MySpiders: Evolve Your Own Intelligent Web Crawlers

GAUTAM PANT AND FILIPPO MENCZER   {gautam-pant, filippo-menczer}@uiowa.edu
Department of Management Sciences, The University of Iowa, Iowa City, IA 52242

Abstract. The dynamic nature of the World Wide Web makes it a challenge to find information that is both relevant and recent. Intelligent agents can complement the power of search engines to meet this challenge. We present a Web tool called *MySpiders*, which implements an evolutionary algorithm managing a population of adaptive crawlers who browse the Web autonomously. Each agent acts as an intelligent client on behalf of the user, driven by a user query and by textual and linkage clues in the crawled pages. Agents autonomously decide which links to follow, which clues to internalize, when to spawn offspring to focus the search near a relevant source, and when to starve. The tool is available to the public as a threaded Java applet. We discuss the development and deployment of such a system.

Keywords:

1. Introduction

The World Wide Web is a rapidly growing and changing information source. Its growth and change rates make the task of finding relevant and recent information harder. Search engines attempt to crawl the Web exhaustively for new pages, and to keep track of changes made to pages visited earlier. The crawled pages are stored in a static index mapping every term from a controlled vocabulary onto the collection of URLs containing that term. The index is used at query time to return the sets of URLs containing the terms in users' queries.

Due to the dynamic nature of the Web, the set of relevant pages for any given query is also highly dynamic, leading to a *scalability* problem—the assumption of an accurate and complete static image of the Web breaks down with its rate of change. As search engines fail to satisfy the user's need for complete and recently updated information, it becomes highly desirable to improve the coverage and recency of search engines.

One approach to address the coverage issue is the use of meta-search techniques to combine relevant sets from multiple search engines [7, 15, 16]. Yet even meta-search engines cannot locate recent pages unknown to the individual engines whose results are combined. A number of intelligent agents have been developed in recent years to help users manage the information available through the Web [1, 2, 8, 13, 14]. However, such agents have generally limited autonomy—either they rely completely on search engines, or they must be told where to go by the user, or they follow some fixed heuristics. Other systems, such as Fish Search [4] and Fetuccino [3], crawl at query time but are hindered by a lack of adaptability—all agents are identical clones following fixed search strategies.

We suggest a more radical solution to the scalability problem: complementing index-based search engines with intelligent search agents at the user's end. This paper describes an evolutionary multi-agent system designed to browse adaptively on behalf of the user to complement search engines. We discuss the implementation of such agents in Java.

2. *MySpiders*

2.1. *InfoSpiders algorithm*

InfoSpiders [9] is a multi-agent system for online, dynamic Web search. Each agent checks its information neighborhood (defined by hyperlinks) looking for new documents relevant to the user's query and having little or no interaction with other agents. *InfoSpiders* are able to display an intelligent behavior by evaluating the relevance of the document content with respect to the user's query, and by reasoning autonomously about future actions that mimic the browsing habits of human users. 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 aspects related to the query, while capturing the relevant features of each agent's local information environment.

InfoSpiders employ an evolutionary computation approach based on *local selection* [12]. An agent's "energy," related to the relevance of the pages visited by that agent, is accumulated over time (Figure 1). We want to reward agents that find relevant pages first, and not agents that visit those pages subsequently. The primary reasons for the first-time-only reward are to detect loops, prevent population explosion, and improve coverage. For this we employ a caching system, and the interaction among individuals is limited to sharing access to this cache. Reproduction and death occur in an autonomous fashion, when an agent's energy reaches a fixed threshold or runs out. Offspring internalize features of the documents leading to reproduction events, by expanding their parents' query representation with terms frequent in the birth pages.

InfoSpiders rely on traditional search engines to obtain a set of starting URLs pointing to pages supposedly relevant to the query submitted by the user. An agent is positioned at each of these starting URLs. Each agent analyzes its current page to decide the next link to follow. The analysis includes looking at a small set of words around each hyperlink. The frequencies of query matching terms in such a neighborhood are used as inputs to a neural net that scores each outlink. One link is then picked with a probability proportional to its score.

After a document has been visited, its similarity to the query is used as a learning signal to update the weights of the neural net. Further, the agent computes a fitness value to update its energy level. Since the energy dynamics of an agent determine its reproductive rate, determining an appropriate fitness function to map the quality and cost of visited pages into energy is crucial for the success of the system. Users will only use tools that are both effective and efficient. For this reason we use two components in our fitness functions: a benefit based on the similarity to the query,

```
MySpiders(query, MAX_PAGES) {
    starting_urls := search_engine(query);
    for agent (1..#INIT_URLS) {
        initialize(agent, query);
        agent.location = one_of(starting_urls);
        agent.energy := THETA / 2;
    }
    foreach agent {
        while (alive & (visited < MAX_PAGES)) {
            p := fetch_page(agent.location);
            lock(cache_semaphore);
            update(cache);
            unlock(cache_semaphore);
            agent.energy += fitness(p);
            learn(agent);
            if (agent.energy > THETA) {
              offspring := mutate(clone(agent));
              offspring.energy := agent.energy / 2;
              agent.energy -= offspring.energy;
            }
            elseif (agent.energy < 0) die(agent);
            agent.location = pick_new_location();
        }
    }
}
```

Figure 1. Pseudocode of *MySpiders*.

and a cost to account for the network resources used to download pages:

$$\texttt{fitness}(p) = sim(p,\ \texttt{query}) - c_t \frac{t(p)}{t_{\max}}$$

where $sim()$ is the cosine similarity function, $t(p)$ is the real time in which the page is downloaded, $t_{\max}$ is the timeout parameter for the socket connection, and

$$c_t = \frac{\#\texttt{INIT\_URLS} * \texttt{THETA}}{\texttt{MAX\_PAGES}}$$

is a constant designed to ensure that the system will crawl `MAX_PAGES` pages (cf. Figure 1). We have found through experiments that such a latency dependent cost helps in getting faster results without penalizing the quality [5].[1]

The output of the algorithm is a flux of links to documents, ranked by estimated relevance. The algorithm can terminate before `MAX_PAGES` documents if it is terminated by the user, or if the population goes extinct for lack of relevant pages. Additional details on the algorithm can be found elsewhere [9].

2.2. *MySpiders applet*

Due to the parallel nature of the *InfoSpiders* algorithm, multi-threading is expected to provide better utilization of resources as compared to a single thread (sequential) implementation. Since Java has built-in support for threads and allows for classes

to be loaded at runtime over the Web, we implemented a multi-threaded version of *InfoSpiders* as a Java applet, called *MySpiders*.

The multi-threaded implementation allows one agent to use the network connection to retrieve a document, while other agents can use the CPU, or access cache information on the local disk. Figure 1 illustrates the *MySpiders* applet. The only addition to the algorithm described in the previous section is a synchronization mechanism to allow concurrent access to the shared cache.

MySpiders is deployed on a public Web server.[2] Figure 2 shows the user interface of the applet in the course of a query search. *MySpiders* hides from the user many parameters in order to keep the user interface as simple as possible. For example, the parameter `MAX_PAGES` is provided by the user while `THETA` is determined empirically and hidden.

Once the start button is pressed, an automated browsing process is initiated. `#INIT_URLS` hits are retrieved from a search engine[3] and a spider is initiated at each of the links (Figure 2a). The user can scan through a ranked list of relevant pages found up to that time, in a tabular format. The relevant URLs are preceded by name of the agent (spider) that found the page. If the URL is one of the initial links from the search engine, it is preceded by the name of the search engine. The hierarchy of spiders is shown as an expandable tree structure (Figure 2b). If the user likes a page, clicking on the spider that found it provides additional details about the spider, e.g., its history of crawled pages (Figure 2c). Clicking on a URL in the results frame brings up the page in a browser window (Figure 2d). 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.

2.3. Software architecture

The *MySpiders* system can be divided into five subsystems—user interface, manager, shared objects, utilities, and spiders (Figure 3). The algorithm has many aspects that can be tuned for better performance. For example we could tweak some parameter values or change the fitness function to bias the search. We are also looking at using *InfoSpiders* in non-Web environments. Hence, we needed a system that can be easily reconfigured and extended. For the same reason, it was important to separate the front-end of the system from the algorithmic details. In the latest implementation, the user interface (described above) acts as a client that can query the back-end search agents for their status.

The isolation of the front-end from the agents is managed through use of a middle manager tier. The manager provides an API that allows the front-end to start and stop the search, provide user modifiable parameters, and to query for search status. The manager knows how to initiate and monitor threaded search agents. The current design makes it easy to plug-in a new search mechanism with minimal code change. For example, the latest implementation allows the user to change the crawling algorithm from *InfoSpiders* to *Best-First*, a crawler that has proved to be a valid alternative under certain circumstances [9, 11].[4]
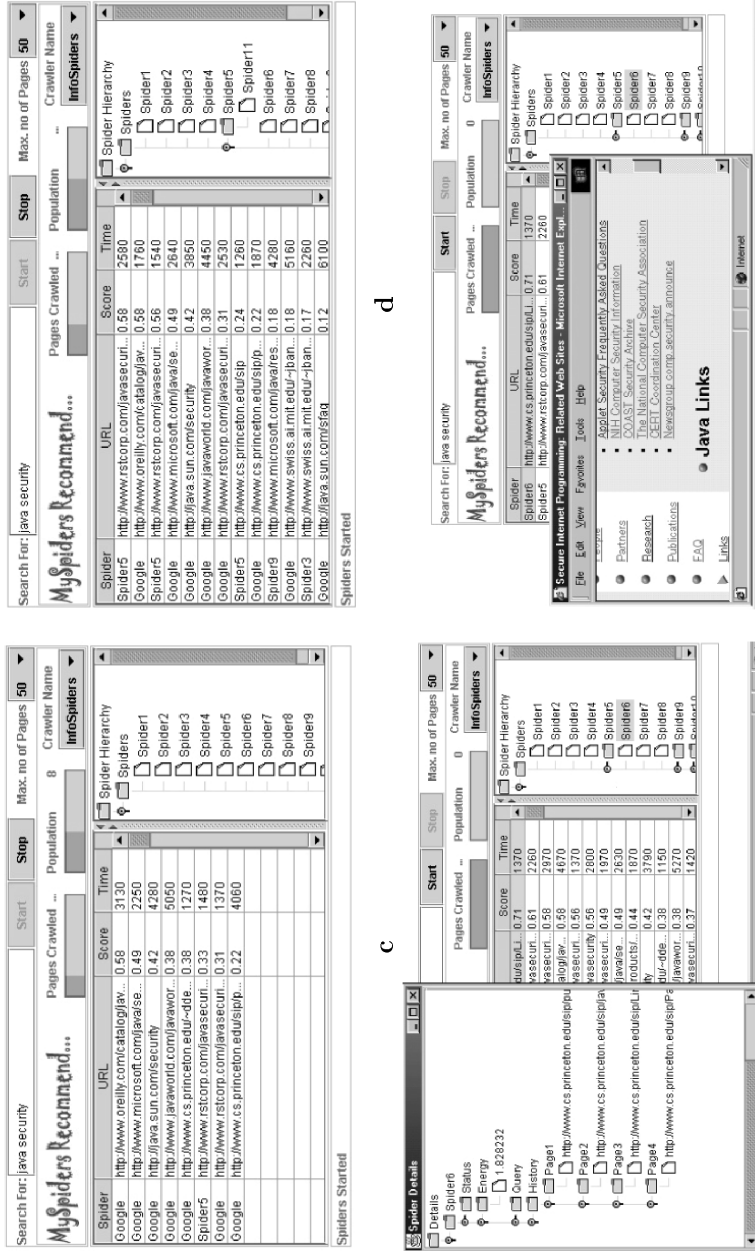
Figure 2. The user interface of *MySpiders* during a crawl. (a) Early in the search process; most of the top hits are from Google. (b) Later in the search process; Spider 5 has reproduced and its progeny is visible in the expandable tree. (c) At the end of the crawl; the details of Spider 6, who found the top hit, are displayed in a pop-up window. (d) Viewing the top hit by clicking its URL in the results frame.
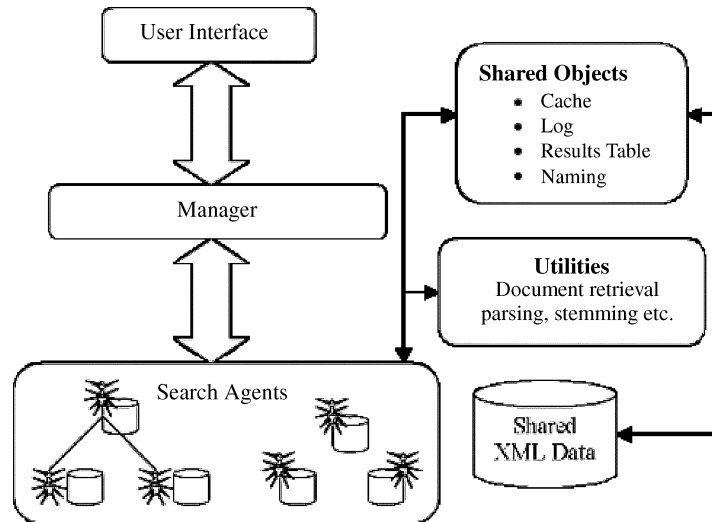
Figure 3. Software architecture of the system.

As mentioned before, the shared cache is essential for the working of the multi-threaded *MySpiders* system. Another shared object is the naming mechanism that provides a unique name to each spider. A results table is a shared object that maintains details about the pages that the spiders want to add to the results. Finally, a shared log object is used to maintain a system wide mechanism for logging major events. The shared objects other than the cache can be easily replaced by non-shared ones (as would be required for a distributed implementation), but have been kept for efficiency. Access to all shared objects is thread-safe.

The utilities include an entire gamut of functions that provide facilities for retrieving documents from the Web, HTML and XML parsing, stemming, measuring similarity and other tools used by most Web crawlers.

The architecture of a *MySpiders* agent is shown in Figure 4. In addition to the current location, the agent stores a back-link so that it can move to the previous page and escape (some) dead-ends. It also maintains a backup of inputs used by the neural network's learning algorithm.

All the back-end storage is done in XML. This aids system portability and helps in storing and parsing hierarchical data. An example is the spider hierarchy shown in Figure 2.

2.4. Java security

While Java is a powerful language there are issues that weaken the Java choice, the most important of which are the low speed of execution of Java byte code and the need to provide a mechanism for granting privileges to the applet. 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
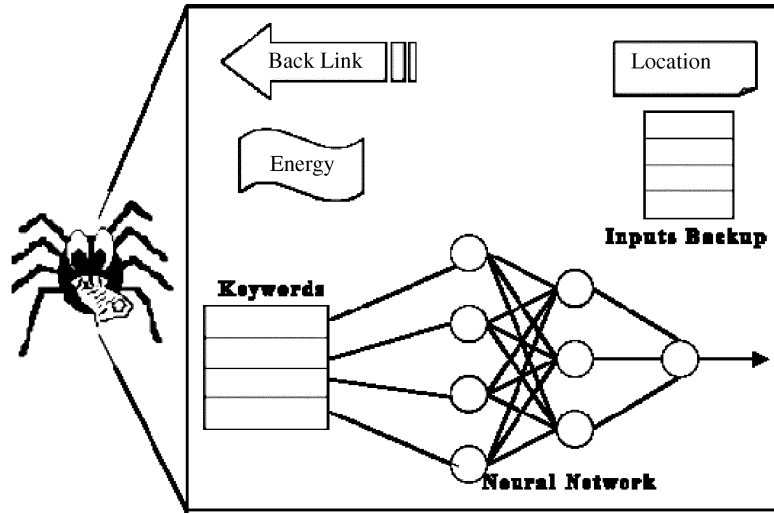
Figure 4. Representation of a *MySpiders* agent.

we used a combination of digitally signed applets and policy files [6]. Both of these solutions have platform dependencies that hinder portability and ease of use. We are currently exploring the use of the Java Network Launching Protocol and Java Web Start.

3. Conclusions

We have described *MySpiders*, a deployed system in which adaptive information agents browse the Web concurrently and autonomously on behalf of the user. This tool can be applied to several problems confronting information seekers:

- As a search refinement tool, to save the user from manually browsing through the hits returned by a search engine when the relevant pages may not have been indexed due to their recency.
- As a personalized background search and notification tool, serving long-standing queries such as user profiles.
- As a competitive intelligence tool, to monitor sites mining for new information on a given subject.
- As a portal crawler, to gather and index relevant pages and update specialized (topical or personal) search engines.

The object-oriented nature of Java and built-in facilities for multi-threading decreased our implementation effort. A modular software design makes it easy to reconfigure the system for various experiments. Though Java code is portable by nature, lack of adequate support for applets on popular browsers adds extra effort to its deployment. In addition, an applet based implementation of Web browsing agents suffers further portability limitations due to its security issues.

InfoSpiders has been evaluated favorably on a controlled corpus [9], as well as through anecdotic evidence on the Web [10]. We continue to evaluate the utility of the technique as compared to other state-of-the-art crawling methods [11]. The Applet has gained considerable visibility since its deployment (for example, there were 1,723 unique users in the month of April 2001).

Acknowledgments

Notes

1. Note that client connections with higher bandwidth also lead to higher population sizes.
2. http://myspiders.biz.uiowa.edu
3. We currently use http://www.google.com for this purpose.
4. *Best-First* selects links from a priority queue where it keeps outlinks from visited pages sorted by similarity of these pages to the query.

References

1. R. Armstrong, D. Freitag, T. Joachims, and T. Mitchell, "Webwatcher: A learning apprentice for the world wide web," in *AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, 1995.
2. M. Balabanović, "An adaptive web page recommendation service," in *Proc. 1st Int. Conf. Autonomous Agents*, 1997.
3. I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. Maarek, D. Pelleg, M. Shtalhaim, V. Soroka, and S. Ur, "Adding support for dynamic and focused search with fetuccino," in *Proc. 8th Int. World Wide Web Conf.* 1999.
4. P. De Bra and R. Post, "Information retrieval in the world wide web: Making client-based searching feasible," in *Proc. First Int. World Wide Web Conf.*, 1994.
5. M. Degeratu, G. Pant, and F. Menczer. "Latency-dependent fitness in evolutionary multithreaded web agents," in *Proc. GECCO Workshop on Evolutionary Computation and Multi-Agent Systems (ECOMAS'01)*, 2001.
6. L. Gong, *Inside Java 2 Platform Security: Architecture, API Design, and Implementation*, Addison Wesley, 1999.
7. S. Lawrence and C. Giles, "Context and page analysis for improved web search," *IEEE Internet Computing*, vol. 2, no. 4, pp. 38–46, 1998.
8. H. Lieberman, "Autonomous interface agents," in *Proc. ACM Conf. Computers and Human Interface*, Atlanta, GA, 1997.
9. F. Menczer and R. Belew, "Adaptive retrieval agents: Internalizing local context and scaling up to the web," *Machine Learning*, vol. 39, nos. 2/3, pp. 203–242, 2000.
10. F. Menczer and 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.

11. F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. "Evaluating topic-driven web crawlers," in *Proc. 24th Annual Int. ACM SIGIR Conf. Research and Development in Information Retrieval*, 2001.

12. F. Menczer, W. Street, and M. Degeratu, "Evolving heterogeneous neural agents by local selection," in M. Patel, V. Honavar, and K. Balakrishnan (eds.), *Advances in the Evolutionary Synthesis of Intelligent Agents*, MIT Press, 2001.

13. A. Moukas, "Amalthaea: Information discovery and filtering using a multiagent evolving ecosystem," in *Proc. Conf. Practical Applications of Intelligent Agent Technology*, 1996.

14. M. Pazzani, J. Muramatsu, and D. Billsus. "Syskill & Webert: Identifying interesting websites," in *Proc. National Conf. Artificial Intelligence (AAAI96)*, 1996.

15. E. Selberg and O. Etzioni, "The metacrawler architecture for resource aggregation on the web," *IEEE Expert*, vol. 12, no. 1, pp. 8–14, 1997, http://www.metacrawler.com.

16. C. Vogt and G. Cottrell, "Predicting the performance of linearly combined ir systems," in *Proc. ACM SIGIR Conf.*, 1998.

Annotations from agnt399757.pdf

Page 1

Annotation 1; Label: Gautam; Date: 2/14/2002 2:26:48 PM
au: Please supply keywords.

Annotation 2; Label: Gautam; Date: 2/14/2002 2:27:14 PM
au: OK to delete "s" from algorithm?