ARTICLE IN PRESS

ELSEVIER

Decision Support Systems 992 (2002) xxx–xxx

Decision Support Systems

www.elsevier.com/locate/dsw

Complementing search engines with online web mining agents

Filippo Menczer

Department of Management Sciences, The University of Iowa, Iowa City, IA 52242, USA

Abstract

While search engines have become the major decision support tools for the Internet, there is a growing disparity between the image of the World Wide Web stored in search engine repositories and the actual dynamic, distributed nature of Web data. We propose to attack this problem using an adaptive population of intelligent agents mining the Web online at query time. We discuss the benefits and shortcomings of using dynamic search strategies versus the traditional static methods in which search and retrieval are disjoint. This paper presents a public Web intelligence tool called *MySpiders*, a threaded multiagent system designed for information discovery. The performance of the system is evaluated by comparing its effectiveness in locating *recent*, *relevant* documents with that of search engines. We present results suggesting that augmenting search engines with adaptive populations of intelligent search agents can lead to a significant competitive advantage. We also discuss some of the challenges of evaluating such a system on current Web data, introduce three novel metrics for this purpose, and outline some of the lessons learned in the process.
© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Web mining; Search engines; Web intelligence; InfoSpiders; MySpiders; Evaluation metrics; Estimated recency; Precision; Recall

1. Introduction

With the increasing amount of information available online, the World Wide Web has rapidly become the main source of competitive intelligence for businesses, and consequently search engines represent invaluable decision support tools. However, disciplines like information retrieval and machine learning are facing serious difficulties in dealing with the Web in a scalable way. The search technologies employed on the Web are faced with several features of this environment that make the task of indexing the Web a formidable challenge. As of February 1999, the size of the publicly indexable Web was reported to be over 800 million pages [17], more than double that reported only a year earlier [15] and growing at a very fast pace—Google alone reports over 1.6 billion URLs indexed at the time of this writing. Furthermore, the Web has a dynamic nature, with pages being added, deleted, moved, updated, or linked to each other at a fast rate and in an unprincipled manner. It is also heterogeneous, without imposed restrictions on document content, language, style, or format.

Search engines are powered by state-of-the-art indexing algorithms, which make use of automated programs, called *robots* or *crawlers*, to continuously visit large parts of the Web in an exhaustive fashion. The pages accessed undergo some performance-enhancing steps, such as the removal of "noise" words (too frequent to be representative), the conflation of terms via stemming and/or the use of thesauri, and the use of different weighting schemes to represent indexed pages. Then, the pages are stored in inverted index databases. Such indexes map every term from a

E-mail address: filippo-menczer@uiowa.edu (F. Menczer).
URL: http://dollar.biz.uiowa.edu/~fil.

0167-9236/02/$ - see front matter © 2002 Elsevier Science B.V. All rights reserved.
PII: S0167-9236(02)00106-9

controlled vocabulary onto the collection of URLs containing that term, and are used at query time to return the sets of URLs containing the terms in users' queries.

The key idea of all search engines is that the index built by a crawl of the Web can be used many times, so that the cost of crawling and indexing is amortized over the many queries that are answered by a search engine by looking up the same static database. The two factors that differentiate among distinct search engines are the crawling and ranking algorithms. Crawling strategies determine which documents are indexed and ultimately affect the *recall* (fraction of relevant pages that are retrieved) and the *recency* (fraction of retrieved pages that are current) of a search engine. Ranking algorithms determine how the documents matching a query are sorted and ultimately affect the *precision* (fraction of retrieved pages that are relevant) of a search engine. For example, the recent success of the Google search engine is generally attributed to its ranking algorithm, which is based on weighted counts of links pointing to pages rather than on lexical similarity between pages and query [13].

While the indexing approach is ideal for static collections of documents, in the case of the World Wide Web, the sets of relevant pages draw one important characteristic from their environment: they are highly dynamic. The static snapshots of the Web captured in search indexes, left to themselves, become less and less accurate and complete. They need to be updated periodically, by having the crawlers revisit the same documents to assess the changes that might have occurred since the last visit—in addition to looking for new URLs. Therefore, as any Web site administrator can easily verify by inspecting server logs, search engine crawlers impose a huge load on network resources, even if this is amortized over multiple queries.

Search engine users are often faced with very large hit lists, low recall, low precision, and low recency. Despite increasing computing power and network bandwidth, no search engine is able to index more than 16% of the Web, and it takes a search engine an average of 6 months to index a new document [17].

These problems are explained by the simple observation that the static "index snapshot" approach does not scale with the growing size and rate of change of the Web. This *scalability* issue is at the root of the discrepancies between the current Web content and its snapshots represented by search engines' indexes. Further, such discrepancies can only grow with the size of the Web, diminishing the expected level of performance of search engines and hindering their effectiveness as decision support systems for competitive intelligence (Web intelligence tools).

The scalability problem posed by the size and dynamic nature of the Web may be addressed by complementing index-based search engines with intelligent search agents at the user's end. In this paper, we discuss a type of agent-based systems inspired by ecological and artificial life models. Such agents can browse online on behalf of the user, and evolve an intelligent behavior that exploits the Web's linkage and textual cues [21,23].

Despite its huge size, the Web forms a small-world network, characteristic of social and biological systems, in which any two documents on the Web are on average 19 clicks away from each other [1]. This result suggests that "smart" browsing agents that can distinguish the relevant links inside the pages might be able to find the desired information in a limited time. By using this approach, the relative load of the network may be switched from the blind crawlers employed by search engines to intelligent browsing agents that can reach beyond search engines and mine for new, up-to-date information. Online agents search only the current environment and therefore will not return stale information, and will have a better chance to improve the recency of the visited documents.

InfoSpiders [25,26] is the name of a multiagent model for online, dynamic information mining on the Web, which uses several artificial intelligence techniques to adapt to the characteristics of its networked information environment—heterogeneity, noise, decentralization, and dynamics. In this approach, each agent navigates from page to page following hypertext links, trying to locate new documents relevant to the user's query, having limited interactions with other agents. Agents mimic the intelligent browsing behavior of human users, being able to evaluate the relevance of a document's content with respect to the user's query, and to reason autonomously about future links to be followed. In order to improve the performance of the system, adaptation occurs at both individual and population levels, by evolutionary and reinforcement learning. The goal is to maintain diversity at a global level, trying to achieve a good coverage of all different

aspects related to the query, while capturing the relevant features of the local information environment, i.e., the textual hints leading to relevant pages.

The idea of intelligent online crawlers such as InfoSpiders is to complement search engines in order to improve upon their precision, recall, and recency. Unfortunately, assessing whether one is successful with respect to this goal is a difficult task. Traditional measures used to evaluate the performance of information retrieval systems assume that (i) all systems have global knowledge of the information space in the form of an index, and (ii) the experimenter has knowledge of the relevant set of documents for each test query. Neither of these assumptions holds in the present case. First, a query-driven crawler relies on a search engine rather than building an independent index—indeed, the pages crawled by the agents can be indexed and added to the engine's database, so that the two approaches are cooperating for mutual benefit. Second, nobody knows the real relevant sets for any query because nobody has knowledge of the whole Web. At best, we can know of a *subset* of the relevant set, i.e., an incomplete set of pages judged to be relevant by human experts. Given these difficulties, we propose in this paper a set of empirical measures aimed at approximating the performance of Web mining agents. Using such measures, we rely on the incomplete knowledge in existing search engines and human-maintained directories to assess the value added of query-driven crawlers with respect to conventional search engines alone.

In the reminder of this paper, we first describe the InfoSpiders model in detail, in Section 2. Then, Section 3 presents the design and implementation of MySpiders, a multithreaded Java applet based on InfoSpiders that has been deployed on the Web. In Section 4, we introduce three novel metrics designed to enable the evaluation of intelligent online Web mining agents and their capability to improve on the performance of search engines, complementing them in a scalable way. These metrics are then used to analyze the results of crawling experiments conducted to compare the quality and recency of pages retrieved by MySpiders versus those retrieved by a major commercial search engine. Section 5 discusses some lessons learned through the deployment of MySpiders and reviews related research. We conclude in Section 6 with a look into the future.

2. The InfoSpiders model

InfoSpiders are an evolutionary multiagent system in which each agent in a population of peers adapts to its local information environment by learning to estimate the value of hyperlinks, while the population as a whole attempts to cover all promising areas through selective reproduction. Fig. 1 shows the representation of each InfoSpiders agent. The agent interacts with the information environment that consists of the actual networked collection (the Web) plus information kept on local data structures.

The adaptive representation of InfoSpiders roughly consists of a list of keywords, initialized with the query terms, and of a feed-forward neural net. The keywords represent an agent's opinion of what terms best discriminate documents relevant to the user from the rest. The neural net is used to estimate links; it has an input for each keyword and a single output unit.

The InfoSpiders algorithm is illustrated in Fig. 2. InfoSpiders rely on either traditional search engines or a set of personal bookmarks in order to obtain a set of seed URLs pointing to pages supposedly relevant to the query handed in by the user. The size of the seed set is determined by the user and in turn determines the initial population size. The starting documents are prefetched, and each agent in the population is "positioned" at one of these documents and given an initial amount of *energy*. Energy is the currency that allows agents to survive and crawl. The initial population size is another parameter set by the user, determining how many starting points to use (e.g., how many hits to obtain from a search engine).

At each step, each agent analyzes the text of the document where it is currently situated to estimate the relevance of its information neighborhood, given by the outgoing hyperlinks in the current page. In simple terms, an agent estimates each outgoing link by looking at the occurrence of query terms in the vicinity of the link. The agent then uses the link relevance estimates to choose the next document to visit. More formally, for link l and for each keyword k, the neural net receives input:

$$\mathrm{in}_{k,l} = \sum_{i:\mathrm{dist}(k_i,l)\le\rho} \frac{1}{\mathrm{dist}(k_i, l)} \qquad (1)$$

where k_i is the ith occurrence of k in document D and $\mathrm{dist}(k_i,l)$ gives more weight to keyword occurrences in

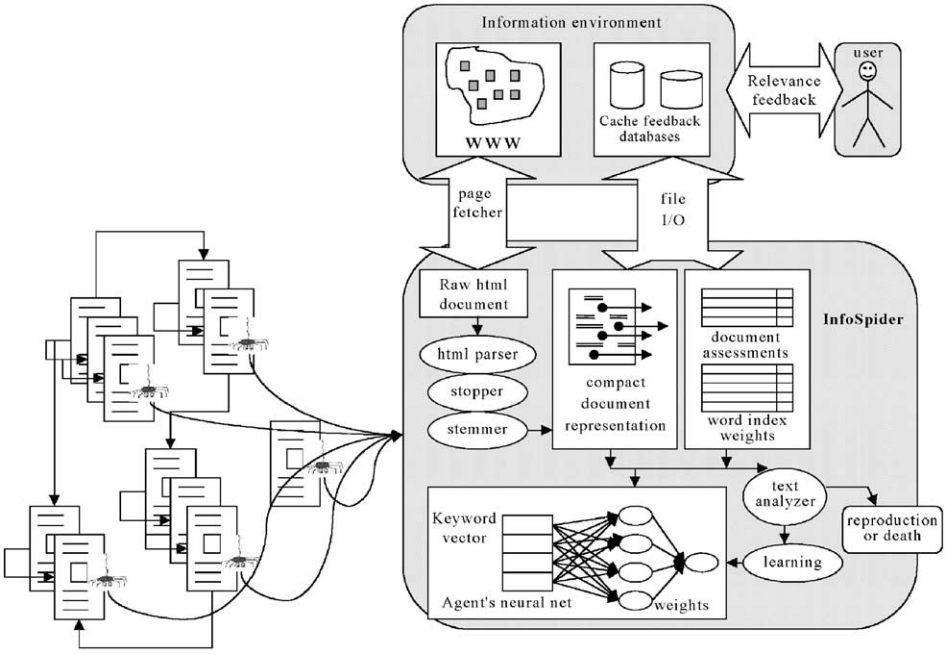

Fig. 1. InfoSpiders architecture and adaptive representation.

the vicinity of l, by counting intervening links up to a maximum window size of $\pm\rho$ links away ($\rho = 5$ in the experiments described below). The neural network then sums activity across all of its inputs; each unit j computes a logistic activation function

$$o_j = \tanh(b_j + \sum_k w_{jk} \mathrm{in}_k^l) \quad (2)$$

where b_j is its bias term, w_{jk} are its incoming weights, and in_k^l its inputs from the lower layer. The output of the network is the activation of the output unit, λ_l. The process is repeated for each link in the current document. Then, the agent uses a stochastic selector to pick a link with probability distribution:

$$\Pr[l] = \frac{e^{\beta\lambda_l}}{\sum_{l' \in D} e^{\beta\lambda_{l'}}}. \quad (3)$$

After a document has been visited, the agent needs to update its energy. The energy is used to survive and move, so the agent will be rewarded with energy based on the estimated relevance of the visited documents. Agents are also charged with energy to account for the work performed to download the page from the network. The relevance estimation function and the cost function must be specified in an implementation of the InfoSpiders algorithm (see Section 3).

An agent can modify its neural net-based behavior during its life by comparing the relevance of the current document (as evaluated once the document is visited) with the estimation that was made from the previous page, prior to following the link that led to the current one. Reinforcement learning is employed to achieve this goal, namely a connectionist version of Q-learning [19]. The neural net is trained online to predict values of links based on local context. The value returned by the estimate( ) function is used as an internally generated reinforcement signal to compute a teaching error. The neural net's weights are then updated by back-propagation of error [35]. Learned changes to the weights are inherited by offspring at reproduction. This learning scheme is completely unsupervised.

InfoSpiders adapt not only by learning neural net weights, but also by evolving both neural net and keyword representation. If an agent accumulates enough energy, it clones a new agent in the location where it is currently situated. At reproduction, the

```
InfoSpiders(query, INIT_POP) {
   starting_urls := search_engine(query, INIT_POP);
   for agent (1..INIT_POP) {
      initialize(agent, query);
      situate(agent, starting_urls);
      agent.energy := THETA / 2;
   }
   until extinction {
      foreach agent {
         pick_out_link_from_current_document(agent);
         agent.doc := fetch_new_document(agent);
         agent.energy += estimate(agent.doc) - cost(agent.doc);
         Q_learning(agent, estimate(agent.doc));
         if (agent.energy >= THETA) {
            offspring := mutate(clone(agent));
            offspring.energy := agent.energy / 2;
            agent.energy -= offspring.energy;
         }
         elseif (agent.energy <= 0) kill (agent);
      }
   }
}
```

Fig. 2. InfoSpiders algorithm.

offspring's neural net is mutated by adding random noise to a fraction of the weights. The keyword vector is mutated by adding the most frequent term from the current document. The idea is that since the current page led to reproduction, it probably contains features that are correlated with the user's interests and therefore we want to expand the agent's representation to capture such features. The neural net weights associated with keyword mutations are initialized randomly, and reinforcement learning is in charge of adjusting them in the early stages of the new agent's life. The parent and offspring agents can continue the search independently of each other.

If an agent runs out of energy, it is destroyed. Such reproduction and death mechanisms with their fixed thresholds make selection a *local* decision, independent of other agents. This way, agents are dispatched to the most promising areas of the information space [24,28]. Another important consequence of the local selection mechanism is that agents have minimal mutual interactions, making it possible for them to execute concurrently. In the following section, we focus on one such concurrent implementation of InfoSpiders. Further details on some aspect of the algorithm, as well as extensions not discussed in this paper, such as relevance feedback, can be found elsewhere [25,26].

3. Implementation of MySpiders

Due to the parallel nature of the InfoSpiders algorithm, multithreading is expected to provide better utilization of resources compared to a single thread (sequential) implementation such as described in the previous section. Since Java has built-in support for threads, we decided to implement a multithreaded version of InfoSpiders as a Java applet. The multithreaded implementation allows one agent to use the network connection to retrieve documents, while another agent can use the CPU, and another can access cache information on the local disk.

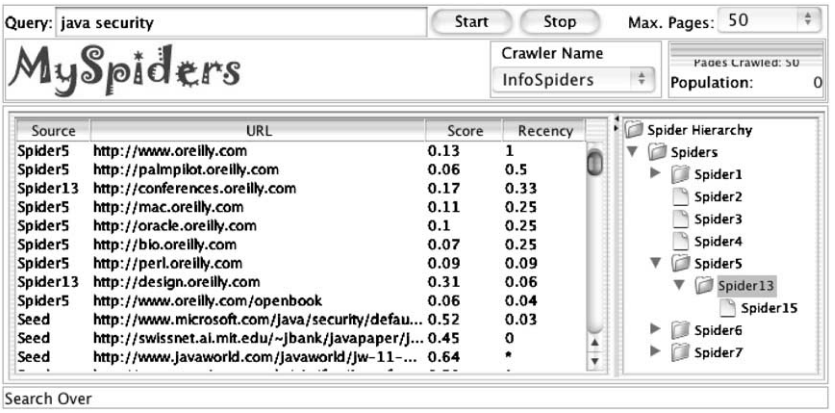

Fig. 3. Screen shot of MySpiders, a Web-based multithreaded Java applet implementation of the InfoSpiders multiagent system.

Another benefit of a Java applet implementation is that classes can be loaded at runtime over the Web, thereby allowing for a Web-based deployment of the client-based application. This approach prevents the scalability problems of server-based deployment and sidesteps the logistic difficulties of porting, distributing, maintaining and updating a binary application.

The multithreaded applet implementation of InfoSpiders, called MySpiders, is available on a public Web server[1]. Fig. 3 shows the user interface of the MySpiders applet in the course of a query search. The applet hides many parameters from the users in order to keep the interface as simple as possible. Such parameters are set to default values (for example, INIT_POP is set to 10). The user specifies a query and a maximum number of pages to be crawled by MySpiders, which is an indication of how long the user is willing to wait.

Once the start button is pressed, the system obtains INIT_POP seed pages from a trusted search engine (we currently use http://google.yahoo.com for this purpose). Then, an automated crawl process is initiated. The user can scan through a ranked list of suggested pages found up to that time. A crawled page is displayed to the user if its similarity to the query is above a threshold (cf. Eq. (4) and GAMMA in Fig. 4). Clicking on any suggested URL during the crawl brings up the page in the default browser. The suggested URLs are preceded by a number that represents the agent (or spider) that found the page. If the user likes a page, clicking on the spider that found it provides a list of links to other pages found by that spider. The user has the choice to stop the search process at any time if she has already found the desired results, or if the process is taking too long. Otherwise the crawl stops when the population visits MAX_PAGES documents or goes extinct for lack of relevant information resources.

The multithreaded version of the algorithm behind MySpiders is presented in Fig. 4. As illustrated by the pseudocode, there are a few design decisions made to get from the generic algorithm of Fig. 2 to this implementation, in addition to the user interface issues described above.

- Security: The user has to grant the applet permissions to access the local disk (to read and write cache files) and the network (to open and close HTTP connections to Web servers).
- Cache: A lock mechanism allows concurrent access to a shared cache; the cache optimizes network usage and also forces agents to explore new pages by preventing energy gains for previously visited pages. A small cost is charged for accessing the cache to prevent cyclic behaviors by the agents. To maintain the integrity of the cache, only a single agent can perform read/write operations at one time. However, since network communication is the most expensive resource for the algorithm, the delay imposed by caching semaphore operations is insignificant when

[1] http://myspiders.biz.uiowa.edu.

```
MySpiders {
    input_from_user(disk_permissions, network_permissions);
    input_from_user(query, MAX_PAGES);
    starting_urls := search_engine(query);
    foreach agent {
        initialize(agent, query);
        situate(agent, starting_urls);
        agent.energy := THETA / 2;
    }
    initialize(Cache, Cache_semaphore);
    foreach agent {
        while visited < MAX_PAGES and not (user_stop or extinction) {
            pick_out_link_from_current_document(agent);
            agent.doc := fetch_new_document(agent);
            lock(Cache_semaphore);
            update(Cache);
            unlock(Cache_semaphore);
            agent.energy += sim(query, agent.doc) - latency(agent.doc);
            display_to_user(agent.doc) if sim(query, agent.doc) > GAMMA;
            Q_learning(agent, estimate(agent.doc));
            if (agent.energy >= THETA) {
              offspring := mutate(clone(agent));
              offspring.energy := agent.energy / 2;
              agent.energy -= offspring.energy;
            }
            elseif (agent.energy <= 0) kill(agent);
        }
    }
}
```

Fig. 4. Pseudocode of multithreaded MySpiders applet.

compared with the speedup acquired by avoiding the transfer of duplicate documents.

• Estimate function: *Cosine similarity* is used as an estimate( ) function to determine the amount of energy an agent receives the first time it visits a page:

$$\mathrm{sim}(q,p) = \frac{\sum_{k \in q \cap p} f_{kq} f_{kp}}{\sqrt{\left(\sum_{k \in p} f_{kp}^2\right)\left(\sum_{k \in q} f_{kq}^2\right)}} \quad (4)$$

where q is the query, p is the page, and f_{kd} is the frequency of term k in d. This choice is in line with the standard use of similarity to estimate relevance in information retrieval (and in most search engines). A global "inverse document frequency" factor is not used in this weighting scheme because we assume only local knowledge about the Web collection being crawled.

• Cost function: HTTP response *latency* is used as a cost( ) function to determine the amount of energy an agent spends the first time it visits a page:

$$\mathrm{latency}(p) = \frac{\mathtt{INIT\_POP} \cdot \mathtt{THETA}}{\mathtt{MAX\_PAGES}} \cdot \frac{\mathrm{time}(p)}{\mathtt{TIMEOUT}} \quad (5)$$

where time(p) is the time taken for page p to be downloaded, THETA is the reproduction threshold (cf. Fig. 4), and TIMEOUT is the timeout parameter for the socket connection. The purpose of the first term is to allow the population to visit on average MAX_PAGES irrelevant documents before running out of its initial energy. This choice of a latency-based cost function is based on the idea that agents should be charged more for visiting large pages and/or slow

servers. Empirical observations justify this cost model on both efficiency and quality bases [10]. Another benefit of this cost function is that the applet response time remains reasonable even for users with a slow connection to the Internet (e.g. over a dial-up modem line).

4. Evaluation

4.1. Performance metrics

Evaluating an online, dynamic query-driven Web mining system such as MySpiders presents serious difficulties. First, relevant sets corresponding to queries are available on certain collections, making it possible to use standard information retrieval performance measures such as precision and recall [22]. For example, in previous work [25], an earlier version of the InfoSpiders system was evaluated on a limited and controlled chunk of the Web—a subset of the Encyclopaedia Britannica—for which a large number of test queries were readily available, along with the corresponding relevant sets. The collective performance of InfoSpiders was assessed and compared to *best-first-search*, a baseline heuristic in which the links to be followed are ranked according to the similarity between the query and the pages containing the links. We found that for queries whose relevant sets were not too far from the InfoSpiders' starting pages, InfoSpiders had a significant advantage over best-first-search. Conversely, InfoSpiders did worse when the relevant documents were farther away. These results suggested using search engines to provide InfoSpiders with good starting points. To test this hypothesis, we ran InfoSpiders as a front-end to a traditional search engine for an ad hoc query that the search engine could not satisfy alone [26]. InfoSpiders did locate all of the relevant pages quickly in this special case.

In spite of these results, given our goal to overcome the recency/coverage limitations of search engines, it is clear that a satisfactory evaluation of our system can only be achieved by testing on actual, real-time Web data rather than on limited or artificial collections. However, the lack of relevant set information on the real Web makes the use of standard performance metrics impossible.

The second major difficulty is that comparing the added value of online, query-driven Web mining with respect to using a search engine alone is like comparing apples and oranges. Clearly performing some additional search can only yield an improvement, so an evaluation based on recall-like statistics alone would be unfairly biased in favor of query time crawling. On the other hand, the cost of query time crawling cannot be amortized over many queries, so that an evaluation based on efficiency alone would be unfairly biased in favor of the search engine approach of separating crawling from querying, discounting recency and coverage effects.

To address such difficulties, we propose to use some novel metrics designed to allow for a fair comparison between the two approaches, and for a quantitative evaluation of the value added by query-driven crawling. The idea is to design two metrics that approximate recall and precision, respectively, when the relevant set is unknown but some subset is known, that is, some description of relevant pages is available. This is quite a realistic situation for the Web. Further, we will define a third metric to capture the recency of a retrieved set.

Let us assume that we have a query q and some description r_q of a subset of the relevant pages. r_q could be the text of one or a few relevant pages, or an abstract of a relevant paper, or a summary of related resources prepared by a hub or portal site. Now consider a user who has some good starting points, such as the top hits returned by a trusted Web directory or search engine, and the time to browse through some additional pages. Such a user might look at further hits returned by a search engine, or alternatively launch a query driven crawler like MySpiders.

Let us call C_q the set of additional pages obtained either from the engine or from the crawl. We can then define an *estimated precision* of the crawl set

$$P(C_q) \equiv \frac{1}{|C_q|} \sum_{p \in C_q} \mathrm{sim}(r_q, p) \tag{6}$$

where sim( ) is the cosine similarity function defined in Eq. (4). This metric approximates precision to the extent that r_q is a faithful approximation of the (unknown) relevant set.

Table 1
Sample queries and relevant set descriptions (prior to removing stop words and stemming)

Query	Description
EDUCATION STATISTICS	Education Census—from the US Census Bureau. National Assessment of Educational Progress—data and reports from the National Center for Education Statistics. National Education Statistical Information Systems (NESIS)—joint programme of UNESCO/ADEA to develop self-sustainable statistical information systems for education policy needs in Africa. School District Data Book Profiles: 1989–1990—social, financial and administrative data for school districts in the United States. School Enrollment—data from the US Census Bureau.
ARTS ART HISTORY CRITICISM AND THEORY	Art Historians' Guide to the Movies—a record of appearances of and references to famous works of painting, sculpture, and architecture in the movies. Art History: A Preliminary Handbook—guide to studying art history. Artists on Art—excerpts from writings and interviews of great artists past and present on the concept and process of art, as well as artist chronologies of the periods in which they worked. Brian Yoder's Art Gallery and Critic's Corner Part—magazine of art and theory produced by CUNY graduate students. Pre-Raphaelite Criticism Underground Art Critic—offering modern art criticism for the postmodern masses.

We can further define an *estimated recall* of the crawl set

$$R(C_q) \equiv P(C_q) \cdot |C_q| = \sum_{p \epsilon C_q} \mathrm{sim}(r_q, p) \tag{7}$$

that intuitively approximates actual recall, which is obtained by multiplying precision by the size of the retrieved set (modulo the size of the relevant set, an unknown constant in our case).

Finally, let us define the *estimated recency* of the crawl set

$$T(C_q) \equiv \frac{1}{|C_q|} \sum_{p \epsilon C_q} \frac{1}{1 + \delta_t(p)} \tag{8}$$

where

$$\delta_t(p) = \begin{cases} t_m(p) - t_i(p) & \text{if } t_m(p) > t_i(p) \\ 0 & \text{otherwise.} \end{cases} \tag{9}$$

Here $t_m(p)$ is the date when p was last modified by the author and $t_i(p)$ is the date when p was indexed either by the search engine or by the query driven crawler. Clearly it would be desirable for a decision support system to be based on an up-to-date image of the mined documents, as reflected by a high value of the estimated recency metric.

4.2. Experimental results

The three metrics defined above can now be used to gauge the value of MySpiders against a search engine as a Web mining decision support tool. We report on experiments using AltaVista[2] as a representative search engine both because it is one of the major commercial portals and because it is one of the very few disclosing information on the indexing date t_i, which allows us to evaluate recency.

Without loss of generality, we assume a query to be a topic found in the Yahoo directory[3]. To this end, we conducted a preliminary crawl of over 99,700 Yahoo pages and out of these we identified a set of 100 "leaf" topics such that each had at least 10 links to external pages and no subtopics. We formed 100 queries ($q = 0 \ldots 99$) using these Yahoo topics, and then we obtained the corresponding r_q for each of these queries from the textual description of relevant pages written by the Yahoo editors. We assume that the pages classified in the Yahoo directory under each topic are a subset of the relevant set for the corresponding query. Table 1 shows a couple of sample queries and descriptions.

For each query q, we obtained the 100 top-ranked pages from AltaVista and allowed MySpiders to crawl $|C_q| = 100$ pages, for a total of 200 pages/query × 100 queries = 20,000 pages crawled (display threshold GAMMA = 0). Yahoo pages were removed from both

[2] http://www.altavista.com.
[3] http://www.yahoo.com.

sets for fairness. Fig. 5 plots the mean estimated precision

$$\bar{P} = \frac{1}{100}\sum_{q=0}^{99} P(C_q) \tag{10}$$

versus the number of pages returned by AltaVista and crawled by MySpiders. Error bars correspond to standard errors, i.e., $\pm 1\ \sigma_{\bar{P}}$. MySpiders significantly outperform AltaVista in the first phase of the crawl, with peak precision after about five pages. Around the middle of the crawl, AltaVista catches up and the difference becomes insignificant. After about 80 pages, AltaVista's precision declines and MySpiders again have a significant advantage.

Fig. 6 plots the mean estimated recall

$$\bar{R} = \frac{1}{100}\sum_{q=0}^{99} R(C_q) \tag{11}$$

versus the number of pages returned by AltaVista and crawled by MySpiders. Error bars again correspond to standard errors, $\sigma_{\bar{R}}$. MySpiders display a slight recall advantage over AltaVista at the beginning of the crawl, and a larger advantage at the end of the crawl when AltaVista basically stops retrieving relevant pages while MySpiders continues to do well.

Fig. 7 plots the mean estimated recency

$$\bar{T} = \frac{1}{100}\sum_{q=0}^{99} T(C_q) \tag{12}$$

versus the number of pages returned by AltaVista and crawled by MySpiders. Error bars again correspond to standard errors, $\sigma_{\bar{T}}$. MySpiders display a strong recency advantage of more than one and a half orders of magnitude over AltaVista throughout the crawl. $\bar{T} = 1$ for MySpiders by design, since the crawl occurs at query time and therefore the system automatically filters out stale links and relies on the current version of any page.

Fig. 8 shows a precision-recall plot based on the mean estimated metrics $\bar{P}$ and $\bar{R}$ for AltaVista and MySpiders. Error bars along the y-axis correspond to precision standard errors, $\sigma_{\bar{P}}$, and error bars along the x-axis correspond to recall standard errors, $\sigma_{\bar{R}}$. MySpiders display a statistically significant advantage over AltaVista in the early phase of the crawl

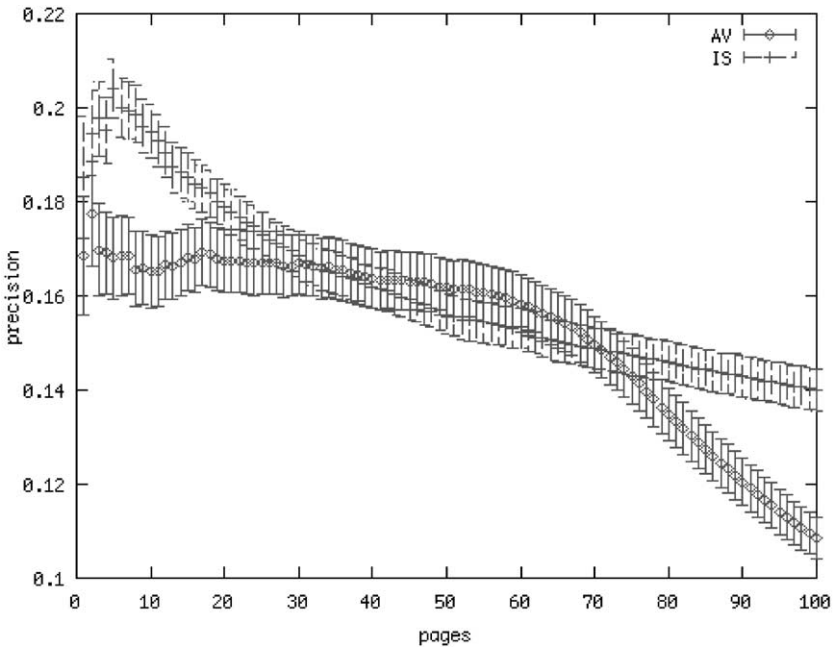

Fig. 5. Estimated precision as a function of pages retrieved by AltaVista (AV) and crawled by MySpiders (IS).

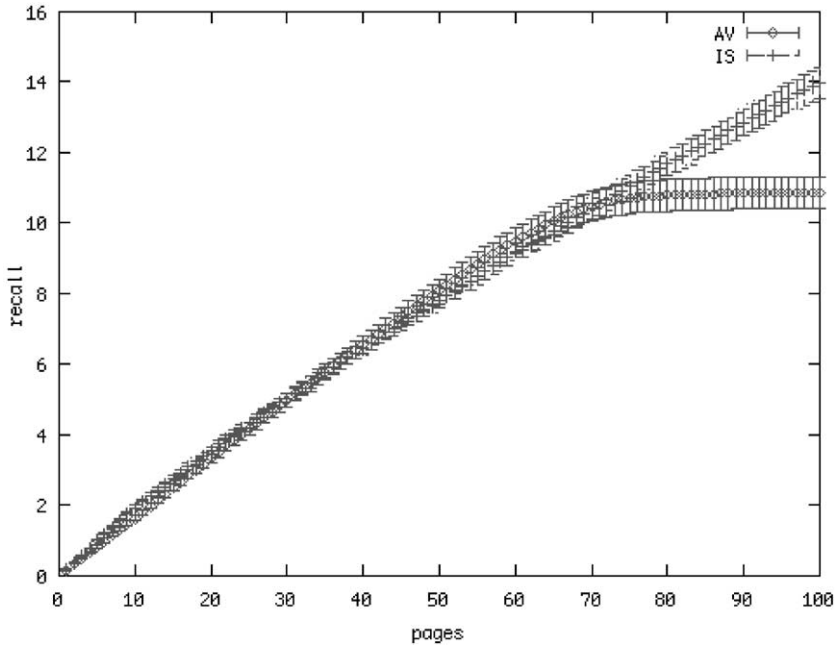

Fig. 6. Estimated recall as a function of pages retrieved by AltaVista (AV) and crawled by MySpiders (IS).

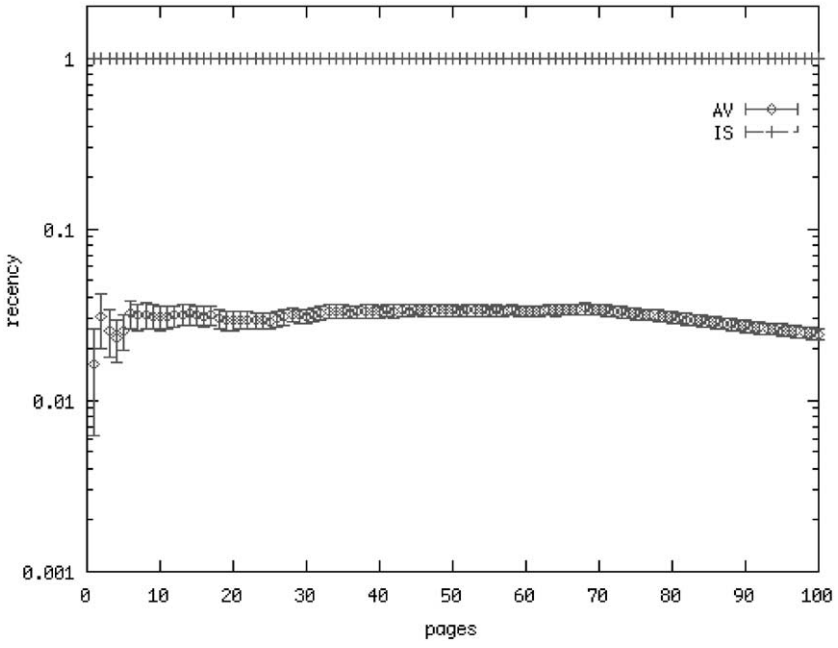

Fig. 7. Estimated recency as a function of pages retrieved by AltaVista (AV) and crawled by MySpiders (IS).

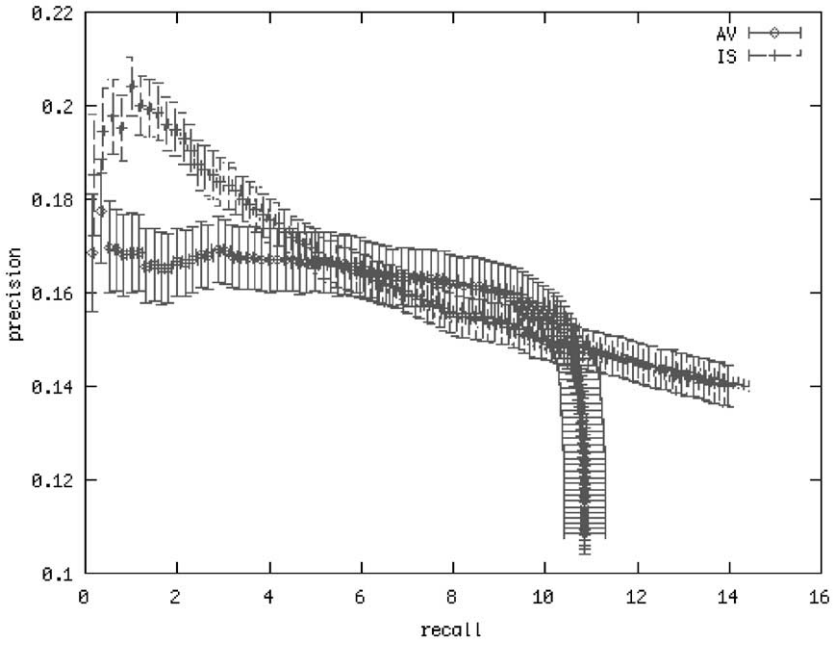

Fig. 8. Estimated precision versus estimated recall for AltaVista (AV) and MySpiders (IS).

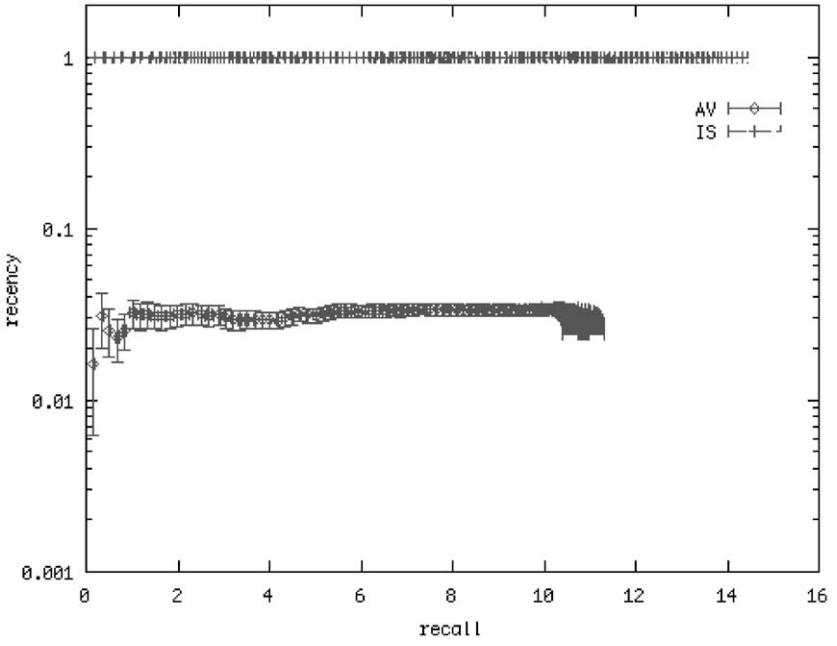

Fig. 9. Estimated recency versus estimated recall for AltaVista (AV) and MySpiders (IS).

(when MySpiders find relevant pages at a high rate) and in the final phase (when the search engine fails to locate any additional relevant documents), while AltaVista has a nonsignificant advantage in the middle phase.

Fig. 9 shows a recency-recall plot based on the mean estimated metrics $\bar{T}$ and $\bar{R}$ for AltaVista and MySpiders. Error bars along the y-axis correspond to recency standard errors, $\sigma_{\bar{T}}$ and error bars along the x-axis correspond to recall standard errors, $\sigma_{\bar{R}}$. MySpiders outperform AltaVista by a strong margin of over one and a half orders of magnitude. More interestingly, the mean recency of the AltaVista pages suffers a clear loss in correspondence to the final recall increase, while the recency of the MySpiders pages is obviously unaffected. This trade-off between recall and recency is a manifestation of the scalability issue discussed in Section 1.

Another illustration of the scalability effect is offered by Fig. 10, in which the product of the mean estimated recency and recall, $\bar{T}\bar{R}$, is plotted versus the number of pages returned by AltaVista and crawled by MySpiders. Error bars for the product are estimated by $\bar{T}\sigma_{\bar{R}}+\bar{R}\sigma_{\bar{T}}$. Not only there is a large margin of advantage for MySpiders, but the margin grows wider as the product keeps growing at a healthy rate for MySpiders while it starts declining after about 70 pages for AltaVista.

5. Discussion

5.1. Lessons learned

In the course of designing and running the experiments described in the previous sections, we have learned several lessons regarding the evaluation of dynamic search tools, such as MySpiders. First, comparing these systems with traditional search engines is difficult because the design goals of the two approaches are very different. The idea of query-driven crawling is to complement and augment search engines, not replace them. But how to demonstrate the value added by online browsing agents without full access to a search engine database? This problem is compounded by the fact that commercial search engines do not publish the inner workings of their crawling and ranking algorithms.

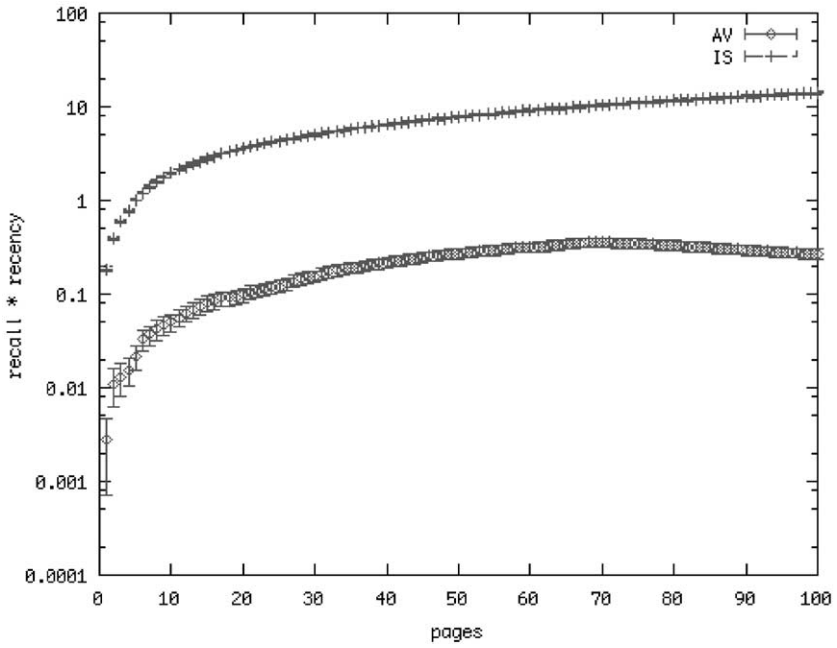

Fig. 10. Product of estimated recency and recall as a function of pages retrieved by AltaVista (AV) and crawled by MySpiders (IS).

Another difficulty is introduced by the lack of an easy way to decide how long to run MySpiders. However, the experiments described in this paper demonstrate that even crawling as few as five pages at query time can lead to substantial performance improvements (cf. Fig. 5).

The above observation also addresses the question of the apparent inefficiency of query driven crawling due to its lack of amortization over queries. The extra bandwidth used at query time is negligible compared to that used blindly by search engine crawlers, and can easily be made up for by allowing search engine indexes to become only slightly more out of date—something of little consequence if offset by crawling at query time.

Evaluation of MySpiders is complicated by their reliance on search engines to provide starting points. We could alternatively get the starting points from a meta-search engine, but then it would be even harder to attribute differences in effectiveness to the crawling algorithm versus the starting points.

A serious problem in testing any system on actual Web data is that the relevant sets are unknown. Consequently, standard evaluation tools such as precision-recall plots have to rely on approximate relevant sets, as we have proposed in this paper. Such sets are likely incomplete, biased, and small—all characteristics that hinder the significance of our analysis. The Web Track of the Text Retrieval Conference is attempting to construct a Web data set with queries and relevant sets, but so far it has not been possible to construct a data set of fully connected pages, allowing for browsing agents to navigate across it.

The software engineering aspects of developing and deploying the MySpiders prototype applet have taught us other lessons. First, Java is a good choice of language for this type of application. Its object-oriented nature is a good match for agent-based computation in general. More importantly, Java allowed for an efficient implementation of InfoSpiders, thanks to its threading support. Development was also simplified because of Java's networking support.

Unfortunately, although Java applet portability would appear to be a big advantage on the surface, it turns out that security issues—the need to provide a mechanism for granting privileges to the applet—leave such portability goals largely unrealized. In fact, to be able to open network connections to hosts other than the one from where the applet itself was downloaded, and to access the local disk for cache I/O operations, the applet has to bypass the browser's security manager. To accomplish this task, we used a combination of digitally signed applets and policy files. Both these solutions have platform dependencies that hinder portability and ease of use. Another issue weakening the Java choice is the low speed of execution of Java byte code.

5.2. Related research

Autonomous agents, or semi-intelligent programs making automatic decisions on behalf of the user, have been viewed for some time as a way of decreasing the amount of human–computer interaction necessary to manage the increasing amount of information available online [20]. A number of intelligent agents have been developed and deployed in recent years to help users find information on the Web.

Most existing information search agents suffer from a common limitation: their dependence on search engines. Typical examples of "parasite" agents that rely on centralized repositories to find information on behalf of the users are homepage and paper finders [31,38]. While such agents can add to a search engine very useful heuristics, they cannot overcome the limited coverage and recency of the engines they exploit.

While an agent whose search process consists of submitting queries to a search engine cannot find a document that had not already been located and indexed by the engine, multiple search engines can be combined in order to improve on the recall ratio of any single engine. Such meta-search agents, including popular ones like MetaCrawler[4] and Sherlock[5], have proven successful in gathering and combining the relevant sets from many different search engines [16,37,39].

Recently, linkage information has been used in algorithms designed to identify hub and authority pages, such as HITS/Clever, and to determine the reputation of pages, as in TOPIC [5,14,30]. Although such techniques have demonstrated to be very effec-

[4] http://www.metacrawler.com.
[5] http://www.apple.com/sherlock.

tive, they too are bound by the limitations of the search engines upon which they rely to get the pages they evaluate. In contrast, in our proposed system, linkage information is used to locate recent pages that are not known to search engines.

A different class of agents have been designed to learn a user's interest profile in order to recommend pages [2,7,18,33]. These agents learn to predict an objective function online and can track time-varying user preferences. However, they need supervision from the user in order to work, either through direct feedback or by monitoring the user's browsing activity; no truly autonomous search is possible.

Other systems, based on multiagent paradigms, adapt a matching between a set of discovery agents (typically search engine parasites) and a set of user profiles (corresponding to single- or multiple-user interests) [3,32]. These systems can learn to divide the problem into simpler subproblems, dealing with the heterogeneous and dynamic profiles associated with long-standing queries. However, they share the weak points of other agents who perform no active autonomous search, and therefore cannot improve on the limitations of the search engines they exploit.

One last set of agent-based systems actually relies on agents searching (browsing) online on behalf of the user. The first of such systems, Fish Search [9], was inspired by ecological and artificial life models. Fish Search was hindered in effectiveness by the absence of any adaptability in the agents. One unfortunate consequence of its fixed search strategy was the possibility of load-unfriendly search behaviors, partially mitigated by the use of a cache. This factor, coupled with the growing popularity of search engines and the relatively slow performance of Fish Search, did not help to focus on the potential advantages of such models. Recently, the area of query-driven crawling (also referred to as *focused crawling*) has gained new popularity [4,6,8], in part due to the emergence of topic-specific portals.

A number of artificial intelligence techniques are embedded in the InfoSpiders model and prototype. The main one is the evolutionary algorithm outlined in Fig. 2. This algorithm employs a novel *local selection* scheme that has been shown to be particularly suitable for *cover* optimization—we are not trying to find the best possible page, but rather as many of the relevant pages as possible [27,28]. The algorithm biases the search process to focus the search on promising regions, where agents flourish.

A consequence of the evolutionary algorithm is the adaptation of keyword representations via mutations. This mechanism implements a form of *selective query expansion*. Based on local context, the query can adapt over time and across different locations in a completely unsupervised fashion. In contrast, traditional information retrieval notions of query expansion are dependent upon the availability of relevance feedback from the user [36]. The population of agents thus embodies a distributed, heterogeneous model of relevance that may comprise many different and possibly inconsistent features. However, each agent focuses on a small set of features, maintaining a well-defined model that remains manageable in the face of the huge feature dimensionality of the search space.

The central adaptive representation employed by InfoSpiders is the neural net that allows each agent to assess the relevance of outgoing links. This task is crucial because an agent pays a cost in network load to visit a page, so it must have some confidence that the energy is well spent. A neural net can represent complex relationships between query terms, including negative weights and nonlinear interactions [35]. Such a level of complexity seems necessary if we expect that agents learn to mimic the browsing skills of human users.

Learning is also central for InfoSpiders because an agent must be able to adapt during its life, predicting document relevance over small time and space scales. Examples are not available except for those based on the agent's own experience, so that reinforcement learning is called for. Q-learning [40] is particularly appropriate because the environment provides agents with energy cues that can be maximized as future discounted rewards. The use of Q-learning to guide the browsing activity has also been studied independently of our evolutionary framework [34].

Finally, InfoSpiders employ several standard information retrieval techniques, such as the removal of noise words [11], the stemming of terms with common roots [12], and the use of similarity-based relevance measures.

All these methods allow InfoSpiders to adapt to their local environmental context over time and across different parts of the Web. It is our conviction

that the Web is ripe for the application of such artificial intelligence techniques allowing for the construction of more and more intelligent software agents, capable not only of searching information on behalf of the user but also of performing further actions, e.g., buying and selling information from other agents.

6. Conclusion

This paper presented some of the shortcomings of traditional information retrieval systems on the Web. Despite the fast technological developments of computational tools, the rapidly growing and dynamic nature of the Web is making static search engines too incomplete and out of date for Web intelligence applications. The novel contributions of this paper are summarized as follows:

- We identified and discussed the scalability limitations of the traditional search engine approach of disjoint crawling and querying, diminishing the usability of current search engines as an effective technology for decision support and competitive intelligence tools.
- We suggested complementing search engines with query-driven online Web mining, to build Web intelligence tools that scale better with the dynamic nature of the Web, allowing for the location of pages that are both relevant and recent.
- We presented MySpiders, a public Web mining tool deployed as a threaded Java applet. This tool implements the InfoSpiders algorithm, a query-driven algorithm based on an adaptive population of online agents with an intelligent text mining behavior emulating the browsing performed by human users.
- We introduced three measures designed to evaluate the value added of query-driven crawling with respect to the use of search engines alone; these metrics approximate precision and recall in the absence of complete knowledge about relevant sets, and further estimate the recency of retrieved pages.
- We outlined the results of experiments comparing the performance of MySpiders with that of a major commercial search engine in a Web intelligence task; we showed that query-driven crawling is a more scalable approach and leads to mining pages that are significantly more relevant and more recent.

The evaluation measures defined in this paper can also be used to compare different crawling strategies. This would actually be a more direct comparison than the analysis carried out in this paper, since crawler algorithms are designed with the same goal. Experiments with the estimated precision metric show that a crawler based on InfoSpiders significantly outperforms one based on PageRank, the algorithm employed by Google for ranking [29]. In this type of analysis, the metrics can be augmented with factors accounting for the different memory and CPU complexity of the various crawler algorithms.

The MySpiders Web site is under further development to enable the execution of the applet code on those platforms that are yet unsupported at the time of this writing. The Java Network Launching Protocol (JNLP) is being assessed as a means to eliminate the dependencies between operating systems and browsers currently imposed by the use of a security certificate.

Future improvements of MySpiders include the implementation and testing of a relevance feedback module, whereby users could asynchronously assess relevant pages and agents could subsequently focus their search on documents nearby relevant ones. We will also experiment with alternative weighting schemes to estimate a document's relevance, mainly a localized version of the TF-IDF model measuring inverse document frequency over an agent's history, and with variations of the reinforcement learning strategy (e.g., tuning learning rates and discount factors).

Finally, in competitive intelligence applications where recency is crucial, it can be made into an explicit fitness objective of the InfoSpiders algorithm by allowing the energy of a document to depend not only on its estimated relevance but also on its age.

Acknowledgements

The author is grateful to Melania Degeratu for contributions to earlier drafts of this paper and to Padmini Srinivasan for helpful discussions. Gautam Pant is responsible for the ongoing development of MySpiders.

References

[1] R. Albert, H. Jeong, A.-L. Barabasi, Diameter of the world wide web, Nature 401 (6749) (1999) 130–131.
[2] R. Armstrong, D. Freitag, T. Joachims, T. Mitchell, WebWatcher: a learning apprentice for the world wide web, AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, 1995.
[3] M. Balabanović, An adaptive web page recommendation service, Proc. 1st International Conference on Autonomous Agents, ACM Press, New York, NY, 1997, pp. 378–385.
[4] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, V. Soroka, S. Ur, Adding support for dynamic and focused search with Fetuccino, Computer Networks 31 (11–16) (1999) 1653–1665.
[5] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, J. Kleinberg, Automatic resource compilation by analyzing hyperlink structure and associated text, Computer Networks 30 (1–7) (1998) 65–74.
[6] S. Chakrabarti, M. van den Berg, B. Dom, Focused crawling: a new approach to topic-specific web resource discovery, Computer Networks 31 (11–16) (1999) 1623–1640.
[7] L. Chen, K. Sycara, WebMate: a personal agent for browsing and searching, Proc. 2nd International Conference on Autonomous Agents, ACM Press, New York, NY, 1998, pp. 132–139.
[8] J. Cho, H. Garcia-Molina, L. Page, Efficient crawling through url ordering, Proc. 7th Intl. World Wide Web Conference, Brisbane, Australia, Computer Networks 30 (1–7) (1998) 161–172.
[9] P. De Bra, R. Post, Information retrieval in the world wide web: making client-based searching feasible, Proc. 1st Intl. World Wide Web Conference, Geneva, Computer Networks and ISDN Systems 27 (2) (1994) 183–192.
[10] M. Degeratu, G. Pant, F. Menczer, Latency-dependent fitness in evolutionary multithreaded web agents, Proc. GECCO Workshop on Evolutionary Computation and Multi-Agent Systems (ECOMAS'01), San Francisco, CA, July 2001, pp. 313–316.
[11] C. Fox, Lexical analysis and stop lists, Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Upper Saddle River, NJ, 1992.
[12] W. Frakes, Stemming algorithms, Information Retrieval: Data Structures and Algorithms, Prentice-Hall, Upper Saddle River, NJ, 1992.
[13] M. Henzinger, Link analysis in web information retrieval, IEEE Data Engineering Bulletin 23 (3) (2000) 3–8.
[14] J. Kleinberg, Authoritative sources in a hyperlinked environment, Journal of the ACM 46 (5) (1999) 604–632.
[15] S. Lawrence, C. Giles, Searching the world wide web, Science 280 (1998) 98–100.
[16] S. Lawrence, C. Giles, Context and page analysis for improved web search, IEEE Internet Computing 2 (4) (1998) 38–46.
[17] S. Lawrence, C. Giles, Accessibility of information on the web, Nature 400 (1999) 107–109.
[18] H. Lieberman, Autonomous interface agents, Proc. ACM Conference on Computers and Human Interface. Atlanta, GA, ACM/Addison-Wesley, Boston, MA, 1997, pp. 67–74.
[19] L.-J. Lin, Self-improving reactive agents based on reinforcement learning, planning, and teaching, Machine Learning 8 (1992) 293–321.
[20] P. Maes, Agents that reduce work and information overload, Communications of the ACM 37 (7) (1994) 31–40.
[21] F. Menczer, ARACHNID: adaptive retrieval agents choosing heuristic neighborhoods for information discovery, Proc. 14th International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1997, pp. 227–235.
[22] F. Menczer, Life-like agents: Internalizing local cues for reinforcement learning and evolution, PhD thesis, University of California, San Diego Department of Computer Sciences and Engineering, 1998.
[23] F. Menczer, R. Belew, Adaptive information agents in distributed textual environments, Proc. 2nd International Conference on Autonomous Agents, Minneapolis, MN, ACM Press, New York, NY, 1998, pp. 157–164.
[24] F. Menczer, R. Belew, Local Selection, Evolutionary Programming VII, Lecture Notes in Computer Science, vol. 1447 Springer, Berlin, 1998, pp. 703–712.
[25] F. Menczer, R. Belew, Adaptive retrieval agents: internalizing local context and scaling up to the web, Machine Learning 39 (2–3) (2000) 203–242.
[26] F. Menczer, A. Monge, Scalable web search by adaptive online agents: an InfoSpiders case study, in: M. Klusch (Ed.), Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet, Springer, Berlin, 1999, pp. 323–347.
[27] F. Menczer, M. Degeratu, W. Street, Efficient and scalable pareto optimization by evolutionary local selection algorithms, Evolutionary Computation 8 (2) (2000) 223–247.
[28] F. Menczer, W. Street, M. Degeratu, Evolving heterogeneous neural agents by local selection, in: M. Patel, V. Honavar, K. Balakrishnan (Eds.), Advances in the Evolutionary Synthesis of Intelligent Agents, MIT Press, Cambridge, MA, 2001, pp. 337–365.
[29] F. Menczer, G. Pant, M. Ruiz, P. Srinivasan, Evaluating topic-driven web crawlers, Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval, ACM Press, New York, NY, 2001, pp. 241–249.
[30] A. Mendelzon, D. Rafiei, What do the neighbours think? Computing web page reputations, IEEE Data Engineering Bulletin 23 (3) (2000) 9–16.
[31] A. Monge, C. Elkan, The WEBFIND tool for finding scientific papers over the worldwide web, Proceedings of the 3rd International Congress on Computer Science Research, 1996.
[32] A. Moukas, G. Zacharia, Evolving a multi-agent information filtering solution in amalthaea, Proc. 1st International Conference on Autonomous Agents, ACM Press, New York, NY, 1997, pp. 394–403.
[33] M. Pazzani, J. Muramatsu, D. Billsus, Syskill and Webert identifying interesting web sites, Proc. National Conference on Artificial Intelligence AAAI96, AAAI Press, Menlo Park, CA, 1996, pp. 54–61.
[34] J. Rennie, A.K. McCallum, Using reinforcement learning to spider the web efficiently, Proc. 16th International Conf. on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1999, pp. 335–343.

[35] D. Rumelhart, G. Hinton, R. Williams, Learning internal representations by error propagation, in: D. Rumelhart, J. McClelland (Eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition, vol. 1, Bradford Books (MIT Press), Cambridge, MA, 1986, Chap. 8.

[36] G. Salton, C. Buckley, Improving retrieval performance by relevance feedback, Journal of the American Society for Information Science 41 (1990) 288–297.

[37] E. Selberg, O. Etzioni, The metacrawler architecture for resource aggregation on the web, IEEE Expert 12 (1) (1997) 8–14 http://www.metacrawler.com.

[38] J. Shakes, M. Langheinrich, O. Etzioni, Dynamic reference sifting: a case study in the homepage domain, Proc. 6th Intl. World Wide Web Conference, Computer Networks 29 (8–13) (1997) 1193–1204.

[39] C. Vogt, G. Cottrell, Predicting the performance of linearly combined IR systems, Proceedings of the ACM SIGIR Conference, ACM Press, New York, NY, 1998, pp. 190–196.

[40] C. Watkins, Learning from delayed rewards, PhD thesis, King's College, Cambridge, UK, 1989.

Filippo Menczer is an Assistant Professor in the Department of Management Sciences at the University of Iowa, where he teaches courses in information systems. After receiving his Laurea in Physics from the University of Rome in 1991, he was affiliated with the Italian National Research Council. In 1998 he received a dual PhD in Computer Science and Cognitive Science from the University of California at San Diego. Dr. Menczer has been the recipient of Fulbright, Rotary Foundation, NATO, and Santa Fe Institute fellowships, among others. Dr. Menczer developed the MySpiders system, which allows users to launch personal adaptive intelligent agents who search the Web on their behalf. He also wrote the LEE artificial life simulation tool, distributed with Linux and widely used in experimental and instructional settings. The Adaptive Agents Research Group led by Dr. Menczer pursues interdisciplinary research projects spanning from ecological theory to distributed information systems; these contribute to artificial life, agent-based computational economics, evolutionary computation, neural networks, machine learning, and adaptive intelligent agents for Web, text, and data mining.