buddy allows the user to achieve greater coverage of the web in general and also allows the user to focus on particular areas if he/she desires to do so. Like other metasearch engines, Buddy relies on the services of other search engines to satisfy the user's information needs. Each search engine is accessed through a "search adapter". Search adapters allow Buddy to speak to each search engine in its own language and parse the results according to each engine's individual format.



Add a new topic as a leaf to the selected topic.

Remove the selected topic and subtopics.

Add a new topic next to the selected topic.

Remove the selected topic and subtopics.

Back to main menu.

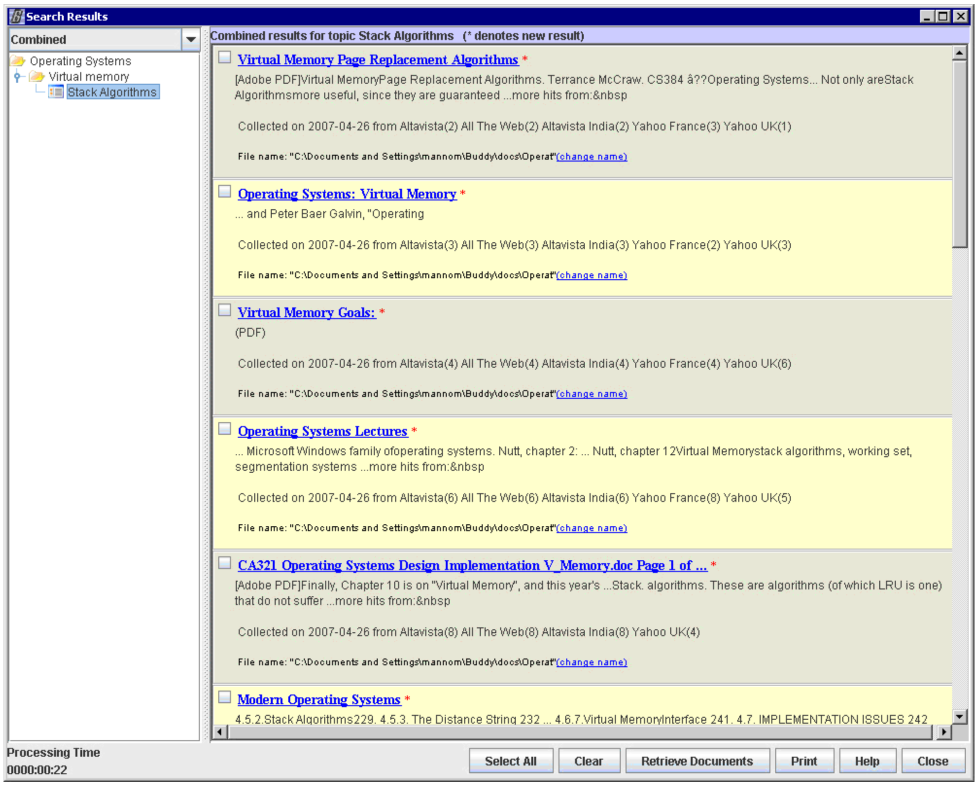
Search For:
○ Selected Topic Only
● Folder and Topics

Search:
● Rome2K
○ Off-Line

Figure 2. Buddy Main Window

Since every query is generally unique and will differ from the previous search before, Buddy groups information to help the user search as effectively as possible. Buddy does this by both grouping the search engines, and also the way the data is being searched. For example, using the Buddy system, a user could build a topic tree, search news archives, or perform a simple keyword search to query multiple sources on the web (that specialize in Law, Medicine, Financial, Foreign sources, etc.), to help return the most relevant results based on that particular search. Alternatively, the user can decide not to allow for engine grouping, and search against all of the engines available to return the largest number of hits possible.

Once Buddy obtains the results, parses the page and stores the information, a summary page is presented. The summary page provides the user a quick synopsis of what is available from the various sources for each of the topics chosen. From this page the user can get results from an individual search engine, display a combined list ranked by Buddy’s own ranking algorithm, retrieve the individual documents or print out the results. The results of each search are compared to the results of past requests of the same topics that are stored in a database and any new items are identified on the summary page. Figure 3 shows the results window after performing a single topic search on “Operating Systems” AND “Virtual Memory” AND “Stack Algorithms”.



Combined results for topic Stack Algorithms (* denotes new result)

□ Virtual Memory Page Replacement Algorithms *
[Adobe PDF]Virtual MemoryPage Replacement Algorithms. Terrance McCraw. CS384 â??Operating Systems... Not only areStack Algorithmsmore useful, since they are guaranteed ...more hits from:

Collected on 2007-04-26 from Altavista(2) All The Web(2) Altavista India(2) Yahoo France(3) Yahoo UK(1)

File name: "C:\Documents and Settings\mannom\Buddy\docs\Operat"(change name)

□ Operating Systems: Virtual Memory *
... and Peter Baer Galvin, "Operating

Collected on 2007-04-26 from Altavista(3) All The Web(3) Altavista India(3) Yahoo France(2) Yahoo UK(3)

File name: "C:\Documents and Settings\mannom\Buddy\docs\Operat"(change name)

□ Virtual Memory Goals: *
(PDF)

Collected on 2007-04-26 from Altavista(4) All The Web(4) Altavista India(4) Yahoo France(4) Yahoo UK(6)

File name: "C:\Documents and Settings\mannom\Buddy\docs\Operat"(change name)

□ Operating Systems Lectures *
... Microsoft Windows family ofoperating systems. Nutt, chapter 2: ... Nutt, chapter 12Virtual Memorystack algorithms, working set, segmentation systems ...more hits from:

Collected on 2007-04-26 from Altavista(6) All The Web(6) Altavista India(6) Yahoo France(8) Yahoo UK(5)

File name: "C:\Documents and Settings\mannom\Buddy\docs\Operat"(change name)

□ CA321 Operating Systems Design Implementation V_Memory.doc Page 1 of ... *
[Adobe PDF]Finally, Chapter 10 is on "Virtual Memory", and this year's ...Stack. algorithms. These are algorithms (of which LRU is one) that do not suffer ...more hits from:

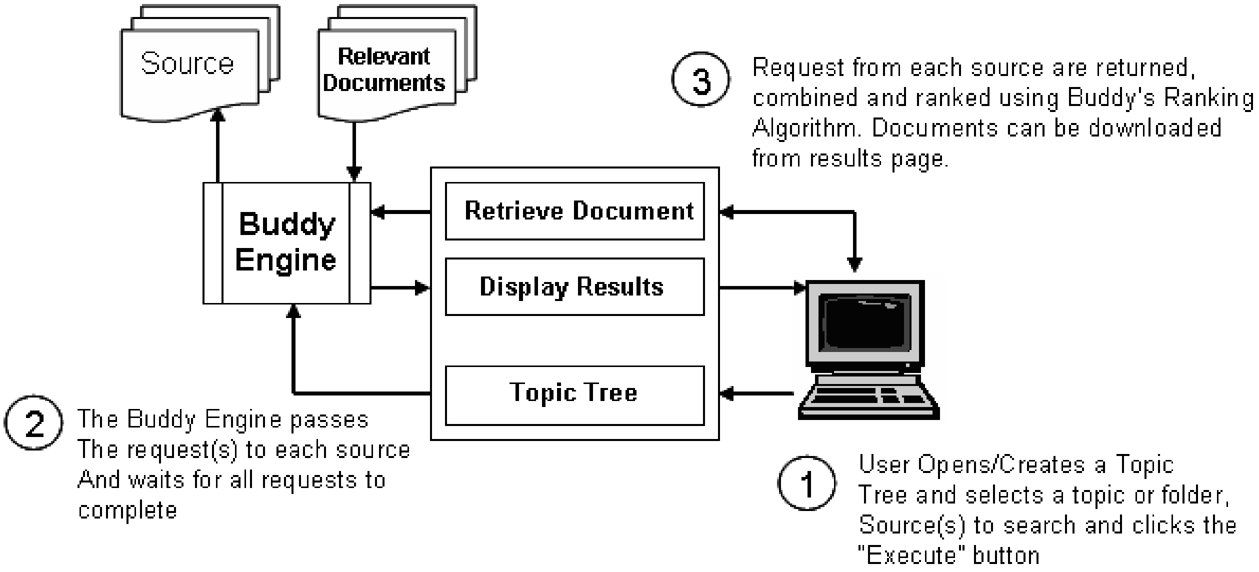
Collected on 2007-04-26 from Altavista(8) All The Web(8) Altavista India(8) Yahoo UK(4)

File name: "C:\Documents and Settings\mannom\Buddy\docs\Operat"(change name)

□ Modern Operating Systems *
4.5.2.Stack Algorithms229. 4.5.3. The Distance String 232 ... 4.6.7.Virtual MemoryInterface 241. 4.7. IMPLEMENTATION ISSUES 242

Figure 3. Results Page

The database of past results also allows Buddy to operate in the absence of the internet. Upon startup, Buddy checks to see it has Internet access. If no network connection was found, Buddy will ask if the user wishes to continue in an off-line mode. If the user chooses to continue in off-line mode, the Buddy main screen will appear and all subsequent requests will be performed against the database of previous search results. Figure 4 provides a summary of the basic flow of using Buddy.



Request from each source are returned, combined and ranked using Buddy's Ranking Algorithm. Documents can be downloaded from results page.

The Buddy Engine passes
The request(s) to each source
And waits for all requests to
complete

User Opens/Creates a Topic Tree and selects a topic or folder, Source(s) to search and clicks the "Execute" button

Figure 4: Overview of the Buddy Process

3.1 Topic Trees

Buddy stands out from traditional metasearch engines by utilizing topic trees, providing dynamic source access, easy maintenance and a number of additional features to support research. Through Buddy, a user defines his/her information requirements in terms of an outline or topic tree. Unlike any other metasearch engine, Buddy uses the concept of topic trees to assist the user in defining their problem. Topic trees allow the user to define their problem as a set of hierarchical terms starting with a broad term and continuing to link terms together to define the problem set using more detailed terms further down in the tree. Figure 5 shows an example of a topic tree developed to study a particular country using the PMESII (Political, Military, Economy, Society, Information and Infrastructure) model. In this example, the main topic is “Country A”. Six sub-topics have been defined below the main topic (e.g., “politics”, “military”, “economy”, etc.) The folder icon in front of each sub-topic indicates that each one has additional subtopics. For example, the “economy” sub-topic has been further defined with such topics: “labor force”, “industry”, etc.

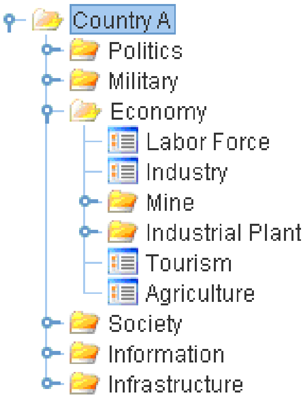


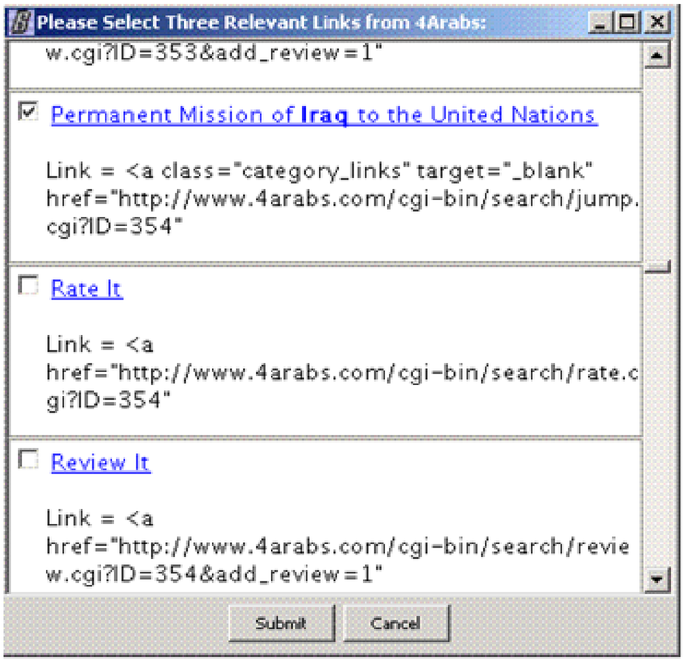
Figure 5. Example of a Topic Tree

Topic trees are extremely useful for a number of reasons. The tree structure serves as a convenient storage structure to organize query results and allows the user to establish context with multiple query terms. For example, consider the classical example of the word “bank”. Without any context, the word bank may refer to a financial institution, a geographic feature of a river, or the verb describing an action made with an airplane. If used alone, the results may contain documents covering financial, geographic or aeronautical banks. To address this issue, many search engine vendors are attempting to provide semantic capabilities within the engines themselves or to provide a natural language interpreter to discern the meaning. Both have had limited success. Another approach is to require the user to define their query in the correct search engine specific syntax. Topic trees allow for this capability to be automated for the user. Child nodes in the tree are joined with their parent to form an “and” query. In the tree example above, one potential query could be “Country A” and “politics.” This query would only return results that mention both “Country A” and “politics.”

Binning is an additional advantage provided by topic trees. A number of metasearch vendors provide a clustering capability that organizes the results of a query as a list of hierarchical terms/concepts derived from the content of the returned documents. In these cases, the user cannot define their own language nor do they have any control over the cluster topics. Topic trees, on the other hand, allow the user to define the concepts of interest and view the documents accordingly. In addition, to organizing the results for the user, topic trees also allow the user to perform multiple queries at once. If the user selects to query the folder rather than the topic only, every topic beneath the selected folder will also be sent to each search engine. Because the topic trees allow Buddy to save the results of each query, they can also be used as standing queries that may be performed daily, or weekly. One tree can also be used as a template for a second. In the country example used above, the user need only change the root node from Country A to Country B to begin an entirely new collection. This feature benefits researchers as well as everyday uses such as vacation planning. Finally, topic trees also allow Buddy to optimize the collection process. Buddy inspects the results of each query to determine if a specific source returns back zero hits for the specified topic. If a source failed to find any results for a particular topic then Buddy removes all subtopic requests from the source's request queue.

3.3 Dynamic Access

Buddy has the ability to take a tree as described above, convert it to a series of requests and pass the requests to the selected search engines. As previously mentioned, search engines are accessed through "search adapters". Search adapters allow Buddy to communicate the information request to each individual search engine using that particular search engine's language. However, commercial search engines often change their web site's format which will break most metasearch engines until the new format is discovered. Some search engines may change as frequently as twice a month while others may only change once a year or even less frequently. Furthermore, these changes are often in HTML, or some scripting language, unrelated to the visual appearance of the page so the page appears the same in spite of the underlying changes. Buddy provides a unique capability within its interface to dynamically add and modify existing sources without any programming. In order to query a search engine Buddy only needs to know how to make a request to the search engine, and how to read the results the search engine returns. Initially, this knowledge was manually entered into the adapter files through Buddy's user interface. While this approach had the benefit of allowing users to add or change any source without performing any programming it failed as an acceptable solution because manually figuring out the information to enter was a rather involved task. To alleviate these concerns Buddy was given the intelligence to learn how to interpret the results from a search engine. If an adapter breaks, Buddy simply uses previous results to relearn how to read the results from the search engine. This approach alleviated some of the maintenance concerns and also greatly facilitates the addition of new sources. If an analyst wishes to add a source, they need only specify the URL Buddy will use to query the source. Once entered, Buddy will connect to the source and ask the analyst to select three relevant results, as shown below in Figure 6. After the user selects three links, Buddy will learn how to parse the page by looking for similar structure around each of the user supplied examples. Once the process is complete, Buddy presents the new parsing information for the user's approval.



Please Select Three Relevant Links from 4Arabs:

w.cgi?ID=353&add_review=1"

☑ Permanent Mission of **Iraq** to the United Nations

Link = <a class="category_links" target="_blank" href="http://www.4arabs.com/cgi-bin/search/jump.cgi?ID=354"

☐ Rate It

Link = <a href="http://www.4arabs.com/cgi-bin/search/rate.cgi?ID=354"

☐ Review It

Link = <a href="http://www.4arabs.com/cgi-bin/search/review.cgi?ID=354&add_review=1"

Figure 6. Parser Training

3.4 Automatic Retrieval

Buddy introduces a number of additional features to aid the analysis process as well as the casual user and professional researcher. The results screen, for example, utilizes a voting algorithm to produce a ranked list of the sites that are most likely relevant. In addition, the user may still navigate through the results as they were ranked from each source. Any sites found to be particularly useful may be automatically downloaded to the user's machine for storage or additional processing. Because websites often contain many pages that are linked together and many sites may only have a table of contents on the first page, Buddy allows the user to specify that it should follow the links on each page until it is at some specified depth. For example, if the user instructs Buddy to retrieve to a depth of two from CNN’s website then Buddy will go to the first page and then to every link on the first page that remains within CNN’s website. For each of those links, Buddy will follow each link that remains within CNN’s site and store each page. In addition, the user may also specify whether sites should be stored as html pages or as plain text files in which all html tags have been removed.

3.5. The Customization Workbench

The Customization Workbench allows the user to customize Buddy’s search process for specific information needs. It was added to Buddy in order to help researchers and analysts navigate through the potentially large volumes of data that Buddy could return. Figure 7 shows schematically how the search process can be customized.

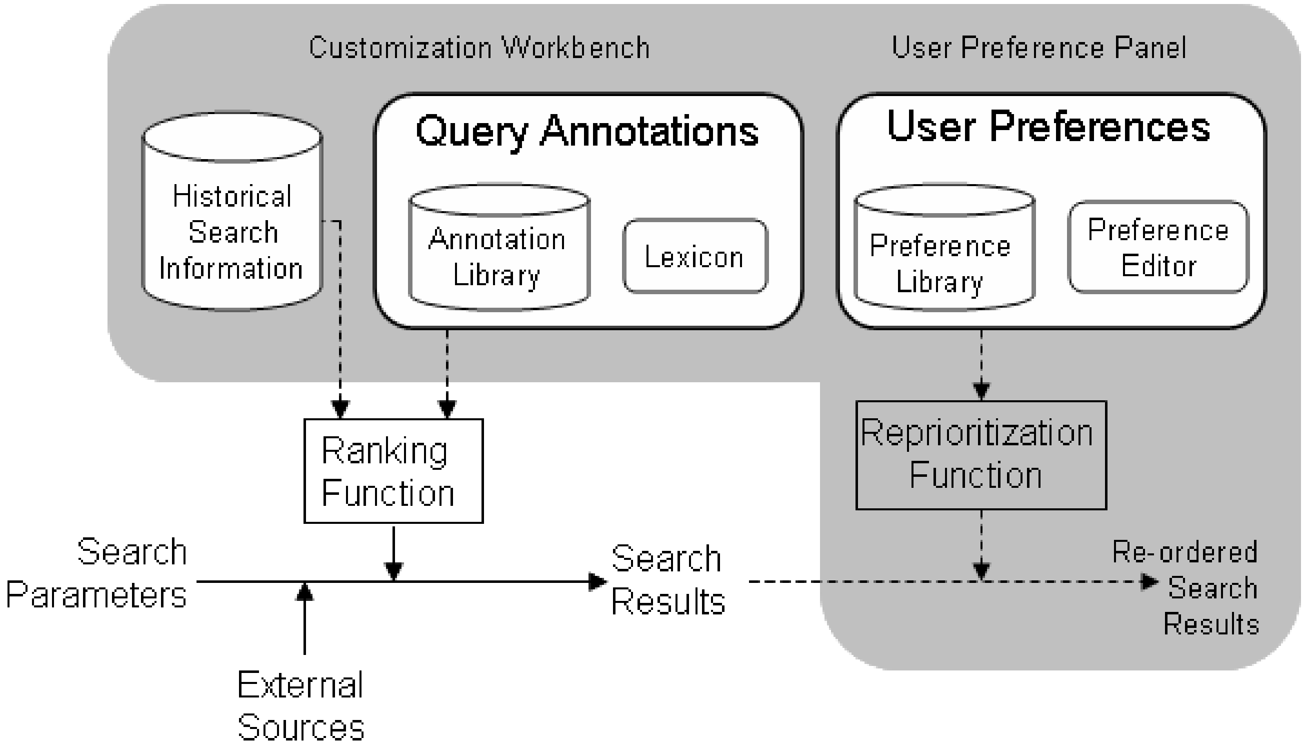


Figure 7. Customized search process. The unshaded portion represents the default search process

The unshaded portion of the figure represents Buddy's current default search process: the user inputs the query and selects the sources to be searched; the sources are queried; and a generic ranking function is used to order the results of the metasearch for presentation to the user. We use the term *generic* to emphasize that fact that the ranking algorithm makes no reference to the actual query that produced the results. We also note here that the ranking algorithm is used to evaluate only the passages of text that are returned by the search engine for each hit – i.e. the title of the hit, and any short summary from the search engine.

The shaded portion highlights the customizations that the user can add to the search process. Before the search is executed, the user can optionally choose to provide the ranking function with *historical information* about the sources being queried; he or she can also optionally choose to *annotate* the query terms with information about the exact intended meaning of the query terms. This annotation information can be built from a lexicon, or retrieved from a library of previously created annotations. The resulting ranking function is hence tuned to the specific search being executed, producing a better ordering of the results. After the search is executed, the user can choose to specify additional preferences that he or she has concerning other attributes of a desired document, such as the author of the document, or the domains from which a preferred document originates. These *user preferences* are used to reprioritize the search results, improving the presentation of the results further.

This modular design gives the user the choice of using any combination of the customization options described above or none at all. Hence he or she can control the trade-off between customization effort and improvement of the search process. For the rest of this section, we will describe each of the options discussed above. Section 3.5.1 will discuss the creation and use of a customized ranking function. Section 3.5.2 will discuss reprioritization of the results, using user preferences. We will also describe the different kinds of annotations that can be created, in Section 3.5.3.

3.5.1. Customizing the Ranking Function

By default, Buddy presents the results of a search to the user using only positional information from the queried search engines: the rank of a hit in the aggregated list is a function of how many search engines returned that hit, and how it was ranked by each of those sources. This ranking function is generic, in the sense that it does not take any information about the query itself into account. However, many potential query terms such as *virus*, *intelligence*, or *bank* have multiple common meanings. It is important, especially in a metasearch, where hundreds or thousands of documents can be returned, to sort out those documents which are pertinent to the meaning of the search term intended by the user. The Customization Workbench allows the user to add needed query information to the ranking function, on a per-search basis.

We refer to the query information as an *annotation*. The annotation is used to evaluate the title of each returned hit, *h*, as well as any text snippets about the hit that are returned by the search engine(s). The result of the evaluation is two weights, designated R (*h*) and NR(*h*). R(*h*) is a weight corresponding to the strength of the hypothesis that *h* is relevant to the user's query; NR(*h*) is the weight corresponding to the strength of the hypothesis that *h* is not relevant to the query. If one merely wished to label the hit as relevant or not, then the label could be determined by comparing which of R(h) and NR(h) has the greater value. How these weights are determined will be discussed in Section 3.5.3.

In this case, rather than labeling, we wish to sort the hits in order of decreasing probable relevance. We use the ratio

$$s_h = R(h)/NR(h) \tag{1}$$

to sort each document. We have found that using a score that accounts for both positive and negative evidence of a hit's relevance produces better results than devising a scoring function based on positive evidence alone.

3.5.1.1 Use of Historical Search Information

The Customization Workbench also enables the user to automatically collect documents from a search and label them as relevant or non-relevant, using an annotation. The new information can then be used to create and update historical performance records – specifically, how many documents have been collected from a given search engine on a given topic, and how many of the collected documents are relevant. By performing this process periodically, the user can refine the estimates of which search engines tend to be most useful for different topics.

We organize the historical information along the same hierarchies as the topic trees. If the user wishes to use historical information to further tune the ranking function, and there is no historical information for the specific query being made, then the system will look for information by generalizing – that is, traversing up the topic tree.

The historical information about search engines is used in the ranking function by adding an *engine weight* to the text-based score for each search engine return. For example, suppose the hit *h* were returned from search engine *E* in response to a query. Evaluating *h* with an annotation will result in a text-based score, $s_h = R(h)/NR(h)$. If P_E is the historical proportion of relevant document from E for this topic, then the engine weight w_E is given by

$$w_E = P_E /(1 - P_E). \tag{2}$$

If the system does not have historical information about *E*, then w_E is set to unity ($P_E = 0.5$). This gives us the total score for *h*:

$$S_h = s_h \cdot w_E = [R(h) \cdot P_E] \,/\, [NR(h) \cdot (1 - P_E)]. \quad (3)$$

Heuristically, this score approximates weighting the text-based weights with the probability that a return E will be relevant. Collecting and using historical information to rank the results of a search in this manner can improve the efficiency of the search over time, as it replicates the process of looking for information from the most likely sources first, without the user having to explicitly keep track of this information manually.

3.5.2 User Preferences

In developing the text based ranking functions and scoring algorithms, we came to the conclusion that text analysis alone is not always enough to give the user the documents that are most useful to his or her goals or needs. For example, a computer scientist doing research on the latest data mining algorithms, and a business analyst looking for the latest data mining-based analytics products may use many of the same queries, but would each consider different subsets of the returns to be relevant. Even two researchers making the same queries, would probably have different preferences for the returned documents. One may especially prefer the work of a certain professor, while another may prefer a specific type of technical approach. Yet, it is problematic to express those preferences in a search query, without risking the loss of potentially valuable documents that do not exactly match the query.

We developed our user preference representations because we felt that the inclusion of “subjective” criteria, examples of which we have given above, is a complement to “objective” query-based criteria for determining document relevance. Together, they allow a user to trade off the recall and precision of their searches.

As shown in Figure 8, the user has the option of reprioritizing the search results after the search has been executed. This allows the user to develop the preferences interactively, or as needed, as he or she explores the search results. User preferences are represented as the *attribute* of the hit that is to be examined, and *values* for that attribute that are preferred or not-preferred. For example, the attribute might be the domain from which the hit originates; a preferred value might be kdnuggets.com; a not-preferred value might be businessintelligence.bitpipe.com.

We based our reprioritization algorithm on existing algorithms for rank aggregation, found in the literature on voting theory[12,13]. In general, most voting and rank aggregation algorithms assume that every voter’s opinion has equal weight in the aggregate decision. This is not true in our case: the rank based on the query-relevance analysis should dominate the others, because it is possible to have search results with favored attributes (for example, a document that originates from a preferred site) that are only peripherally relevant to the actual query. The user preferences are used only to adjust the rankings of individual documents up or down, according to these other attributes of the document. While algorithms do exist in the literature that can accommodate different weights for different voters’ opinions, they tend to either use an ad-hoc assignment of the weights, or rely on a supervised training step to adjust the weights[16,17]. Neither of these properties was desirable for this application.

3.5.2.1. Combining Preferences: Borda Count

We considered variations of two different well-known voting algorithms. The ***Borda Count***[13] assigns scores to each candidate document, according to each voter. For example, if there are three documents to be ranked, then for a given voter, the top-ranked document would get a Borda Score of 1, the next document a score of ½, and the bottom-ranked document a score of 0. The total Borda Score of each document is the sum of all of its individual scores, and the aggregate ranking is given by sorting the documents in descending order of Borda Score. In general, the Borda Score B_i assigned to a document by the *i*th voter is:

$$B_i = (N - r_i)/(N - 1) \tag{4}$$

where N is the total number of documents, and r_i is the rank of this document on voter i's list. The total Borda score for the document is then the sum of B_i over all the individual voters.

The Borda Count procedure is easy to implement, and efficient to run. It is also easy to modify so that the ranking based on the textual relevance analysis dominates the others. To see this, observe that, in general, user preferences will not specify a total ordering on the documents returned. Documents that have attribute values about which the user has expressed an opinion can be ordered by the preference rule; documents that have neutral or unknown values of the attribute cannot be, other than to specify that they come below documents with preferred values, and above documents that are not preferred values.

Since the Borda Count does not explicitly take partial orders into account, we would like to impose an order on the unordered documents. In order to accomplish this, we take the ordering from the original ranking. In terms of the implementation, this is accomplished by starting with the documents in the original relevance-ranked order, and stable-sorting the list, using the preference rule as the comparison function. A *stable sorting algorithm* is a sorting algorithm that preserves the original order of elements that are of equal value, according to the comparison function. Java's internal sort methods are guaranteed to be stable. This is shown in Figure 8.

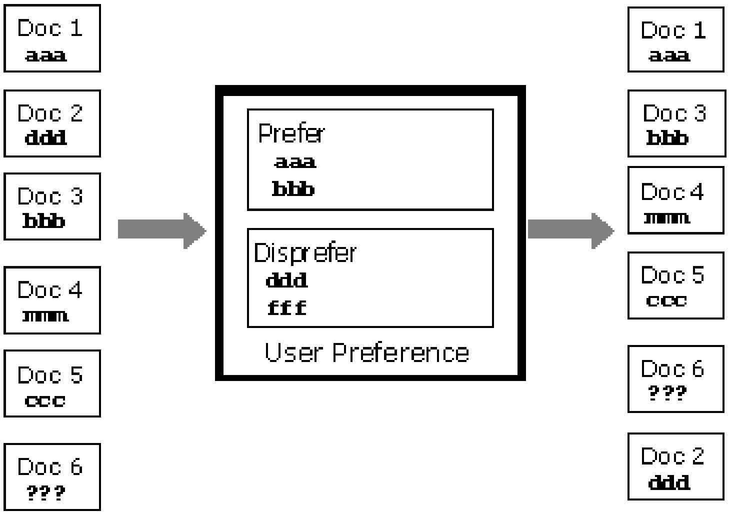


Figure 8. Stable-sorting a list, using a preference rule.

3.5.2.2 Combining Preferences: Copeland Method

We also considered the ***Copeland method***[12]. The Copeland method is essentially the method by which professional hockey team rankings are calculated. For an individual voter's ranking, each document gets 1 point for every document that it outranks, 0 points for every document that outranks it, and ½ point for every document that it ties. The document's total Copeland score is the sum of its Copeland scores over all voters. The Copeland method is also easy to implement, and does explicitly handle partial orders. This gives us the procedure shown in Table XI for combining user preferences with the relevance ranking.

Table XI. Procedure for reprioritizing the results of a search, based on a set of preferences

```
Given: a set of documents: $D$,
       a relevance ranking of $D$: $RR$,
       a set of user preferences: $\{P_i\}$,
Produce: a new ranking of $D$, $R$, that preserves the relevance information from $RR$, while accommodating the user preferences expressed in $\{P_i\}$.

For each $P_i$ in $\{P_i\}$:
    Start with the documents sorted according to $RR$, and create $p_i$, as appropriate:
     Borda:  Let $p_i$ be the ranking produced by stable-sorting $RR$, using $P_i$ as the comparison function
     Copeland:  Let $p_i$ be the partial order imposed on $RR$ by $P_i$

$R$ is then the produced by aggregating the set of rankings $\{p_i\} \cup RR$, using Borda or Copeland, as appropriate.
```

For the modified Borda method, it seems intuitively clear that the text-based relevance ranking *RR* from Table XI will in fact dominate the aggregate ordering, without any explicit weight-setting required, since it is the basis of each of the individual preference orderings, p_i. We had hoped that *RR* would also dominate the ordering using the Copeland method, since *RR* imposes the only total ordering on the documents. This did not prove to be the case, as we verified in a series of experimental trials.

3.5.2.3 Reprioritization

Both the Borda and Copeland algorithms are available to the user for reprioritization. The *Mild Resort* uses the modified Borda Method. It is recommended for cases where the search returns many nonrelevant documents, which have been pushed to the bottom by a relevance-based ranking function. The mild re-sort keeps non-relevant documents from coming back up to the top of the list. The *Strong Resort* uses the Copeland Method, and is recommended for cases when most of the documents are relevant, or when the original ranking is not relevance based - Buddy's original generic ranking, for instance. As an example of the difference between Mild and Strong Re-sort, consider the following hits, returned by the query *intelligence analysts*: There were 118 total returns (refer to Table XIII), most of which were relevant.

Table XII. First three results of query *intelligence analysts*

1. IALEIA main page
From http://www.ialeia.org
IALEIA: International Association of Law Enforcement Intelligence Analysts…by analysts, called Foundations of Intelligence Analysis Training (FIAT)….

2. Intelligence Analysts Bookshelf
From http://www.tscm.com
… Intelligence Analysts Bookshelf. Recommended Readings. Brotherhood. by Frank McCourt. In the Line of Duty. A Tribute to New York's Finest and Bravest

3. BBC – Press Office – Letter from Richard Sambrook to Alastair Campbell
From http://www.bbc.co.uk
bbc.co.uk/pressoffice is the online BBC Press Office there was debate amongst intelligence analysts whether the [45-minute source's] claims should have

Suppose that the user was interested in researching a possible career as an analyst: he or she would be interested in seeing results about the training or education required, or about the different kinds of job opportunities. Such results might include terms like *career*, *job*, *training*, *course*, or *employment*. It would be problematic to try to construct a query in Buddy that contained all of those terms. However, one could create a preference that designates the above terms as being desirable in a document. For this example, we will assume that the preference for each term is in the order given (i.e. *career* is more preferred than *job*). Using this preference, the user can find the most interesting results from the existing list. The results for both a mild and a strong re-sort are shown in Table XIII.

As expected, the re-prioritization based on the modified Borda method adheres more closely to the original ordering of the documents than the re-sorting based on the Copeland method. Which reordering is better depends on the needs of the user, and on the quality of the original collection. The interactivity of the process allows the user to explore and rearrange the list according to different, possible multiple, preferences, as desired.

Table XIII. Reprioritizations of the results from query *intelligence analysts*

Mild Re-sort (Modified Borda)
1. (Originally ranked 15) Career Paths at the National Security Agency (NSA) - Intelligence ...
From http://www.nsa.gov
As an intelligence analyst with the National Security Agency (NSA) you will provide America's leaders with the critical information they need to protect our...

2.(Originally ranked 1) IALEIA main page

3. (Originally ranked 4) Crime and Intelligence Analysts
From http://www.calmis.cahwnet.gov
This California Occupational Guide provides statewide information about job duties, working conditions, employment outlook, wages, benefits, ...

Strong Re-sort (Copeland)
1. (Originally ranked 15) Career Paths at the National Security Agency (NSA) - Intelligence ...

2. (Originally ranked 4) Crime and Intelligence Analysts

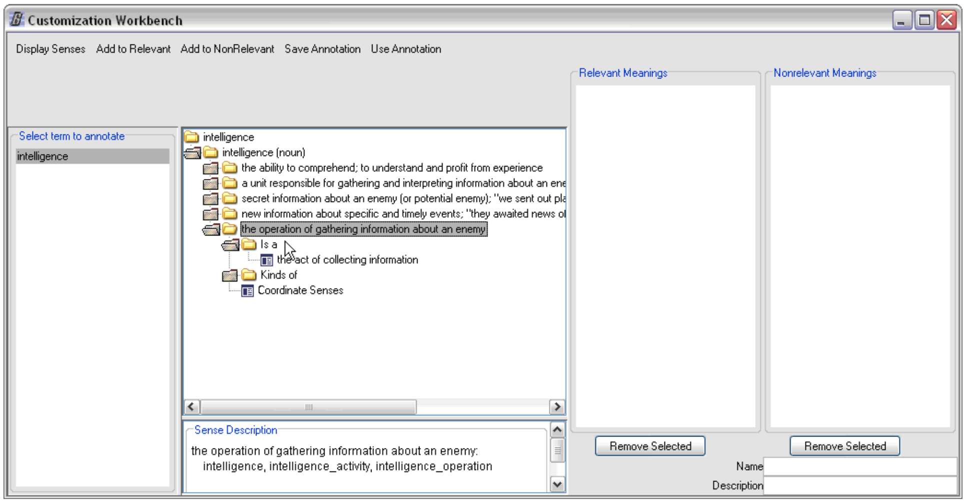
3. (Originally ranked 38) View Job #13Jan0500007
From http://64.157.137.11
...Business Intelligence Analyst...Specifically, two analysts are needed to support business reports and metrics development using the industry standard...

3.5.3 Creating Annotations

In the previous sections, we have discussed the various options that a user has for customizing the search process, but how does the user create the annotations that are used to customize the ranking function? Annotations encapsulate semantic information about the query that is used by the ranking function to favor hits that are most likely to be relevant to the user's needs. We implemented two kinds of annotations. *Query annotations* are descriptions of the meanings of ambiguous query terms. *Learned annotations* are learned from a combination of collected documents and a query annotation. This process requires somewhat more effort on the part of the user than simply the query annotations; the trade-off is improved performance on subsequent executions of related searches.

3.5.3.1 Query Annotations

Query annotations allow the user to explicitly define the exact meaning of ambiguous query terms. They are created by using information from a lexicon. We used the WordNet[14] 2.0 hierarchical lexical reference system for the information about different word meanings (or *word senses*, in the WordNet vocabulary). WordNet organizes its lexicon in terms of synonym sets, or *synsets*. Different relations link the synsets, including parent-child (*is-a*/*kind of*) relationships, category membership, usage relationships, and *part-of* relationships. Figure 9 shows the workbench for building annotations.



intelligence

intelligence (noun)
- the ability to comprehend; to understand and profit from experience
- a unit responsible for gathering and interpreting information about an ene
- secret information about an enemy (or potential enemy); "we sent out pla
- new information about specific and timely events; "they awaited news ol
- the operation of gathering information about an enemy
  - Is a
    - the act of collecting information
  - Kinds of
  - Coordinate Senses

the operation of gathering information about an enemy:
intelligence, intelligence_activity, intelligence_operation

Figure 9. Annotation Builder

The Annotation Builder displays all of the synsets corresponding to the query term, as well as the children, parents, and coordinate (sibling, or other children of the same parent) senses for each synset. The user can then select appropriate sets of relevant and nonrelevant senses to narrow down the exact intended meaning of the term. This is the lexicon-based annotation of the query. The annotation can optionally be saved to the library for re-use, later.

Once the user has selected relevant and nonrelevant senses of his/her query terms, this information can be used to evaluate a passage of text, by measuring the "distance" of the text passage from relevant and non-relevant concepts. This is done by measuring the distance of the individual terms within the text passage from the selected word senses. This distance is estimated by traversing the network of WordNet relationships, from the candidate term to each of the target senses. For this application, we measure the distance between two synsets with respect to several different relationships, and choose the shortest path to determine distance. After identifying all the terms in a text passage, *h*, whose distance from a relevant sense is finite, the text passage's relevance weight, R(*h*) is calculated as

$$\mathrm{R}(h) = \sum_{N_{terms}} \frac{1}{d_i + 1}, \qquad (5)$$

where N_{terms} is the number of terms a finite distance from a relevant sense, and d_i is the nearest such distance. The term $1/(d_i + 1)$ is equal to 1 if the term in question has a directly relevant sense, and goes to zero the more distant the term becomes. A similar calculation holds to obtain the text's non-relevance score, NR(*h*). Table XIV shows an example annotation for the query *virus*, where the query term is meant in terms of a computer virus.

Table XIV. Selected senses for query "virus" (virus-3)

Senses	**Representative Terms**
Relevant (computer virus)	
a software program capable of reproducing itself and usually capable of causing great harm to files or other programs on the same computer	virus, computer virus
a computer program designed to have undesirable or harmful effects	malevolent program
a delayed action computer virus	logic bomb
a program that appears desirable but actually contains something harmful	trojan, trojan horse
a software program capable of reproducing itself that can spread from one computer to the next over a network	worm
Non Relevant	
any organism of microscopic size	microorganism
(virology) ultramicroscopic infectious agent that replicates itself only within cells of living hosts	virus
a minute life form (especially a disease-causing bacterium)	microbe, bug, germ
any disease-producing agent	pathogen
(microbiology) single-celled or noncellular spherical or spiral or rod-shaped organisms lacking chlorophyll that reproduce by fission; …	bacteria, bacterium

Table XV shows the evaluation results for a result that was returned using this query (we omit the summary of the result from the table, for brevity). The hit receives a score of R/NR = 1/ 2.2 = 0.45; it would probably be fairly low on the list of ranked hits, as would be appropriate.

Table XV. Example search result scoring

Permissiveness of Guinea Pig Alveolar Macrophage Subpopulations to Acute Respiratory Syncytial Virus Infection In Vitro		
Relevant		
Virus	0	1
R		**1**
Non Relevant		
Pathogen	0	1
respiratory syncytial virus	4	0.2
Virus	0	1
NR		**2.2**

A significant issue with the WordNet lexicon is that it has fairly uneven coverage; for instance, in the above example, there was no visible relationship between the term *virus* (in any sense) and the term *infection*. Hence, we felt that a lexical distance-based technique (at least one based on WordNet) could still be improved by an algorithm for extracting more identifying terms, such as the semi-supervised algorithm that we will discuss in the following section.

Another option is to replace WordNet with a lexicon with greater coverage. Recently, there have been several research efforts on automating the process of finding synonyms, and other relations among words; one such recent effort is the ongoing KnowItAll[9,11] project at the University of Washington.

3.5.3.2. Learned Annotations and UDLH

In addition to lexicon-based annotations, the system can also learn annotations for a query, using documents collected from the query (but not labeled), and a simple labeling heuristic - for example, the lexicon-based query annotations. The labeling heuristic relates a subset of the data to concepts that are meaningful to the user, while structural similarity of the documents is used to label other documents that could not be labeled accurately by the heuristic. The annotations are in the form of two sets of indicators, one for relevant documents, and another for non-relevant documents, which can be used to partition documents or hits for similar queries in the future. We refer to our approach as *UDLH: **U**nlabeled **D**ata + **L**abeling **H**euristic*.

UDLH frees the user from any manual labeling of documents. However, a fundamental limitation of the approach is that the final performance depends on the precision of the labeling heuristic. If the labeling heuristic introduces too large an error into the initial document partitioning, this noise can be propagated by the rule-learning part of the process. Hence, it is best to have a very high precision heuristic, and let the rule-learning process generalize from a clean, albeit small, initial set of labels. We will discuss some ideas we have for addressing this limitation later in this section.

Figure 10 illustrates the UDLH process. Once the search of interest has been executed, we collect the full text of the search results; this provides more data for the pattern mining algorithm, as compared to using only the titles and summaries of the returned hits.

The unlabeled documents are passed into an initial relevance filter, which labels the documents as relevant, not relevant, or leaves them unlabeled (no decision). The labeled documents are passed to a pattern mining algorithm, which learns two sets of patterns: one set of patterns that strongly indicate that a document is relevant, and another set of patterns that strongly indicate that a document is not relevant. These sets of pattern are used to build a new relevance filter, which is then used to refilter the entire collection of documents. This process is iterated until the convergence criterion is met. At this point, the process returns with the documents partitioned into relevant and not relevant, and with the final set of learned relevance patterns (and by extension, the final relevance filter).

To terminate, we iterate the process either until all of the documents are labeled, or until some maximum number of iterations is performed (we use five). In the second case, we take the labeling and relevance patterns that produced the fewest number of uncertain cases as our final set. Information about the labeled documents is used to build up historical profiles of the search engines queried, for future use. The resulting relevance patterns are essentially the *learned annotations*, and they are stored in the annotation library, for use in subsequent executions of related searches. We use query annotations and the WordNet distance function to provide the initial filtering. A document *d* is relevant if R(*d*) > 1.1*NR(*d*); not relevant if NR(*d*) > 1.1*R(*d*) , and is left unlabeled R(*d*) and NR(*d*) are within 10% of each other.

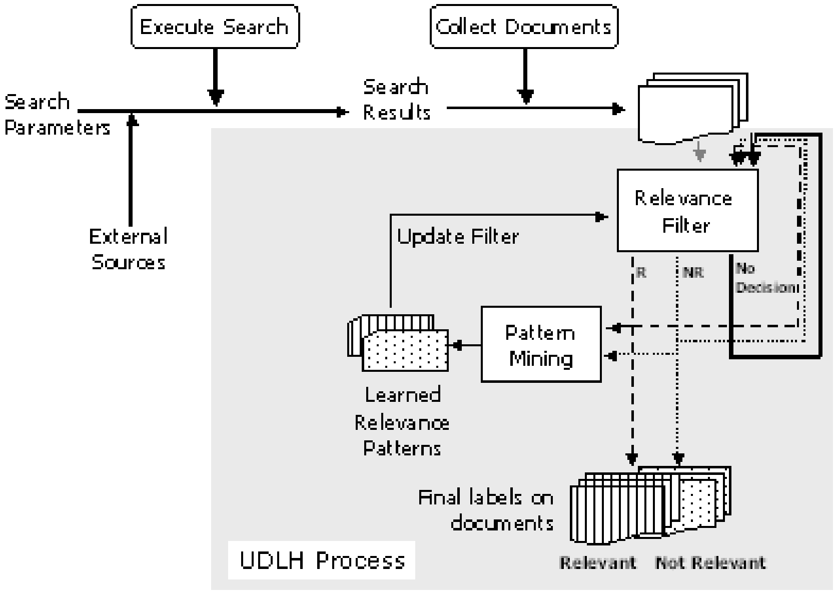


Figure 10. The UDLH Process

We mention here two approaches similar to ours. McCallum and Nigam[10] presented a semi-supervised method for classifying text from unlabeled documents by iteratively learning a series of Naïve Bayes Classifiers. The classification task was to classify Computer Science papers into a hierarchy of research topics, and the classification process was started by using a set of keywords corresponding to each topic to assign preliminary labels to the documents. Aggarwal, et al[6] use information from a pre-existing taxonomy (the *Yahoo!* taxonomy) along with K-means clustering to "supervise" the creation of clusters of documents; each cluster corresponds to a meaningful class, partially based on the original taxonomy.

3.5.3.3 Pattern Mining Module

For the Pattern Mining module, we use an algorithm based on the CAEP algorithm by Dong, et al[7]. The resulting learned patterns are in the form of a specific type of association rule, called *Emerging Patterns*, which can be considered strong indications of membership in a class. Emerging patterns (EPs) are itemsets whose supports change significantly from one dataset to another. This is defined by the growth rate: the *growth rate from class A to class B* of an itemset is the ratio of the itemset's support in class A, to its support in class B. For example, if a term *x* is present in 60% of the documents in class A, while it is present in only 1% of the documents in class B, then the growth rate of *x* from A to B is 0.6/0.01 = 60. Each EP, e_i, in class A, is assigned a weight

$$w(e_i) = [\gamma(e_i)/(\gamma(e_i)+1)]\sigma(e_i), \tag{6}$$

where $\sigma(e_i)$ is the support for e_i in A, and $\gamma(e_i)$ is the growth rate from A to B. This weight approximates the conditional probability that an instance *x* is a member of A, given that it contains e_i, scaled by the support of e_i in A. By finding all of the EPs of A that are found in an instance *x*, and summing their weights, one finds the total weight of the indicators that *x* is an element of A; a similar procedure holds for B. Comparing the two weights allows one to assign *x* into either A or B, or leave it unlabeled if the two weights are commensurate. We transform the weights to a 0-1 scale before comparison, using the Cumulative Distribution Frequency (CDF) of raw weights over all of the labeled examples of each class.

For this application, an itemset is simply a set of words that may or may not be found within an individual document. Since these documents are web pages, we strip out headers, navigation menus, and other extraneous data, using an entropy-based technique similar to the one described by Y_i, et al[15], before passing the documents to the pattern miner. We use a support threshold of 10% and a growth rate threshold of 2. We also consider only itemsets of length less than 10. In practice, the EPs found tend to be of length four or less.

Using this representation, we can create an EP-based relevance filter from the current set of labeled documents, $\mathcal{R}$ and $\mathcal{NR}$ (labeled relevant and non-relevant, respectively). The resulting filter, F, then becomes the new relevance filter for the next iteration of the UDLH process. Table XVI shows some example indicators learned for the relevant and non-relevant senses of *intelligence*. The relevant sense of intelligence revolved around the concepts of "the operation of gathering information about an enemy," rather than the cognitive sense of the term ("the ability to comprehend; to understand and profit from experience"). Note that the terms in the bottom panels seem to bear no relationship to the concepts of interest; we will address this point in the next section.

Table XVI. Example Indicators for query intelligence

Relevant sense of *intelligence*	**Nonrelevant sense of *intelligence***
agency intelligence cold war community intelligence east middle homeland security intelligence military intelligence security	disability learning intelligence quotient intelligence test problem solving psychometric test reaction time attention deficit disorder
ai aircraft intention years	lawyer trial civil service contact eye

We selected this particular representation of patterns because we believe that it is well set up to investigate the processes by which a user can interact with the learning system – for example, to be able to experiment with the actual effects of a given rule or to test the effect of the rule's deletion. In the short term, an interactive system would enable us to address some of the limitations of the current UDLH approach. For example, as mentioned above, the performance of the final filter depends on the precision of the initial labeling heuristic. If the labeling heuristic introduces too large an error into the initial document partitioning, this noise can be propagated by the rule-learning part of the process. A useful feature to have, then, would be the ability for the user to be able to correct the labeling heuristic's partitioning, without laboriously reviewing the entire list. We view this as a visualization research issue, in that the user should be able to review the list quickly and detect obvious mislabelings, and correct them, while the bulk of the partitioning and classification effort remains with the computer.

Another use of interactivity is demonstrated by the indicators in Table XVI. While the patterns in the top two panels do accurately express the concepts in question, the terms in the bottom panels are questionable. In fact, on further inspection, we discover that the "relevant" term *ai* corresponds to several pages about artificial intelligence that have been erroneously classified as relevant. If the user were able to examine the effects of a rule, he or she could identify such spurious or incorrect rules, enabling the system to improve the learner's performance, both in accuracy and computational efficiency.

In addition to being an interesting and useful research direction, the impact of such advisable learning would have benefits beyond metasearch. Learning processes are often "black box". The ability to interact with the learning system in an easily understandable way will enable even users who are not experts on machine learning to understand why the system behaves as it does, and hence, perhaps, what to do about undesired behavior. This in turn would increase the trust that users would have in these automatic processes.

3.6 Future Enhancements

While Buddy currently surpasses the competition, a number of enhancements will be made in the near future. Perhaps the most exciting and challenging of these enhancements is the development of an automated maintenance server that will allow Buddy users to enjoy all of the benefits Buddy has to offer without having to worry about maintaining the search adapters. The maintenance server queries each source at a predefined frequency and compares the results it receives to the results that it expects to find, or those from the previous search. If the results deviate significantly the server will attempt to fix the page by using the links it expected to find to learn the structure of the page. It turns out that results for a particular search may change because the parser has broken, or simply because there are new results. However, initial tests of a very simple technique to distinguish between these two situations have produced very promising results. The key, as shown in the equation below, is to reconcile each new return with each missing return.

$$X \;=\; \#\ \text{New Returns} - \#\ \text{Missing Returns} \qquad (7)$$
$$\begin{Bmatrix} X > 0 -> Broken \\ X = 0 -> Healthy \end{Bmatrix}$$

Upon startup, Buddy queries the central maintenance server to find out what sources are available. The server responds with each source available as well as the date on which each adapter file was last updated. After comparing the timestamps of each local file with the corresponding server timestamp Buddy presents a list of adapters that have changed. Finally, the user selects which adapters, if any, to update.

In addition to the maintenance server, a prototype version of a Buddy Server has also been developed. The server version allows centralized collection and storage capabilities and may facilitate collaboration by allowing clients to share topic trees with each other. Buddy Server also takes advantage of allowing the user to schedule reoccurring collections. This capability allows the user to specify that a particular set of queries should be repeated after the expiration of a specified time interval that may range from minutes to weeks. Both Buddy and Buddy Server create opportunities to develop larger systems that will allow the user to more fully exploit the wide range of existing and future technologies. Recently, a number of emerging technologies were integrated to demonstrate an architecture in which natural language extractors and a graph-based pattern detection capability could build upon Buddy's collection capabilities to alert intelligence analysts when key events occured[4].

4. CONCLUSION

Buddy was designed specifically to help analysts collect open-source intelligence. However, the tailored capabilities Buddy provides extend far beyond the existing limitations of current metasearch technology and make it equally useful to casual web surfers and professional researchers. No other engine offers the breadth and quality of search features in a single package. Buddy allows users to extend their coverage to the web sites most applicable to their information requirements with dynamic source access and maintenance. With offline storage, recursive document retrieval and increased navigational flexibility Buddy can begin to help analysts harness the power of open-source intelligence. Despite tapping into such a vast array of sources, Buddy prevents information overload by focusing the user's attention to the most relevant documents with the customization workbench. Finally, while helping to support existing challenges Buddy may also play a vital role in a new system of systems that integrates a range of intelligent technologies to meet tomorrow's demands.

References

1. Mercado, S. 2004. Sailing the Sea of OSINT in the Information Age. *Studies In Intelligence: Journal of the American Intelligence Professional*. 48(03): http://www.odci.gov/csi/studies/vol48no3/article05.html
2. Boulware, D; Salerno, J; Bleich, R; Hinman, M. 2004. Towards Building a Comprehensive Data Mart. In Proc of the SPIE Conference, Orlando, FL. 12-13 April, 2004, 236-245
3. Marcus, S; Coffman, T. 2004. Dynamic Classification of Suspicious Groups Using Social Network Analysis and HMMs. In Proc of the 2004 IEEE Aerospace Conference. 6-13 March, 2004.
4. Salerno, J; Hinman, M; Boulware, D. 2004. Building a Framework for Situation Awareness. Proc of the International Society of Information Fusion (ISIF) Conference. Stockholm, Sweden. June 2004.
5. Search Engine Watch, http://searchenginewatch.com.
6. Aggarwal, Charu C.; Stephen C. Gates and Philip S. Yu. On Using Partial Supervision for Text Categorization. *IEEE Transactions on Knowledge and Data Engineering*, 16(2), February 2004.
7. Dong, Guozhu; Xiuzhen Zhang, Limsoon Wong, Jinyan Li. CAEP: Classification by Aggregating Emerging Patterns *Discovery Science: Second International Conference,* S. Arikawa, K. Furukawa (Eds.). 1999.
8. Dong, Guozhu and Jinyan Li. Efficient Mining of Emerging Patterns: Discovering Trends and Differences. *Proceedings, SIGKDD Fifth International Conference on Knowledge Discovery and Data Mining*. 1999.
9. *KnowItAll Project Page* http://www.cs.washington.edu/research/knowitall
10. McCallum, A. and Kamal Nigam. Text Classification by Bootstrapping with Keywords, EM and Shrinkage. *ACL Workshop for Unsupervised Learning in Natural Language Processing*, 1999.
11. Popescu, A.M.; A. Yates and O. Etzioni. Class Extraction from the World Wide Web. *AAAI 2004 Workshop on Text Extraction and Mining*. 2004.
12. Saari, D.G. and V. Merlin, The Copeland method 1: Relationships and the dictionary, *Economic Theory*, Vol. 8, pp. 51-76, 1996.
13. Saari, Donald G. and Fabrice Valognes, Geometry, Voting, and Paradoxes, *Mathematics Magazine*, Vol. 7, No. 4, pp. 243-259. 1998
14. *WordNet – Princeton Cognitive Science Laboratory.* http://wordnet.princeton.edu/index.shtml
15. Yi, Lan; Bing Liu and Xiaoli Li. Eliminating Noisy Information in Web Pages for Data Mining. *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (KDD-2003), Washington, DC, USA, August 24 - 27, 2003.
16. Freund, Yoav; Raj Iyer, Robert E. Schapire and Yoram Singer. An Efficient Boosting Algorithm for Combining Prefe-

rences. *Journal of Machine Learning Research*, Vol. 4, 2004.

17. Lebanon, Guy and John Lafferty. Cranking: Combining Rankings Using Conditional Probability Models on Permutations. *Proceedings of the International Conference on Machine Learning (ICML-2002)*, 2002.