Computer Science
Technical Report

Colorado State University

An Information Gathering Agent for Querying Web Search Engines *

Daniel Dreilinger Adele E. Howe
Computer Science Department, Colorado State University
Fort Collins, CO 80523
email: {dreiling,howe}@cs.colostate.edu
url: http://www.cs.colostate.edu/~{dreiling,howe}

Technical Report CS-96-111

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*This research was supported in part by ARPA-AFOSR contract F30602-93-C-0100 and NSF Research Initiation Award #RIA IRI-930857

An Information Gathering Agent for Querying Web Search Engines *

Daniel Dreilinger Adele E. Howe
Computer Science Department, Colorado State University
Fort Collins, CO 80523
email: {dreiling,howe}@cs.colostate.edu
url: http://www.cs.colostate.edu/~{dreiling,howe}

Abstract

Information gathering agents have attracted much attention of late. As a new application, they are attractive because the need for intelligent assistance in navigating the World Wide Web and large databases is acute. Information agents provide an open-ended and complex, yet easily accessible environment in which ideas from many areas can be integrated. We have developed an information gathering agent called *SavvySearch* for intelligently searching multiple search engines on the Web. SavvySearch tracks responses from existing search engines to manage resource usage and submit queries only to the most appropriate search engines.

To implement SavvySearch, we adapted simple ideas from machine learning, information retrieval and planning and tested two issues in the designs: Can search engine selection knowledge be acquired to improve performance? Do users find that high quality results are being returned early within the limited parallelism provided by *SavvySearch*? Current results indicate that automated acquisition does lead to good selection performance and restricting resources through ordering parallel searches can still produce satisfied users.

*This research was supported in part by ARPA-AFOSR contract F30602-93-C-0100 and NSF Research Initiation Award #RIA IRI-930857

1 Information Gathering on the Web

Without help, finding a specific site or particular information among the deluge of informally connected sites on the World Wide Web is difficult. Tools for searching the Web, such as Lycos [15] and DejaNews [14], match user queries to Web resources of interest. These programs act as directories and indexes to the Web, considerably alleviating the navigational difficulties. However, to some extent, they push the problem back one level; now one needs to navigate the search engines: know their location, their language or user interface and their areas of expertise.

Each search tool is limited by the sub-corpus of documents known to it and by its ability to find relevant documents within that corpus. It has been suggested that search engines should not attempt to thoroughly index the entire Web space, as that will lead to much repeated effort [16]. A user with a specific information need will often need to query several search engines before finding relevant documents.

To address the problem of navigating the search engines, we have developed a *meta-search* agent, called *SavvySearch*, that directs queries to search engines judged to be most appropriate. The agent balances expected resource consumption against the expected benefits of submitting queries to multiple search engines. To do so, the agent collects, maintains and integrates knowledge about the current and past function of the search engines and the Internet. Because the application of meta-search is relatively new, SavvySearch was designed to provide a framework for investigating the merits of various approaches; it is easily extensible to other search engines and has a modular design that expedites replacing components.

In this paper, we describe the architecture and one of the implementations of SavvySearch which borrows, in a simple way, ideas from machine learning, information retrieval and planning. We tested two issues in its design: Can search engine selection knowledge be acquired to improve performance? Do users find that high quality results are being returned early within the limited parallelism provided by SavvySearch?

2 Approaches to Information Gathering

Agent-based information retrieval research is becoming incredibly popular. Last year's Spring Symposium on Information Gathering included a large number of papers on diverse agents [10]. Some of the agents focused on the Web as an information source; agents have been designed for filtering, browsing [2,13], and traversing the Web [1], as well as searching specific information on it (e.g., FAQ files for newsgroups [9]) and heterogeneous sources [17].

Agents often reference domain dependent databases and utilize a complex logical or semantic domain model [17,11,6]. For example, [6] describes a distributed purchasing agent that aids in location and pricing of products. An internal predicate logic repre-

sentation relates product descriptions to information domains. Using output from one information source as input for another is a common technique in domain-specific systems. These systems divide their functionality among diverse agents, exploit parallelism where appropriate, and gracefully deal with the imminent failures and changes associated with widely distributed processing. While these are all concerns for our system, our application is much less comprehensive (i.e., does not model goals or treat information gathering as just one task in a larger one) and so does not require as complex a solution.

2.1 Searching the Web

Finding a Web resource is not always a trivial task. The naive approach is to start with your home page and follow links that intuitively seem to lead you in the direction of the resource you are looking for. This is roughly analogous to wandering around the country side hoping to find an address by following road signs, but lacking a map. In fact, the WebWatcher project [1] addresses just this problem by assisting users in following links from starting at a general subject Web page (e.g., a Machine Learning page with links to other sites) to finding the required, related information. WebWatcher learns to predict links likely to be of interest to a user by passively "watching" the user traverse links and offering suggestions.

Alternatively, if you have used the World Wide Web, you have no doubt encountered some of the myriad search services, such as Yahoo [7] and Web Crawler [18]. These powerful search engines can aid the process of searching on the Web. Search results are displayed in the form of another Web document consisting of a list of links that are deemed relevant to the search query. Unfortunately, the search engines only search some subset of the millions of Web documents and thus any given search engine may not return all desired references. For example, the Web Crawler searches a very large database of automatically retrieved and indexed URL references for every occurrence of the query. Yahoo, on the other hand, searches a manually created index of high level Web documents. Thus, Yahoo's results are typically more relevant, but at the cost of returning only a fraction of the quantity that Web Crawler returns.

The availability of many search engines leads to the *text-database discovery problem* [8]. In most information retrieval, a corpus of documents is the target of queries; in the text-database discovery problem, the corpus of documents is reached *through* other databases. Thus a query is matched not to a set of documents but rather to a set of databases or, in this case, search engines.

The *GlOSS* (Glossary-of-Servers Server) project [8] has suggested a solution to the text-database discovery problem. A meta-index is constructed by integrating the indexes of each of the databases. For each database and each word, the number of documents containing that word is included in the meta-index. When a query is submitted to GlOSS, relevant databases are selected by using the meta-index to anticipate the ones that will probably produce relevant results.

A disadvantage with GlOSS is that each of the search engines must cooperate with the meta-searcher by supplying up-to-date index information. This may add a prohibitive amount of administrative complexity as the number of databases increases. While our solution involves a meta-index similar to that described in [8], it differs in that no assumption is made about the availability of search engine indexes, which are often unavailable. Instead, information for the meta-index is accumulated incrementally by learning from data on actual user queries and the results returned by the search engines.

2.2 Searching the Web in Parallel

The natural response to the problem of too many places to search is an additional level of abstraction [3], which we call *meta-search*. One variety of meta-search tool is simply a list of pointers to possible search sites, e.g. [12,4]. These indexes are useful in that they increase the user's awareness of where they can search, but they are inconvenient. The decision of where to search is relegated to the user, who has to enter and submit their query repeatedly. Furthermore, the user must interpret the heterogeneous results which appear in the native formats of the various search engines.

These shortcomings motivate true meta-search engines, such as SavvySearch (the system described here) and MetaCrawler [20]. MetaCrawler distributes user queries in parallel to all search engines within its knowledge base. The full text of each result is retrieved from the Web and then ranked using conventional information retrieval techniques. An added benefit of retrieving the documents is the ability to apply a sophisticated query language. MetaCrawler is very successful at pruning irrelevant results and imposing a standardized ranking.

One important issue of parallel meta-search is where specifically to send the query; it is conceivable that just a handful of databases must be selected from hundreds. Other issues include how to deal with failure, how to evaluate the results, and how to facilitate extensibility.

3 SavvySearch

The goal of SavvySearch is to assist Internet users in finding relevant information by submitting their queries to multiple search engines[1]. This operation must be performed in the presence of two goals: minimizing resource consumption and maximizing search quality. Resource limitations make it impractical to send every query to every known search engine and programs that did so would be considered to be poor citizens of the Web [5]. Furthermore, it is undesirable; users are better served with a small set of results (less than 30) from related search engines rather than being inundated with quasi-relevant information, and having to wait for the privilege. On the other hand, too few search

[1] SavvySearch can be accessed on the Web at http://savvy.cs.colostate.edu:2000/.

engines might produce no results, subjecting the user to the inconvenience of retrying and waiting.

Search engine selection decisions are based upon information that can be dynamic in nature. Characteristics of search engines change over time as indexes are updated and algorithms are improved. Consequently, the association between expertise and search engine is likely to change as well. Other qualities modulate even more frequently (e.g., average response time for a given server might be faster at night and slower during the early afternoon).

Information gathering on the Web can be viewed as a simple planning problem in which the goal is to find sites satisfying specific criteria and where the actions are queries to search engines. Search plans are constrained by the resources available: how much time should be allocated to the query and how much of Internet resources should be consumed by it. Consequently, in this view, a plan to gather information for a specific query consists of a sequence of parallel queries to search engines where the user gets to decide at intermediate points whether further searching, and thus resource consumption, is necessary or desirable.

Figure 1 shows the results interface for this design and the plan for the query: *bach cello suite*. A query consists of a set of *terms* that express the information desired and selection of a query operator (*AND*, *OR*, *Adjacency*) to be inserted between all terms. We opted for a simplified query language to maintain simplicity in the interface and to expedite connection to as many search engines as possible. When the query is submitted, SavvySearch constructs a plan, a sequence of parallel searches, and executes the first step. The user can view the results of the first step and decide whether to execute a later step or follow a link in the current results. The following subsection will describe how this example plan is created.

We learned from an earlier design that users should be able to exert some control over the search engine selection. They may be resolving ambiguities, have some knowledge that SavvySearch does not or may simply wish to be "in control". As a consequence, while the information gathering plan is started automatically, further execution requires user selection.

3.1 Architecture

Our agent-based view of a meta-search tool architecture comprises three types of agents: dispatch, display, and search engine interface (Figure 2). Of these agents, the interface agents use HTTP to communicate with remote, Web-accessible search engines; otherwise, the agents communicate only with each other and the user. The purpose of the interface agents is to facilitate incorporating a variety of search engines; each interface agent is designed to accommodate its target search engine's user interface and failure modes.

SavvySearch relies on considerable knowledge about current conditions and relation-

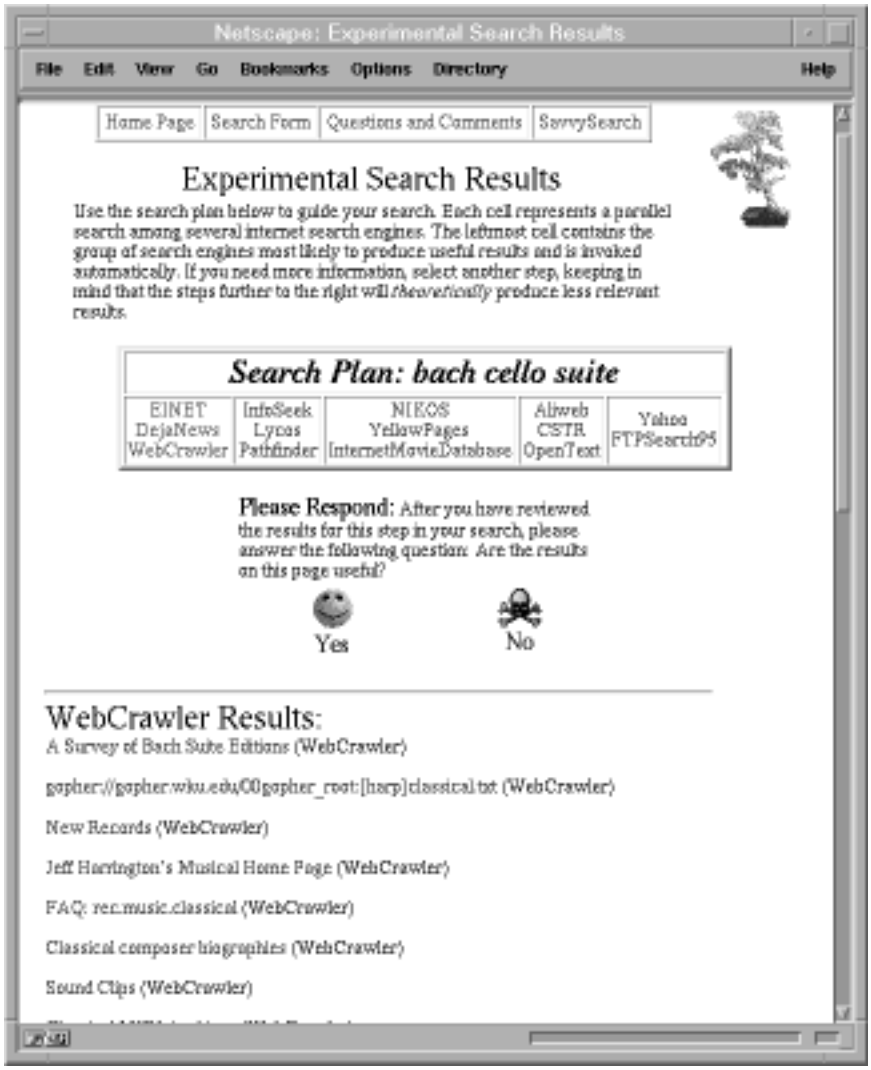

Figure 1: User interface from experimental version showing presentation of results of a query.

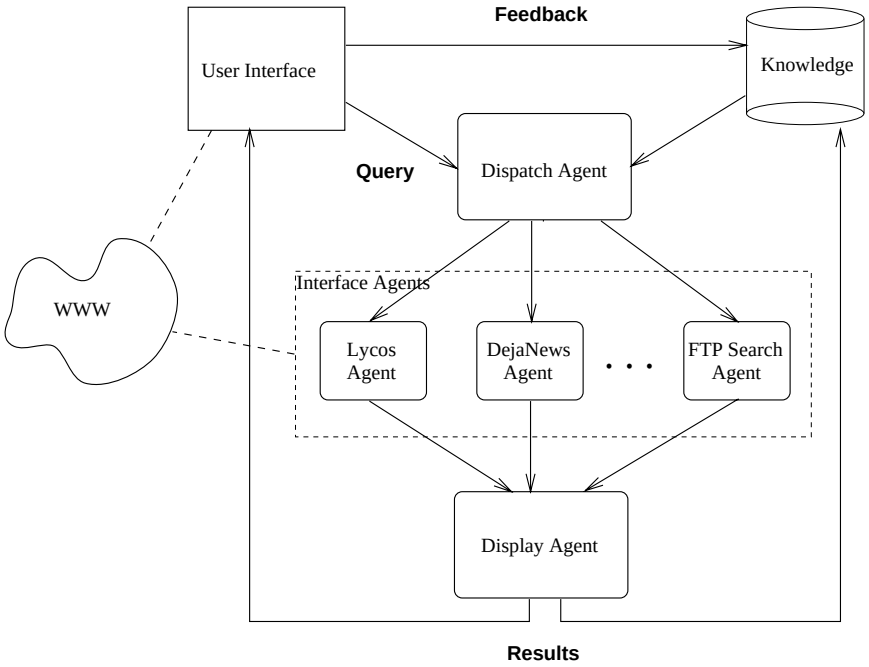

Figure 2: Multiple agent architecture for processing queries.

ships between queries and search engines. The core of that knowledge is the *meta-index* which tracks the effectiveness of each search engine in responding to previous queries. Each word that has been used in a query has an entry in the meta-index; that entry summarizes previous performance of search engines that have been queried using that word. Higher values represent more responses to a query containing that word. The acquisition of the supportive knowledge will be described in section 3.2.

3.1.1 The Dispatch Agent

The dispatch agent uses the query and information from a knowledge base to rank the set of interface agents and determine an initial group to which the query is first submitted. Given a query, a list of search engines, and some knowledge about the search engines and network conditions, the dispatch agent produces a rank-ordering of search engines with respect to their fitness to the query and impact on network conditions. Resource reasoning is included to balance network resource consumption against response quality [21]. Much information can be obtained from a query; for example, we can look at properties of specific search terms used, or we can introduce ancillary query components, such as the general nature of information sought.

The dispatch agent in this version of SavvySearch creates a search plan in a two step process: first, search engines are ranked with respect to the query and current network conditions – those predicted to perform the best are ranked highest. Then concurrency,

term	Lycos	Web Crawler	Yahoo	DejaNews
bach	4	21	3	0
cello	2	1	0	1
suite	*undefined*			
Search Engine Total	500	1000	140	200

Table 1: Example of Meta-Index Data

or the number of search engines to search at the same time, is computed.

1. Search engines are ranked by predicting which are most likely to return useful results. This is done by combining a query score with current data on the expected waiting time and number of results.

   - The query score is determined by looking up the meta-index feedback weight for each search engine and query term pair. The weights are divided by the total feedback of their respective search engines. Finally, the weights for each search engine are added to compute initial search engine scores. Thus, the query score for search engine s and query string q is

   $$Q_{s,q} = \sum_{t \ \epsilon \ q} \frac{W_{t,s}}{T_s},$$

   where $W_{t,s}$ is the meta-index entry corresponding to a single query term (t) and search engine s; and T_s is the sum of all meta-index feedback weights for search engine s.

   For example, Table 1 shows simplified meta-index entries for the terms *bach*, *cello*, and *suite*, as well as the sum of all meta-index data for each of four search engines. Using the query score formula, we can compute the score for query *bach cello suite* and search engine Lycos ($Q_{Lycos,q}$) by adding the scores for the three terms: $\frac{4}{500} + \frac{2}{500} + \frac{0}{500} = 0.012$. Similarly, $Q_{Web\ Crawler,q} = 0.022$, $Q_{Yahoo,q} = 0.0214$, and $Q_{DejaNews,q} = 0.005$.

   - Scores are fine tuned by adding recent performance information for each search engine. The five most recent queries for search engine s are used to compute the average number of hits (H_s) and average response time (T_s). The performance information for search engine s is combined in the fraction

   $$P_s = \frac{H_s}{100 \cdot T_s}.$$

The denominator is multiplied by an arbitrary constant 100 to make the performance data an order of magnitude smaller than the score data. The desired effect was to rank the better search engines utilizing meta-index data alone, then rank the search engines with no meta-index data using externally observable search engine characteristics.

The two computations just described are combined to determine the overall rank $R_{s,q}$, for search engine s and query q:

$$R_{s,q} = Q_{s,q} + P_s.$$

2. The purpose of concurrency calculation is to reduce the resources demanded by SavvySearch in periods of high network and machine demand. So concurrency is inversely proportional to estimated query cost: the more it costs to submit search engine queries at present, the fewer search engines will be queried. Concurrency is computed from three cost variables: expected network load, local CPU load, and query discrimination value. Each of these contributes to a concurrency sum by adding some value up to two.

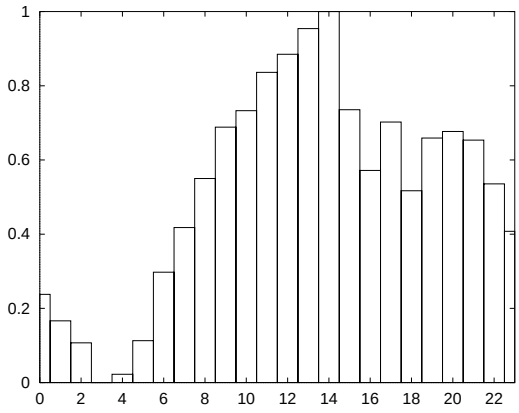

Figure 3: Relative traffic throughout the day.

- Ideally, the *expected network load* should estimate the current traffic on the Internet; since this is difficult to assess, we estimate it based on the traffic we have seen on our nodes (number of queries per unit time) in the past at this time of day. So expected network load is computed statically from the log files produced by our Web server (as in Figure 3) by looking up typical load for this time of day. During periods of low network load (e.g., 3 AM), a high value (up to 2) is contributed to the concurrency, and vice versa.

- The *local CPU load* is computed from current system load; the output from the *UNIX uptime* command is mapped from the interval $0 \ldots 3$ onto the interval $2 \ldots 0$. It is important to incorporate both types of load (CPU and net traffic) because the local CPU load fluctuates constantly and therefore is not a good indicator of the overall network activity.
- The *discrimination value* addresses how much search effort is likely to be needed to find a satisfactory response by measuring how specific or general the query is. If a term has a relatively large amount of meta-index data, for example *bach* in Table 1, the query presumably represents an area that many search engines cover, and thus, we suspect fewer search engines will need to be queried. If the term has little or no data, for example *cello* or *suite*, our current knowledge of search engine expertise in this area may be inadequate or it may be a topic area not covered by many search engines, and thus we will need to search more places to find relevant documents.

  The discrimination value is calculated by dividing the total number of references counted for the term (i.e., the sum of a row in Table 1) by the number of references of the most frequently referenced term in the database (i.e., the same sum but for the most dominant term in the meta-index). An average is used for multi-term queries.

3.1.2 Interface Agents

Interface agents translate the query to a form appropriate for its specific Internet-accessible search engine, submit it and retrieve the results. Each search engine has a different interface and capabilities. For example, some systems require query terms to be joined with boolean operators (i.e., AND, OR), while other systems implement this as a check-box option on the search form.

We build one interface agent for each search engine that we can query. These agents are designed to gracefully respond to exceptional conditions such as errors or unexpected results. Results are forwarded to the display agent in a standardized internal format.

Along with the query, the interface agents can also be given (or maintain internally) additional information including dynamic qualities of their target search engine like average response time, and resource constraints that limit the number of retries in the event of failure. Interface agents employ basic failure coping abilities such as retrying a query in the event of a remote failure.

3.1.3 The Display Agent

The display agent organizes the results from all the interface agents and displays them in an easily interpreted manner to the user. One trade-off that must be addressed is response time versus organization of results.

At present, the display agent is simple. The top k results, where k defaults to 10, for each search engine are displayed in the order in which they arrive. As results are collected and processed from each search engine, the results are displayed in the order suggested by the search engine. Interleaving was avoided because it requires waiting for results from all search engines before beginning the display. Since part of the purpose of the resource reasoning was to create an iterative process in which the user decides when to search further, waiting was deemed inappropriate.

Alternatively, we can wait until all results have arrived, then impose a ranking function based on variables such as scores given by target search engines and local quality measures of the search engines for a particular query. It is conceivable to retrieve the full text of all referenced documents and perform additional conventional information retrieval on them. This is a particularly appealing approach if waiting time is not a priority and has been implemented effectively in the MetaCrawler system [19].

3.2 Acquiring Supporting Knowledge

The supporting knowledge includes static and dynamic knowledge about costs and the successes of queries. Thus far, we have measured cost based on network load, local CPU usage, and expected quality of information. We collect statistics regarding typical usage on our server during different periods during the day and use that profile for CPU usage.

The knowledge of query success is constructed from passively accumulated user feedback and is stored in a meta-index. The meta-index is an $n \times t$ matrix, where n is the number of search engines and t is the number of terms that have been used. Each entry measures the effectiveness of a specific search engine-term combination. This structure is used extensively in classical Information Retrieval in the context of mapping keywords to documents.

Effectiveness values in the meta-index were determined from a log of about $50,000$ queries acquired by an earlier version of the system. Using the heuristic that if a user follows a link it is relevant, the weight matrix was adjusted by incrementing the word values of the query terms for the particular search engine(s) that produced the relevant result. An increment of one was used for all search engine-term feedback events, regardless of query length.

For example, the meta-index entry for the word *bach* (shown in Table 1) indicates that 4 documents returned by Lycos for queries containing the word *bach* were deemed relevant by users (i.e., resulting links were selected for further perusal), 21 links produced by Web Crawler, 3 from Yahoo, and none from DejaNews. While the meta-index will be constantly updated during normal usage, it was held constant for our experiments to reflect both new and changing information.

4 How Well Does the Design Work?

As with any system involving a user, the ultimate test is user utility. We have measured utility in a variety of ways: unsolicited user comments, the number of links followed for given queries and explicitly requested user feedback. We wanted to test whether their underlying techniques improved user utility.

In most general terms, SavvySearch has been a great success. It has been available since April 1995 in the form described here and a previous design. In that time, we have received dozens of favorable comments and the daily use has steadily increased to an average of about 20,000 queries per day.

In our evaluation, we choose to stress determining what aspects of this design seem to work. We view the current design as evolving; we will continue to improve it based on what we learn from experiments after this. We also intend that others should learn about what works and what does not from our experiences so that they can build their own Web information gathering agents. At present, SavvySearch is one of a few meta-search agents (another notable one being MetaCrawler), and MetaCrawler is significantly different in its focus; thus, we have avoided a "who's best" comparison as being too coarse to learn anything meaningful about how such systems should be designed.

We expect that information quantity results directly from resource expenditure, but that information *quality* also results from careful deployment of resources (i.e., queries to search engines). Two aspects of the current design contribute to resource deployment: the allowable resource calculation and the selection of search engines. Plan size is directly proportional to the current resource demands, and plan sequencing depends critically on the search engine selection process.

We ran a series of experiments to determine whether characteristics of the current design affected the quality of the information displayed to the user. We measured quality in two ways: number of links followed per query (link-ratio) and reported user satisfaction. Of the two, we think that link-ratio is the more reliable measure because it tracks what interested the user enough to follow up, it requires no additional effort on the user's part, it should be less susceptible to user bias (e.g., users might give positive feedback just to be nice or negative to encourage us to further improve the system), and because most users did not know that we were collecting this data.

We learned from a pilot experiment that we cannot completely trust direct feedback from the Internet community. As might be expected some users are unwilling to take the time to provide direct feedback, and some users provide noisy feedback (e.g., ballot stuffing or apparently random selections). Unfortunately, we found that the diverse Internet community cannot be directly responsible for making heavily weighted changes to any aspect of the system (an earlier scheme tried a weighting system based on user feedback) and must be eyed skeptically in determining system success. Consequently, we will report the feedback summary data, but we did not use it as the primary quality measure in the analysis.

We wished to answer two questions about the design:

- Does plan order matter? More precisely, are plan steps being ordered so that higher quality results are returned early in the plan?
- Is the selection strategy adequate? More precisely, does quality depend upon which search engines are included in each plan step?

If the current approach is appropriate, then ideally, users will find what they want early in the process of executing the plan; if they do not, we need to know whether the plan sequencing or the selection strategy are at fault.

To address these questions, we collected data on four variants of this design:

Approach A the basic design described in section 3,
Approach B basic design except that the step to be executed first is selected randomly, thus the first display may be generated from any step in the plan,
Approach C basic design except that the search engines are randomly selected without replacement for inclusion in each step of the plan,
Approach D basic design except that the first plan step executed is selected randomly and the search engines are selected randomly.

So for the example given the plan in Figure 1, approach A would immediately execute the first step and display the results from those three search engines; approach B would immediately execute one of the steps as selected at random and display those results; approach C would construct a different search plan from A and B in which the search engines are randomly ordered and would immediately execute the first step; and approach D would construct a different search plan from A and B in which the search engines are randomly ordered and would immediately execute one of the steps selected at random.

These four versions vary along two dimensions: selection of the search engines and plan ordering. Approach A is our target design; the other three are controls which allow us to address the empirical questions. In all cases, the degree of parallelism was based on load.

We conducted the experiments by adding a link from the primary SavvySearch page (by now in common use) to what we described as our "new experimental search tool". Thus, users were not surprised to find a different version and self-selected whether they wanted to participate. We ran each variant for about two days, enough time to collect data on at least 2500 queries. The interface was identical for each one; users were told that a search plan would be constructed in which expected usefulness of the steps was displayed from left to right.

Data was collected on each query processed: number of steps in the plan, which steps were executed, how many links were followed in executed steps, which search engines were queried, whether it was a follow-up query and whether or not the user expressed satisfaction with the query (lack of answer was separated from a negative response). For

Plan Length	3			4				5					7						
Plan Step	1	2	3	1	2	3	4	1	2	3	4	5	1	2	3	4	5	6	7
Requests	296	32	19	1684	227	149	96	416	72	51	37	34	31	7	5	6	6	3	4
Visits	521	19	14	3234	289	71	42	744	92	71	27	15	50	11	10	4	11	2	4
Yes	40	7	1	252	30	16	15	51	5	5	3	2	5	2					1
No	8		2	82	13	20	15	18	2	2		2	3				1		

Table 2: Approach A raw survey data on both performance measures. Blank cells indicate no data available.

analysis, we summarized the data by the two quality measures and separated it by plan length and plan step. Plan length indicates how many were required to query all the search engines given the current level of parallelism; so if plan length was 5, each of the plan steps contained 3 search engines except the last which had 2 (14 search engines were included). Plan step indicated which step in the sequence was being executed.

Overall, we received 10,575 requests which resulted in users following 19,072 links. Users followed 2.0 links per plan step for approach A, 1.76 links per step for approach B, 1.89 links per step for approach C and 1.55 links per step for approach D. We received 2,082 answers to whether the user found the results useful for about a 20% response rate; self-reported satisfaction was 72% for approach A, 60% for approach B, 65% for approach C and 60% for approach D. As an example of the numbers involved, Table 2 shows the raw data counts for approach A. On the summary numbers, the basic approach (approach A) exhibits the highest quality performance on both measures. Presenting the user with the results of what was expected to be a lower quality solution (approach D) is the least satisfactory.

Does Plan Order Matter? We tested this question by analyzing whether the quality of results depended on which plan step was executed. We analyzed separately the data from each approach and from each plan length. For each approach and plan length, we ran a chi-square contingency table test with quality as the dependent variable and plan step as the independent.

We expected that quality would depend on plan step for approaches A and B, but not for C and D because C and D simply selected the composition of steps randomly. In fact, for all but one plan length for approaches A and all plan lengths for approach B, we found a statistically significant effect of plan step on quality ($P < .01$). For approach C for all but one plan length, and D for two lengths, we found no significant effect of plan step on position (in these cases, the lowest was $P < .20$). The one exception in approach A was plan length of seven in which every step included only two search engines; we hypothesize that a plan length of seven with a parallelism of only two may force too

much emphasis on particular search engines. In both C and D, in the cases with the most data (lengths 3 and 4), a decline in following links due to position was enough to be significant. Our best guess is a placebo effect of assuming that the ordering is good and not looking too closely at the contents of later steps.

Is the selection strategy adequate? Addressing this question involves comparing the approaches themselves. Because of the number of factors involved (plan length, plan step and approach), we analyzed the data by comparing pairs of approaches while holding constant the plan steps and lengths. We compared approach A to approach C and approach B to approach D because each used the same method for determining which plan step to execute first. To reduce the amount of analysis, we selected the most common plan length for the comparison: plan length four. For each pair of approaches and each plan step, we constructed a two by two contingency table with a row for each approach, one column for number of queries and one column for links followed.

Quality was significantly higher for approach A than for approach C on steps one, two, and four ($P < .01$), but while higher on step three, the difference was not statistically significant ($P < .37$). For approaches B and D, quality was significantly higher for three steps ($P < .05$); for step 2 it was higher but not significantly ($P < .06$).

5 Conclusion

The tool described in this paper is just one of a set of implementations that we have tried and will be trying. A previous version relied solely on user-supplied categories, as opposed to the terms in the query itself, for selecting search engines and submitted queries to a fixed number of search engines.

Our current focus is on improving search engine selection and evaluating the role of the user. We are in the process of designing and running experiments to test more complex meta-index acquisition techniques. In particular, the newer schemes incorporate both positive and negative feedback and amortize term effectiveness by the number of term in the query (i.e., longer queries receive less weight per word). These schemes adapt information retrieval techniques from the standard paradigm of searching a corpus to searching a set of searchers where the corpus is unknown. Additionally, we are investigating re-introducing categories as a simple mechanism for the user to disambiguate word meaning.

More consideration has to be given to the user's role in the search process. For example, two different modes of Internet information acquisition should be accommodated: *searching* for specific information, and *browsing* (a.k.a. "net surfing") for interesting information [2]. User's in the former mode might need to see results from a greater number of search engines, while those in the latter model might be satisfied with results from fewer sources. Ideally, these two modes could be further decomposed. Approaches to this

might evaluate different levels of interactivity (e.g., more parallel searching with more sophisticated results presentation and followup) and different styles of interaction (e.g., searching versus browsing).

In terms of experimentation, we still need to satisfactorily assess the effect of reducing parallelism on user satisfaction. We know from the current data that on average users execute only one to two plan steps. However, an experiment to gauge the exact effect of reducing parallelism produced inconclusive results. We are formulating a new experiment to address the effectiveness of current resource reasoning.

Meta-search clearly satisfies a need for searching the vast Web space. The architecture described here is based on simple applications of machine learning, information retrieval and planning. Studies such as ours expedite integrating diverse techniques by indicating whether they are even worth consideration. For example, a planning framework seems to be a useful paradigm for giving the user control and the agent-based architecture facilitates extensibility and mitigates domain dependence. We intend to enhance the current design with more sophisticated techniques. We expect that as search engines proliferate and specialize, meta-search engines will continue to be important and will be a useful resource for both human users and for other information gathering agents.

References

[1] Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. WebWatcher: A learning apprentice for the World Wide Web. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[2] Marko Balabanović and Yoav Shoham. Learning information retrieval agents: Experiments with automated web browsing. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[3] C. Mic Bowman, Peter B. Danzig, Udi Manber, and Michael F. Schwartz. Scalable internet resource discovery: research problems and approaches. *CACM*, 37(8), August 1994.

[4] William Cross. All-in-one search page. http://www.albany.net/allinone/.

[5] David Eichmann. Ethical web agents. In *Electronic Proceedings of the Second World Wide Web Conference '94: Mosaic and the Web*, 1994. http://www.ncsa.uiuc.edu/SDG/IT94/Proceedings/Agents/eichmann.ethical/ethics.html.

[6] Richard Fikes, Robert Engelmore, Adam Farquhar, and Wanda Pratt. Network-based information brokers. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[7] David Filo and Jerry Yang. Yahoo home page. http://www.yahoo.com.

[8] Luis Gravano, Héctor García-Molina, and Anthony Tomasic. Precision and recall of *gloss* estimators for database discovery. In *Proceedings of the 3rd international Conference on Parallel and Distributed Information Systems (PDIS'94)*, 1994.

[9] Kristian Hammond, Robin Burke, Charles Martin, and Steve Lytinen. FAQFinder: A case-based approach to knowledge navigation. In Craig Knoblock and Alon Levy, editors, *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[10] Craig Knoblock and Alon Levy, editors. *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[11] Craig A. Knoblock. Integrating planning and execution for information gathering. In *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.

[12] Martijn Koster. Configurable unified search engine (cusi). http://pubweb.nexor.co.uk/public/cusi/doc/about.html.

[13] Henry Lieberman. Letizia: An agent that assists web browsing. In *Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence*, pages 924–929, Montreal, Canada, August 1995.

[14] Steve Madere. DejaNews research service. http://dejanews3.dejanews.com/.

[15] Michael Mauldin. Lycos, the catalog of the internet. http://www.lycos.com/.

[16] Michael L. Mauldin and John R. R. Leavitt. Web agent related research at the center for machine translation. Presented at the SIGNIDR meeting, August 4, 1994 in McLean, Virginia, August 1994.

[17] Tim Oates, M. V. NagendraPrasad, and Victor R. Lesser. Cooperative information gathering: A distributed problem solving approach. Technical report, University of Massachusetts, 1994.

[18] Brian Pinkerton. Web Crawler home page. http://webcrawler.com/.

[19] Erik Selberg and Oren Etzioni. The MetaCrawler WWW search engine. http://metacrawler.cs.washington.edu:8080/home.html.

[20] Erik Selberg and Oren Etzioni. Multi-service search and comparison using the MetaCrawler. In *Proceedings of the 4th International World Wide Web Conference*, December 1995.

[21] Shlomo Zilberstein. An anytime computation approach to information gathering. In *Working Notes of the AAAI Spring Symposium Series on Information Gathering from Distributed, Heterogeneous Environments*, Palo Alto, CA, 1995.