A Smart Itsy Bitsy Spider for the Web

A Smart Itsy Bitsy Spider for the Web

Hsinchun Chen, Yi-Ming Chung, and Marshall Ramsey
Artificial Intelligence Lab, Management Information Systems Department, University of Arizona, Tucson, AZ 85721. E-mail: hchen@bpa.arizona.edu; ychung@bpa.arizona.edu; mramsey@bpa.arizona.edu

Christopher C. Yang
Electrical and Computer Engineering Department, University of Arizona, Tucson, AZ 85721. E-mail: chrisy@ece.arizona.edu

As part of the ongoing Illinois Digital Library Initiative project, this research proposes an intelligent agent approach to Web searching. In this experiment, we developed two Web personal spiders based on best first search and genetic algorithm techniques, respectively. These personal spiders can dynamically take a user's selected starting homepages and search for the most closely related homepages in the Web, based on the links and keyword indexing. A graphical, dynamic, Java-based interface was developed and is available for Web access. A system architecture for implementing such an agent-based spider is presented, followed by detailed discussions of benchmark testing and user evaluation results. In benchmark testing, although the genetic algorithm spider did not outperform the best first search spider, we found both results to be comparable and complementary. In user evaluation, the genetic algorithm spider obtained significantly higher recall value than that of the best first search spider. However, their precision values were not statistically different. The mutation process introduced in genetic algorithm allows users to find other potential relevant homepages that cannot be explored via a conventional local search process. In addition, we found the Java-based interface to be a necessary component for design of a truly interactive and dynamic Web agent.

1. Introduction

Although network protocols and software such as HTTP and Netscape support significantly easy importation and fetching of online information sources, their use is accompanied by the disadvantage of the users' not being able to explore and find what they want in an enormous information space (Berners-Lee, Cailliau, Luotomen, Nielsen, & Secret, 1994; Bowman, Danzig, Manber, & Schwartz, 1994; Schatz, Bishop, Mischo, & Hardin, 1994). While Internet services are popular and appealing to many online users, difficulties with search are expected to worsen as the amount of online information increases. This is mainly due to the problems of information overload and vocabulary differences (Chen, 1994; Furnas, Landauer, Gomez, & Dumais, 1987). Many researchers consider that devising a scalable approach to Web search is critical to the success of Internet and Intranet services, and other current and future National Information Infrastructure (NII) applications (Chen & Schatz, 1994; Schatz & Chen, 1996).

The main information retrieval mechanisms provided by the prevailing Internet WWW-based software are based on either keyword search (e.g., Lycos, Alta Vista, and Yahoo servers) or hypertext browsing (e.g., NCSA Mosaic, Netscape Navigator, and Microsoft Internet Explorer). Keyword search often results in low precision, poor recall, and slow response time because of the limitations of indexing and communication methods (bandwidth), controlled language-based interfaces (the vocabulary problem), and the inability of searchers themselves to fully articulate their needs. Furthermore, browsing allows users to explore only a very small portion of the large Web information space. An extensive information space accessed through hypertext-like browsing can also potentially confuse and disorient its user, the ''embedded digression problem,'' and it can cause the user to spend a great deal of time while learning nothing specific, the ''art museum phenomenon'' (Carmel, Crawford, & Chen, 1992, p. 865).

Our proposed approach, which is grounded on automatic textual analysis of Web documents and general-purpose search algorithms, aims to address the Web search problem by creating dynamic and ''intelligent'' personal spiders (agents) that take users' requests and perform real-time, customized searches. In particular, best first search was adopted for a local search personal spider and a genetic algorithm was used to develop a global, stochastic personal spider. Such personal spiders (agents)

could dynamically take users' selected starting homepages and search for the most closely related homepages in the Web, based on links and keyword indexing. Extensive algorithmic revisions and interface development based on Java have been performed. This article summarizes our current research effort.

2. Literature Review: Intelligent Agents, Machine Learning, and Web Spiders

Our research is based on an intelligent agent approach to Internet/Intranet searching and relies significantly on machine learning algorithms. Although the system architecture for such an agent is non-trivial, we intentionally designed the spider to be compact, dynamic, and friendly—a ''smart itsy bitsy spider.''

2.1. Intelligent Agents

Broadly defined, an ''agent'' is a program that can operate autonomously and accomplish unique tasks without direct human supervision (similar to human counterparts such as real estate agents, travel agents, etc.). The basic idea of agent research is to develop software systems which *engage and help* all types of end users (Riecken, 1994). Such agents might act as ''spiders'' on the Web and look for relevant information (Etzioni & Weld, 1994), schedule meetings on behalf of executives based on their constraints, filter newsgroup articles based on ''induced'' (or learned) users' profiles (Maes, 1994), or assist meeting facilitators in converging ideas (Chen, Houston, Yen, & Nunamaker, 1996). Many researchers have focused on developing scripting and interfacing languages for designers and users such that they can create mobile agents of their own (Waldrop, 1994). Some researchers attempt to address the question of how agents should interact with each other to conduct digital teamwork. Other researchers are more concerned about designing agents which are ''intelligent'' (Riecken, 1994).

Agent research is new and broad; it includes research in different software engineering fields such as interface design, communication and coordination, adaptation and learning algorithms, etc. Various researchers have adopted different names, such as autonomous agents, adaptive interfaces, intelligent interfaces, knowbots, and intelligent agents.

However, many researchers believe that to be called ''intelligent,'' an agent must satisfy several interrelated criteria. Weld summarizes five attributes (Weld, 1995), which we believe capture the essence of an intelligent agent:

- *Integrated:* The agent must support an understandable, consistent interface.
- *Expressive:* The agent must accept requests in different modalities.
- *Goal-oriented:* The agent must determine how and when to achieve a goal.
- *Cooperative:* The agent must collaborate with the user.
- *Customized:* The agent must adapt to different users.

In summary, an intelligent agent must be capable of autonomous, customized, goal-oriented behavior in some environment that acts as a personal assistant to the user. In our research, our goal is to create intelligent agents that perform customized, dynamic search (information retrieval) and textual analysis in the World Wide Web environment for any user. In order to allow our agents to be *goal-oriented, cooperative,* and *customized,* we have adopted a machine learning approach based on our previous work in textual analysis.

2.2. Machine Learning

Several machine learning paradigms have been adopted recently for information retrieval and textual analysis, in particular, neural network, symbolic learning, and genetic algorithms (Chen, 1995).

• **Neural networks.** Neural networks model computation in terms of complex topologies (neurons and synapses) and statistics-based error correction (learning) algorithms. Neural networks computing fits well with conventional retrieval models such as the vector space model (Salton, 1989) and the probabilistic model (Maron & Kuhns, 1960). Doszkocs, Reggia, and Lin (1990) provided an excellent overview of the use of connectionist models in information retrieval. These models include several related information processing approaches, such as artificial neural networks, spreading activation models, associative networks, and parallel distributed processing. The work of Belew probably was the earliest connectionist model adopted in IR. In AIR (Belew, 1989), he developed a three-layer neural network of authors, index terms, and documents. The system used relevance feedback from its users to change its representation of authors, index terms, and documents over time. The result was a representation of the consensual meaning of keywords and documents shared by some group of users. Kwok (1989) developed a similar three-layer network of queries, index terms, and documents. A modified Hebbian learning rule was used to reformulate probabilistic information retrieval. However, representing all authors, index terms, and documents for a large-scale database application could pose severe difficulties, both in representations and in computation (especially for a real-time, interactive relevance-feedback process).

Lin, Soergel, & Marchionini (1991) adopted a Kohonen network for information retrieval. Kohonen's feature map, which produced a two-dimensional grid representation for N-dimensional features, was applied to construct a self-organizing (unsupervised learning), visual representation of the semantic relationships between input doc-

uments. In MacLeod and Robertson (1991), a neural algorithm developed by MacLeod was used for document clustering. The algorithm compared favorably with conventional hierarchical clustering algorithms. Chen et al. (Chen & Lynch, 1992; Chen, Lynch, Basu, & Ng, 1993; Chen & Ng, 1995) reported a series of experiments and system developments which generated an automatically-created weighted network of keywords from large textual databases and integrated it with several existing man-made thesauri (e.g., LCSH). Instead of using a three-layer design, Chen's systems developed a single-layer, interconnected, weighted/labeled network of keywords (concepts) for ''concept-based'' information retrieval. A blackboard-based design which supported browsing and automatic concept exploration using the Hopfield neural network's parallel relaxation method was adopted to facilitate the usage of several thesauri (Chen et al., 1993).

• **Symbolic learning.** Symbolic learning algorithms, rooted in traditional artificial intelligence research, are mostly based on production rule and decision tree knowledge representations. In Blosseville, Hebrail, Monteil, and Penot (1992), the researchers used discriminant analysis and a simple symbolic learning technique for automatic text classification. Their symbolic learning process represented the numeric classification results in terms of IF-THEN rules. Fuhr et al. (1990) adopted regression methods and ID3 for their feature-based automatic indexing technique. Crawford, Fung, and their coworkers (Crawford, Fung, Appelbaum, & Tong, 1991; Crawford & Fung, 1992; Fung & Crawford, 1990) have developed a probabilistic induction technique called CONSTRUCTOR and have compared it with the popular CART algorithm (Breiman, Friedman, Olshen, & Stone, 1984). Their experiment showed that CONSTRUCTOR's output is more interpretable than that produced by CART, but CART can be applied to more situations (e.g., real-valued training sets). Chen and She (1994) adopted ID3 and the incremental ID5R algorithm for information retrieval. Both algorithms were able to use user-supplied samples of desired documents to construct decision trees of important keywords which could represent the users' queries.

Recent work in logical and physical database design and analysis for relational and object-oriented database management systems (DBMS) applications adopted similar symbolic learning techniques. Cai et al. (Cai, Cercone, & Han, 1991; Han, Cai, & Cercone, 1993) developed an attribute-oriented, tree-ascending method for extracting characteristics and classification rules from relational databases. The technique relied on some existing conceptual tree for identifying higher-level, abstract concepts in the attributes. Ioannidis, Saulys, and Whitsitt (1992) examined the idea of incorporating symbolic machine learning algorithms (UNIMEM and COBWEB) into a database system for monitoring the stream of incoming queries and generating hierarchies containing the most important concepts expressed in those queries. The goal was for these hierarchies to provide valuable input for dynamically modifying the physical and logical designs of a database. Also related to database design, Borgida and Williamson (1985) proposed the use of machine learning to represent exceptions in databases that are based on semantic data models. Li and McLeod (1989) used machine learning techniques to handle object flavor evolution in object-oriented databases.

• **Genetic algorithms.** Genetic algorithms are a class of general purpose search methods that feature a stochastic, global search process. Based on principles of evolution and heredity, genetic algorithms strike a remarkable balance between exploration and exploitation of the search space (Michalewicz, 1992). Genetic algorithms are often compared with other conventional serial search methods, such as depth first search, breadth first search, best first search, hill climbing, branch and bound, and A* (Pearl, 1984).

Our literature search revealed several recent implementations of genetic algorithms in information retrieval. Gordon (1988) presented a genetic algorithms-based approach for document indexing. Competing document descriptions (keywords) were associated with a document and altered over time by using genetic mutation and crossover operators. In his design, a keyword represented a gene (a bit pattern), a document's list of keywords represented individuals (a bit string), and a collection of documents initially judged relevant by a user represented the initial population. Based on a Jaccard's score matching function (fitness measure), the initial population evolved through generations and eventually converged to an optimal (improved) population—a set of keywords which best described the documents. Gordon (1991) adopted a similar approach to document clustering. His experiment showed that after genetically altering the subject description of documents, descriptions of documents found co-relevant to a set of queries bunched together.

Raghavan and Agarwal (1987) also studied genetic algorithms in connection with document clustering. Petry, Buckles, Prabhu, and Kraft (1993) applied genetic programming to a weighted information retrieval system. In their research, a weighted Boolean query was modified in order to improve recall and precision. They found that the form of the fitness function had a significant effect upon performance. Yang and his coworkers (Yang & Korfhage, 1993; Yang, Korfhage, & Rasmussen, 1993) have developed adaptive retrieval methods based on genetic algorithms and the vector space model using relevance feedback. They reported the effect of adopting genetic algorithms in large databases, the impact of genetic operators, and GA's parallel searching capability. Frieder and Siegelmann (1991) also reported a data placement strategy for parallel information retrieval systems using a genetic algorithms approach. Their results compared favorably with pseudo-optimal document allocations.

Chen and Kim (1994–1995) reported a GA-neural-network hybrid system for information retrieval, called GANNET. The system performed *concept optimization* for user-selected documents using genetic algorithms. It then used the optimized concepts to perform *concept exploration* in a large network of related concepts through the Hopfield net parallel relaxation procedure. A Jaccard's score was also adopted to compute the ''fitness'' of subject descriptions for information retrieval.

Based on our experience in adopting machine learning in textual analysis and observation of the dynamic and complex nature of the Web, which essentially is a large and evolving directed graph of connected nodes (homepages), genetic algorithms appear to be well suited for such a complex search task.

2.3. Web Spiders

Internet and Intranet searching has been one of the hottest topics at recent World Wide Web Conferences. Two major approaches have been developed and experimented with: One is the client-based search spider (agent), and the other is online database indexing and searching. Most systems employ conventional serial search methods such as: Depth first search, breadth first search, and best first search. However, some systems contain components of both approaches.

• **Client-based search spiders (agents).** Several Web software programs based on the concept of spiders, agents, or softbots (software robots) have been developed. TueMosaic and the WebCrawler are two prominent early examples. Both of them use variations of conventional best first (local) search strategies (Pearl, 1984). DeBra and Post (1994) reported tueMosaic v2.42, modified at the Eindhoven University of Technology (TUE) using the ''fish search'' algorithm, at the First WWW Conference in Geneva. Using tueMosaic, users can enter keywords, specify the depth and width of search for links contained in the current homepages displayed, and request the spider agent to fetch homepages connected to the current homepage. The fish search algorithm is a modified best first search method. However, potentially relevant homepages that do not connect with the currently active homepages cannot be retrieved, and, when the depth and breadth of search become large (an exponential search), the search space becomes enormous. The inefficiency and local search characteristics of BFS/DFS-based spiders and the communication bandwidth bottleneck on the Web severely constrained the usefulness of such a local search approach. At the Second WWW Conference, Pinkerton (1994) reported a more efficient spider (crawler). The WebCrawler extends the tueMosaic's concept to initiate the search using its index and to follow links in an intelligent order. WebCrawler evaluates the relevance of a link based on its similarity to the anchor texts of the user's query. However, problems with local search and the communication bottleneck persist.

Due to the proliferation of WWW sites, many newer spiders having different functionalities recently have been developed. The TkWWW robot was developed by Spetka and funded by the Air Force Rome Laboratory (Spetka, 1994). TkWWW robots are dispatched from the TkWWW browser and are designed to search Web neighborhoods to find logically related homepages and return a list of ''hot'' links. However, their search process is limited to one or two local links from the original homepages. TkWWW robots can also be run in the background to build HTML indexes, compile WWW statistics, collect a portfolio of pictures, or perform any other functions that can be described by TkWWW Tcl extensions. WebAnts, developed by Leavitt at Carnegie Mellon University, investigates the distribution of information collection tasks to a number of cooperating agents (ants). The goal of WebAnts is to create cooperating agents that share searching results and the indexing load without repeating each other's effort. The RBSE (Respository Based Software Engineering) spider was developed by Eichmann and funded by NASA. RBSE spider was the first spider to index documents by content. It uses the Mite program to fetch documents and uses four local search mechanisms: (1) Breadth first search from a given URL, (2) limited depth first search from a given URL, (3) breadth first search from unvisited URLs in the database, and (4) limited depth first search from unvisited URLs in the database. (For a complete review of other similar Web spiders/agents, readers are referred to Cheong, 1996.)

• **Online database indexing and searching.** An alternative approach to Web resource discovery is based on the database concept of indexing and keyword searching. Such systems collect complete or partial Web documents using mostly breadth first search spiders and store all fetched homepages on the host server. These documents are then keyword indexed on the host server to provide a searchable interface. Most popular Web databases such as Lycos, Alta Vista, and Yahoo are based on such a design.

Lycos, developed at CMU (Mauldin & Leavitt, 1994), uses a combination of spider fetching and simple owner-registration. Web servers can access the Lycos server and complete registration in a few simple steps. In addition, Lycos uses spiders based on the connections to the registered homepages to identify other un-registered homepages. With this suite of techniques, Lycos has acquired an impressive list of URLs on the Web. Lycos adopted a heuristics-based indexing approach for these homepages that indexes them based on title, headings and subheadings, 100 most important words, first 20 lines, size in bytes, and number of words. However, Lycos's success also illustrates the vulnerability of the approach and the daunting task of creating ''intelligent'' and efficient Web search engines. Its popularity has caused a severe degra-

dation of information access performance, due to the communication bottleneck and the task of finding selected documents in an all-in-one database of Web homepages.

Alta Vista, developed at Digital's Research Laboratories in Palo Alto, combines a fast Web crawler with scalable indexing software to build a large index of the Web. It was made public on December 15, 1995 and has quickly become one of the most comprehensive searchable databases on the Web. It also provides a full-text index that is updated in real-time for over 13,000 news groups. Although based on similar local search spider algorithms, the Alta Vista server has been successful due to its superior hardware platforms and high-end communication bandwidth.

Instead of taking the all-in-one database approach adopted by Lycos and Alta Vista, the Yahoo server represents an attempt to partition the Web information space to provide meaningful subject categories (e.g., science, entertainment, engineering, etc.). However, its manually-created subject categories are limited in their granularity, and the process of creating such categories is cumbersome and time-consuming. The demand to create up-to-date and fine-grained subject categories, and the requirement that an owner place a homepage under a proper subject category has significantly hampered Yahoo's success and popularity.

Our spider design combines some of the key characteristics of the above two approaches.

3. A Smart Itsy Bitsy Spider: The System Architecture

A Web agent system architecture was designed first, before we proceeded to the actual implementation. The system architecture consists of five modular components, each accomplishing a unique task: (1) Task requests and search control parameters are solicited from the user; (2) a graphical user interface, previously developed in CGI/HTML and currently implemented in Java and on a client-server, takes user input and generates intermediate results and final search summary; (3) the Jaccard's similarity function computes a link score and a keyword score for each newly explored and indexed homepage; (4) a search engine supports both best first search and genetic algorithm; and (5) an HTTP protocol-based program to fetch a remote homepage (Fig. 1).

This modular design allows us to experiment with different subcomponents over time and eventually converge on a more compact and efficient system implementation. For example, a CGI/HTML-based and a Java-based GUI were independently designed and tested. Different combinations of Jaccard link weights and Jaccard keyword weights were tried. A simulated annealing-based spider was developed, tested, and then removed, leading to our eventual selection of the genetic algorithm-based spider. (Both the genetic algorithm spider and the simulated annealing spider achieved very similar search results. However, the genetic algorithm spider was more efficient and flexible.) For homepage fetching, Lynx and HtmlGobble, two freeware HTTP programs were implemented initially. After extensive experimentation, we decided to develop our own compact and efficient homepage fetcher in C.

3.1. Requests and Control Parameters

Users submit their search requests by providing several starting URLs and a few desired search keywords. The agents then perform searches from the staring URLs and find new homepages that match the starting URLs and keywords. In addition, a user can limit the number of URLs returned, the search time for each new homepage, and exercise other search engine parameters (e.g., crossover and mutation rates, to be discussed later). As a customized, cooperative agent, most parameters can be changed during an active search process to affect search results, making the agent a truly dynamic Web search assistant.

3.2. Graphical User Interface (GUI) and Client-Server

The Graphical User Interface (GUI) provides a link between the submitted task and control parameters and the search engine. It includes forms, images, scrollbars, and radio buttons that the user employs to fill up the task input and control parameters. It also displays the search results using tables and graphics. In our research, two interfaces were developed, one based on HTML and Common Gateway Interface (CGI), and the other based on Java.

CGI/HTML communicates with other programs running on the HTTP server. The Web server can activate a program with CGI and pass user-specific data. It then processes the data involved in such a task and sends the results back to the Web browser. The major shortcoming of the CGI interface is its lack of dynamic interaction. The server is completely event driven. It can only respond to requests from clients, but cannot initiate request of its own. In addition, it can only respond to one event at a time. These limitations lead to its being a relatively static user interface. This lack of dynamic interaction severely hampered the feasibility of the agent-based spider. For example, the search time of a spider usually takes from 5 to 20 minutes. Users are often frustrated when they are relegated to a passive role during searches, unable to view intermediate results or change parameters in reaction to search events. As a result, a truly dynamic and integrated user interface based on a Java client-server design and the TCP/IP protocol of the Internet was developed. The WWW protocol, HTTP, used by CGI programs, is inappropriate because it does not support two-way data communication between client and server.

Our client-server architecture was based on UNIX sockets and created in both C/C++ and Java. Java is an object-oriented language that is portable, platform inde-

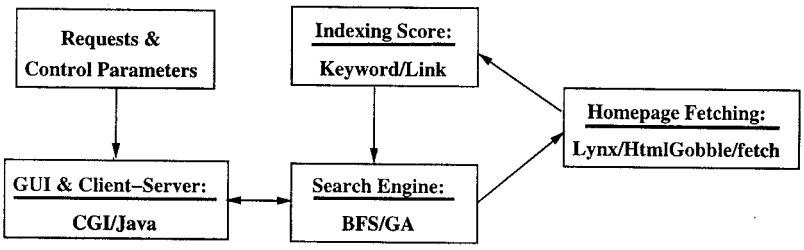

FIG. 1. A smart itsy bitsy spider: The system architecture.

pendent, and supports dynamic applications embedded in HTML documents. The client-server design was chosen because the majority of the indexing, search engine, and homepage fetching code on our server had been developed earlier in C and we needed to achieve efficiency during searches. While the server code was in C, the client program that takes task requests, control parameters, and displays intermediate and final results was written in Java. It can be invoked easily through Netscape browser.

UNIX sockets are used to build up the dynamic connection between the client and the server. A socket is a means by which two processes communicate using UNIX file descriptors—integer handles to disk files, pipes, FIFOs, etc., but it is not limited to UNIX platforms. For the spider, the file descriptor created is a unique handle to a buffered input–output stream, called a pipe, and connects the client and server processes even though they are running on different computers. A socket is first bound to a port (a numbered local communication address) on the server and then waits for the clients to connect to the Internet Protocol (IP) address and port number of that server. Once a connection is made, message passing is handled transparently by the Transmission Control Protocol (TCP)—one of the fundamental protocols of the Internet. Once a connection request is received from a client, a file descriptor is created for communication and a child process is spawned and receives the file descriptor as a parameter. Then, the parent process continues to listen for new connections. Meanwhile, the child process performs the search using the file descriptor to communicate with the client. This technique allows searches for multiple users to be performed simultaneously, each with its own unique file descriptor and handled by a different child process on the server.

After a connection is established between the client and server, token passing is employed to synchronize communication, and code words are used to interpret the transmitted data. A token represents permission to send data and is implemented by sending the word ''TOKEN.'' The server starts with the token and periodically passes it to the client in order to check whether it has any data to transmit. Once the token is obtained, the client sends its data and gives the token back to the server. While the server has the token, it is able to pass data to the client. This avoids the collision of data transmissions. A set of code words is prepended to fixed-length data or frame data of variable length sent between the client and server. These predefined terms will enable the programs to properly parse received data. For instance, if a client is sending the fixed-length mutation rate for a global search, it will prepend ''MUTATION'' to the rate before sending the data. For variable length data, such as keyword matches, the data will have ''ST_KEYS'' prepended and ''END_KEYS'' appended. The process receiving the data will then be able to determine that a keyword list is being received and tell where the data ends.

In order to reduce fetching time during the search process, which had been found to account for over 90% of the total process time, the server spawns a separate child process for each homepage that will be fetched. These children execute in parallel, and each is responsible for fetching one homepage. The server keeps track of the children and terminates them if results have not been returned before the maximum allowable time. Sample Java-based client-server graphical interface will be presented in Section 6.

3.3. Indexing Score/Fitness: The Jaccard's Function

In order to determine the ''goodness'' (or fitness, using GA terminology) of a given new homepage, a Jaccard's similarity function has been adopted (Rasmussen, 1992). Each homepage is represented as a weighted vector of keywords, which have been automatically indexed by our spider, and connecting links. Each new homepage fetched by the spider is compared with the anchor/starting homepages to determine whether or not it is promising. A new homepage which is more similar to the starting home-

pages is considered more promising and thus will be explored first. The Jaccard's functions adopted are based on the combined (equal) weights of the Jaccard's score from links and the Jaccard's score from keywords.

• **Jaccard's scores from links.** Given two homepages, A and B, and their respective connected links/URLs, $X = (x_1, x_2, \ldots, x_m)$ and $Y = (y_1, y_2, \ldots, y_n)$, the Jaccard's score between A and B based on links is computed as follows:

$$J_{link}(A, B) = \frac{\#(X \cap Y)}{\#(X \cup Y)} \quad (1)$$

where $\#(S)$ indicates the cardinality of set S. Intuitively, if a new homepage contains many links identical to those connected to the anchor homepages, it is considered more promising and thus should be explored first.

• **Jaccard's scores from keywords.** For a given homepage, terms are identified based on an automatic indexing procedure developed in our previous research (Chen et al., 1996). Term frequency (tf) and inverse document frequency (idf), term weighting heuristics also adopted in such popular searchable databases as Lycos, are then computed, Term frequency, tf_{ij}, represents the number of occurrences of term j in document (homepage) i. Homepage frequency, df_j, represents the number of homepages in a collection of N homepages in which term j occurs. The combined weight of term j in homepage i, d_{ij}, is computed as follows:

$$d_{ij} = tf_{ij} \times log\left(\frac{N}{df_j} \times w_j\right) \quad (2)$$

where w_j represents the number of words in term j, and N represents the total number of homepages connected to the starting homepages. In essence, two homepages that contain many identical keywords will obtain a high Jaccard's score.

Representing each homepage as a weighted vector of keywords, the Jaccard's score between homepages A and B based on keyword is computed as follows:

$$J_{keyword}(A, B) = \frac{\sum_{j=1}^{L} d_{Aj} d_{Bj}}{\sum_{j=1}^{L} d_{Aj}^2 + \sum_{j=1}^{L} d_{Bj}^2 - \sum_{j=1}^{L} d_{Aj} d_{Bj}} \quad (3)$$

where L is the total number of terms.

The combined Jaccard's score between any two homepages, A and B, is an equally-weighted summation of the above two Jaccard's scores, i.e.,

$$J(A, B) = 0.5 \times J_{link}(A, B) + 0.5 \times J_{keyword}(A, B). \quad (4)$$

In summary, our spider always explores new homepages that are similar (in link and keyword) to the anchor homepages during the entire heuristics-based search process.

3.4. Search Engine: BFS and GA

Two search algorithms, best first search and a genetic algorithm, were investigated in detail. The best first search algorithm was developed to simulate the various client-based spiders developed in earlier studies and was used as a benchmark for comparison. The genetic algorithm was adopted to enhance the global, optimal search capability of existing Web spiders.

3.4.1. Best First Search. Best first search is a serial state space traversal method (Pearl, 1984). In our implementation, the algorithm explores the best (based on Jaccard's score of new homepage vs. anchor homepages) homepage at each iteration and terminates when the system has identified the desired number of homepages requested by a user. A sketch of the best first search algorithm adopted in our personal agent is presented below:

1. Input anchor homepages and initialize. Initialize an iteration counter k to 0. Obtain a desired number of homepages from users and a set of input anchor homepages, ($input_1$, $input_2$, . . . , $input_m$). These input homepages represent the users' preferred starting points for Web search and their interests. Texts of homepages are fetched over the network, in real time, via HTTP communication software; homepages (URLs), connected from these input homepages, are extracted and saved in the unexplored homepage queue, H, where $H = (h_1, h_2, \ldots, h_n)$.

2. Determine the best homepage. Based on the Jaccard's function described earlier, remove the best homepage, p, in H, which has the highest Jaccard's score (against the anchor homepages) among all the homepages in H, and save it as $output_p$. This homepage is considered most similar to the anchor homepages in both keywords and links, and thus should be explored first.

3. Explore the best homepage. Fetch the best homepage and insert its connected homepages to the unexplored homepage queue, H. Increment iteration counter k by 1.

4. Iterate until a desired number of homepages is obtained. Repeat Steps 2 and 3 until k equals to the total number of homepages requested by the user.

Thus, the BFS spider explores one promising (local) link at a time, ranks all unexplored links during the process, and terminates when a desired number of homepages was obtained.

3.4.2. Genetic Algorithm. Genetic algorithms (GAs) (Goldberg, 1989; Koza, 1992; Michalewicz, 1992) are problem solving systems based on principles of evolution and heredity. Genetic algorithms perform a stochastic evolution process toward global optimization through the use of crossover and mutation operators. The search space of the problem is represented as a collection of individuals, which are referred as chromosomes. The quality of a chromosome is measured by a fitness function (Jaccard's score in our implementation). After initialization, each generation produces new children based on the genetic crossover and mutation operators. The process terminates when two consecutive generations do not produce noticeable population fitness improvement (i.e., reach a small threshold value or converge). Due to the difficulty of representing homepages as bit strings and applying conventional crossover and mutation operators (one of the classical problems in GA implementation), we have designed our system based on the general idea of GA (i.e., a global, stochastic search with a evolutionary process). Many domain-specific heuristics were adopted in our GA implementation. A sketch of the genetic algorithm adopted for Web client-based searching is presented below:

1. Initialize the search space. The GA spider attempts to find other most relevant homepages in the entire Web search space using the user-supplied starting homepages. Initially, the system saves all the input homepages in a set called Current Generation, $CG = (cg_1, cg_2, \ldots, cg_m)$.

2. Crossover. A heuristics-based crossover operation is then used. New homepages connected to starting homepages in *CG* set are extracted. Homepages that have been connected to multiple starting homepages (i.e., multiple parents) are considered Crossover Homepages and saved in a new set, $C = \{c_1, c_2, \cdots\}$. The crossover operator supports *exploitation* of promising local linkages and is similar to the best first search process.

3. Mutation. In order to avoid trapping in the local minimum that might result from adopting a simple crossover operator, we have added a heuristics-based mutation procedure to add diversity to the homepage population. A Yahoo spider created in our previous research is used to traverse the Yahoo's 14 high-level subject categories (e.g., science, business, entertainment, etc.) and collect several thousand ''mutation seed'' homepages in each category. These homepages are indexed using the Web indexing freeware, SWISH (Simple Web Indexing System for Humans). When the GA search algorithm requests a mutated homepage, the system retrieves the top-ranked homepage from homepages in the user-specified category based on the keywords presented in the anchor homepages. This process is similar to performing a search on the Yahoo database in order to suggest new, promising homepages for further exploration. New mutated homepages are saved in the set of Mutation Homepages, $M = \{m_1, m_2, \cdots\}$.

The probabilities of mutation and crossover can vary depending on user needs. Higher crossover probabilities generally support *exploitation* of local linkages, while higher mutation probabilities support *exploration* of the global landscape. *Exploitation* and *exploration* are two powerful features of genetic programming (Michalewicz, 1992). Our default settings for crossover and mutation probabilities both are 50%.

4. Stochastic selection scheme based on Jaccard's fitness. Each new crossover and mutation homepage is evaluated based on the same Jaccard's function. Based on an ''elicit selection'' procedure (Michalewicz, 1992), homepages which obtain higher fitness values are selected stochastically. A random number generator controlled by a homepage's fitness value is used to select ''fitter'' homepages for the new generation. Homepages that ''survive'' the (natural) selection procedure become the new population for the new generation.

5. Convergence. The above steps are repeated until the improvement in total fitness between two generations is less than a small threshold value (empirically determined). The final converged set of homepages is then presented to users as the output homepages.

3.5. Homepage Fetching

Several existing URL fetching programs are available on the Web. Among the most popular ones implemented for Web spiders are Lynx and HtmlGobble. Lynx is a full-featured Web client for users running cursor-addressable, character display devices such as vt100 terminals. It fetches and displays HTML documents residing on the local system, as well as files residing on remote servers running Gopher, HTTP, FTP, WAIS, and NNTP. Similarly, HtmlGobble fetches and displays HTML pages from remote Web sites. Both programs are big and feature many other Web page display and management functionalities.

In order to make our spider more portable and lightweight (i.e., ''itsy bitsy''), a simple and generic homepage fetching program was developed in *C* to replace Lynx and HtmlGobble that had been implemented in our earlier prototypes. The new program (called ''fetch'') takes a URL and an output filename as inputs and returns a complete HTML file. It first creates a socket and then establishes a connection to a server where the URL resides via a port (usually port 80). Requests are then sent to the HTTP server according to the HTTP protocol (RFC 1945). Once the requests are granted, HTML files are received. Using this generic fetching program, we were able to significantly speed up the fetching time of our

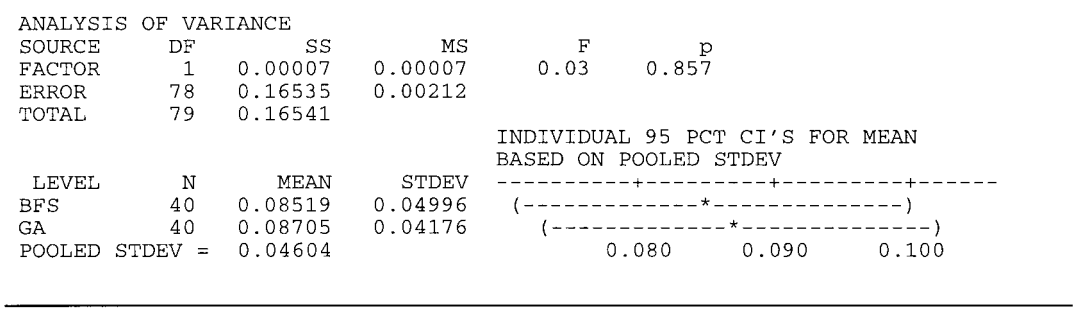

FIG. 2. Statistics of the average Jaccard's scores obtained from 40 test cases by best first search and genetic algorithm.

spider. As a result, the spider code is now more compact and portable.

4. Benchmarking Experiment and User Evaluation

A benchmarking experiment was conducted recently, followed by a user evaluation experiment.

4.1. Benchmarking Experiment

In an attempt to examine the quality of results obtained by best first search and genetic algorithm, we performed a set of benchmarking experiments which compared the performances and efficiency of the best first search and genetic algorithm-based personal spiders. Using a test set of 40 search scenarios, each composed of one to three homepages in different subject areas, we examined the final Jaccard's scores of the BFS/GA-suggested (top 10) homepages and their corresponding CPU times and wall clock times. Search scenarios were selected by the experimenters by grouping one to three homepages in a similar subject area, e.g., the UA/MIS AI group homepage (http://ai.bpa.arizona.edu) and the Illinois Digital Library Project homepage (http://dli.grainger.uiuc.edu/), both in digital library research. Higher Jaccard's scores of system-suggested homepages would suggest a closer match to a user's stated query interests (i.e., the anchor/starting homepages).

Figures 2, 3, and 4 show statistical analyses of the final fitness score, CPU time, and wall clock time of testing 40 cases.

• **Complementary searches through exploitation and exploration.** The results show that the output homepages obtained by genetic algorithm had a slightly higher fitness score than those obtained by best first search, but the difference was not significant. The averages of 40 Jaccard's scores for the genetic algorithm and the best first search were 0.08705 and 0.08519, respectively. Although the Jaccard's scores showed no significant difference between the performances of genetic algorithm and best first search, we noticed that about 50% of the homepages obtained from the genetic algorithm were the result of the mutation operation (crossover and mutation probabilities were set to 50% and 50%, respectively). We found that these homepages, although promising, had never been linked to the starting homepages, and thus could not have been obtained by any local search spiders (including our best first search spider). This suggests the potential usefulness of the genetic algorithm spider to supplement the local best first search spider, i.e., by permitting combination of the results in both sets.

During our experimentation, we also found that the genetic algorithm performed very similarly to best first search when the mutation probabilities were set low (say 5%) and crossover probabilities were high (say 95%). With limited mutation operations, the crossover operation in genetic algorithm accomplished a local *exploitation* process similar to that of a local best first search. The mutation process appears to have been instrumental in allowing our personal spider to get out of the local search minimum.

• **Sparse link constraint.** In addition, we also noticed that starting homepages played an important role in determining system output. If starting homepages contained very few and sparsely connected links, the best first search spider tended to get trapped quickly in the Web search space because of lack of traversal paths. The genetic algorithm, however, was not restricted by such a sparse link constraint because of its mutation operator. On the other hand, for starting homepages that contained rich and dense connections, best first search often resulted in a fruitful final search set. The genetic algorithm added only limited diversity in such a scenario.

• **Exploration and communication are time consuming.** The genetic algorithm-based spider was significantly more time consuming than the best first search spider, as shown in Figures 3 and 4. The average CPU times for best first search and genetic algorithm were 3 minutes and 11 seconds, and 1 minute and 51 seconds, respectively. However, due to the communication bottleneck, the average wall clock time for best first search and genetic algorithm were 45 minutes and 41 seconds, and

```
ANALYSIS OF VARIANCE
SOURCE      DF         SS         MS         F         p
FACTOR       1     129766     129766     12.81     0.001
ERROR       78     790235      10131
TOTAL       79     920001

                                       INDIVIDUAL 95 PCT CI'S FOR MEAN
                                       BASED ON POOLED STDEV
 LEVEL       N       MEAN      STDEV  -+---------+---------+---------+-----
BFS         40      111.3      118.4   (-------*-------)
GA          40      191.9       79.0                    (-------*-------)
                                       -+---------+---------+---------+-----
POOLED STDEV =      100.7             80        120       160       200
```

FIG. 3. Statistics of the CPU time for 40 test cases by best first search and genetic algorithm.

23 minutes and 45 seconds, respectively. The SWISH keyword search procedure implemented in the genetic algorithm spider caused a significant CPU time requirement. The elite selection procedure and multiple generations also consumed significant CPU cycles.

However, the communication bandwidth (i.e., the actual time to fetch a remote homepage) seemed to be the most significant bottleneck of the entire process for both spiders. This deficiency can only be resolved when the current Web backbones are upgraded.

4.2. User Evaluation

In an attempt to further examine the performance of the spiders based on actual user evaluation, we performed a follow-up experiment that involved 256 evaluation tasks. In the user evaluation experiment, input (anchor) homepages and the resulting output homepages of the searches conducted by genetic algorithm and best first search in the benchmarking experiment were provided to human subjects for relevance evaluation. The subjects were asked to decide whether each of the output homepages was relevant to any of the input homepages. Only 32 of the 40 test cases in our initial experiment were used because Web was evolving quickly and eight test cases became out of date during the follow-up evaluation experiment. In addition, five random homepages were added to the genetic algorithm and best first search results to make up an evaluation pool for each task. Eight college-student subjects were recruited to evaluate each of the 32 cases. For each case, we provided two form-based browsers to the users, one for the input homepages and one for the output homepages. Using these forms, users were able to quickly evaluate the relevance of the resulting homepages.

- **Genetic algorithm achieved better recall than best first search, but precision levels were similar.** The subject-selected homepages from each evaluation pool were used as the target set of relevant homepages. Using standard information retrieval system evaluation measures including recall and precision (Salton, 1989), we were able to determine the performance of the two search methods based on users' evaluation. The recall and precision results are shown in Figures 5 and 6, respectively. Overall, genetic algorithm significantly (at 5% significance level) out-performed best first search in recall, but their precision levels were comparable. Genetic algorithm achieved a 61.84% recall level and a 67.71% recall level; whereas best first search achieved a 55.66% recall level and a 65.30% recall level.

- **Complementary searches through exploitation and exploration.** Similar to the conclusions drawn from the benchmarking experiment, we also found the best first search and genetic algorithm search results

```
ANALYSIS OF VARIANCE
SOURCE      DF         SS         MS         F         p
FACTOR       1   34654232   34654232      6.92     0.010
ERROR       78  390805280    5010324
TOTAL       79  425459520

                                       INDIVIDUAL 95 PCT CI'S FOR MEAN
                                       BASED ON POOLED STDEV
 LEVEL       N       MEAN      STDEV  --+---------+---------+---------+----
BFS         40       1425       1565   (--------*--------)
GA          40       2741       2752                    (--------*--------)
                                       --+---------+---------+---------+----
POOLED STDEV =       2238             800       1600      2400      3200
```

FIG. 4. Statistics of the wall clock time for 40 test cases by best first search and genetic algorithm.

```
ANALYSIS OF VARIANCE
SOURCE      DF         SS         MS         F         p
FACTOR       2    18.9463     9.4731    247.73     0.000
ERROR      765    29.2529     0.0382
TOTAL      767    48.1992

                                        INDIVIDUAL 95 PCT CI'S FOR MEAN
                                        BASED ON POOLED STDEV
 LEVEL       N       MEAN      STDEV   ------+---------+---------+---------+
GA         256     0.6184     0.2083         (-*-)
BFS        256     0.5566     0.2375   (-*-)
Combined   256     0.9163     0.1221                                 (-*-)
                                        ------+---------+---------+---------+
POOLED STDEV =     0.1955               0.60      0.72      0.84      0.96
```

FIG. 5. Recall statistics for 256 cases of user evaluation.

to be complementary. As shown in Figure 5, the recall level for the combined results from best first search and genetic algorithm (after removing duplicates) was at 91.63%, almost double that of the individual search results. Similar observations could be made about the characteristics of the two search methods. Best first search spider was able to perform excellent local exploitation, while genetic algorithm's mutation operation significantly improved the spider's ability to perform global exploration.

5. A Dynamic, Agent-Based Interface

Currently, we have developed two interfaces for our spiders. One is based on CGI/HTML and the other is based on Java. The CGI/HTML implementation enables image maps and fill-out forms to interact with the HTTP server. However, it is static and does not support dynamic display and interaction during the search process.

On the other hand, Java is an object-oriented, platform-independent, multi-threaded, dynamic, graphical, general-purpose programming environment for the Internet, Intranet, and other complex, distributed networks. The Java interface allows us to display lively intermediate spider search results and accept changes of input parameters (e.g., crossover and mutation probabilities) dynamically. These dynamic, interactive features of Java are crucial to design of integrated, customized, and intelligent agents. The Java prototype interface is summarized below. Readers are encouraged to connect to the the University of Arizona Artificial Intelligence Group homepage (http://ai.bpa.arizona.edu/) for actual demonstrations.

The Java request and control parameter interface is shown in Figure 7. When the genetic algorithm spider is selected, the system displays a dialog block similar to that designed for the CGI/HTML genetic algorithm spider interface. In addition, users can set their preferred crossover and mutation probabilities. A timeout mechanism was also introduced to avoid time-consuming, unfruitful connections.

Figure 8 shows the window which displays the intermediate and final search results dynamically and graphically. The same control panel is displayed at the top of the window. All input parameters can be changed during an ongoing search process, producing different search results. For example, a user may wish to set a high initial crossover rate to fully explore the local linkages first. When the local search results appear converged, the user can gradually increase the mutation rate to encourage the spider to explore a larger information space. Such real-time analysis and dynamic relevance feedback are crucial to a timely and comprehensive search process. A user can

```
ANALYSIS OF VARIANCE
 SOURCE      DF         SS         MS         F         p
 FACTOR       2     0.1156     0.0578      0.76     0.470
 ERROR      765    58.5270     0.0765
 TOTAL      767    58.6426

                                        INDIVIDUAL 95 PCT CI'S FOR MEAN
                                        BASED ON POOLED STDEV
  LEVEL      N       MEAN      STDEV   -----+---------+---------+---------+-
 GA         256     0.6771     0.2641            (-----------*----------)
 BFS        256     0.6530     0.3076    (-----------*----------)
 Combined   256     0.6496     0.2553   (-----------*----------)
                                        -----+---------+---------+---------+-
 POOLED STDEV =     0.2766              0.630     0.660     0.690     0.720
```

FIG. 6. Precision statistics for 256 cases of user evaluation.

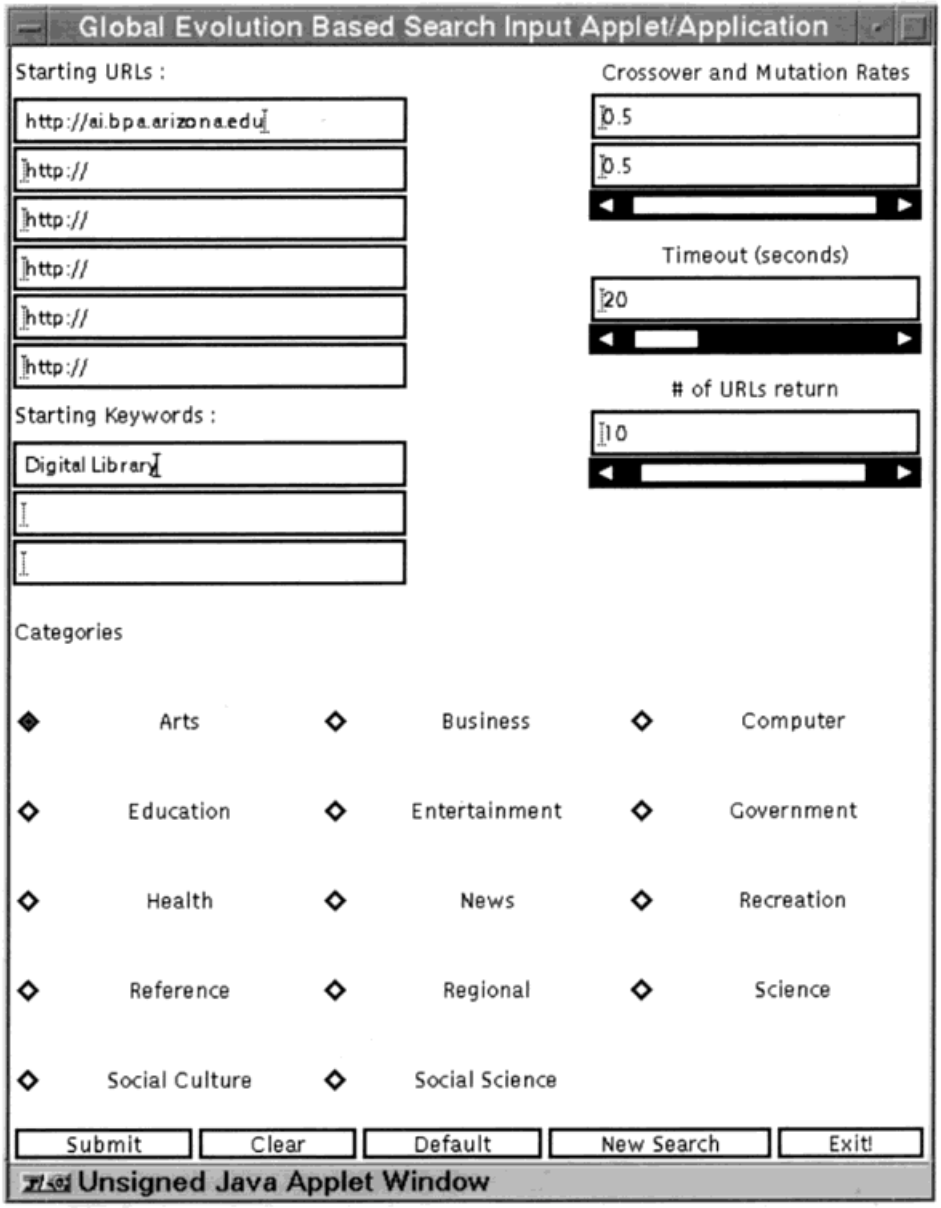

FIG. 7. The control panel for initiating a Java-based genetic algorithm spider.

also initially request a small number of URLs to explore an area of interest and then gradually increase the number of requested URLs at the end of a search process when he/she feels more comfortable about the initial results. Similarly, increasing the timeout limit also allows our spider to work harder, i.e., fetching URLs that are more remote or busy. Such agent-based searches (i.e., integrated, expressive, cooperative, goal-oriented, and customized) were difficult to implement in our earlier CGI/HTML-based spider interface.

As shown in Figure 8, the fetched URLs are displayed during each generation (instead of the the final results at the last generation) and can be clicked on for real-time evaluation and analysis. The system also graphically summarizes the Jaccard's link score, keyword score, and fetch time score for each homepage (in three different colored bars). A spider-chasing-fly animation is displayed dynamically when the spider is out chasing a new homepage (fly). At the end of a search session (after genetic algorithm converged), the spider falls asleep, as shown by a sleeping spider animation.

Since we deployed our Java-based spider on our server in Summer 1996, the response from our initial test subjects has been overwhelming. Users found the Java-based interface to be more interactive, lively, and friendly than our earlier CGI/HTML interface. They have reported our spiders to be a dynamic, intelligent personal agent, instead of a static, non-customized Web database search engine.

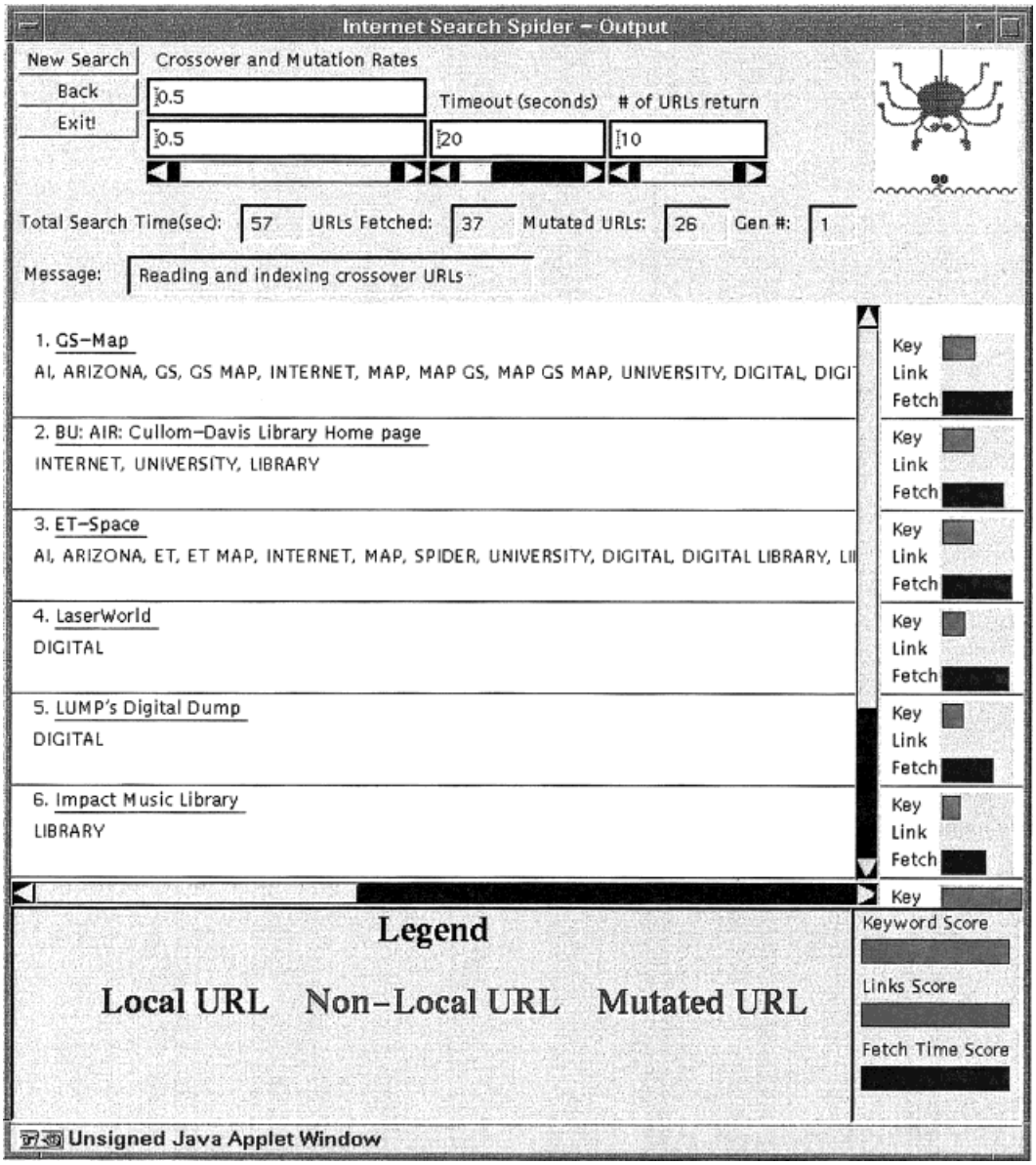

FIG. 8. The display window shows the result of the search process dynamically. Animation is displayed at the upper right-hand corner. Control panel which allows user to change parameters during the process is located at the upper portion. Search results are summarized at the center of the window.

6. Conclusion and Discussion

The results from our current experimentation of Web personal spiders are encouraging. In a benchmarking experiment, although the genetic algorithm spider did not outperform the best first search spider in final Jaccard's score, we found both results to be comparable and complementary. The mutation process introduced in genetic algorithm allows users to find other potential relevant homepages that cannot be explored via a conventional local search process. A user evaluation experiment also showed that genetic algorithm outperformed best first search in recall, although their results were complementary. In precision, the two search methods were comparable. In addition, we found our Web agent system architecture to be modular and flexible and it supported our extensive experimentations and gradual improvement toward the goal of creating a ''smart itsy bitsy spider.'' The recent Java-based interface we developed also proved to be invaluable for creating a truly ''intelligent'' agent-based system.

In our ongoing effort in the Illinois Digital Library Initiative project, we are in the process of exploring other general-purpose search and classification algorithms for Web resource categorization and search. Several neural network-based algorithms have been explored, including the Hopfield network and the Kohonen self-organizing map. Both are under development in Java.

7. Acknowledgments

This project was supported mainly by the following grants:

- NSF/ARPA/NASA Digital Library Initiative, IRI-9411318, 1994–1998 (B. Schatz, H. Chen et al., ''Building the Interspace: Digital Library Infrastructure for a University Engineering Community'');
- NSF CISE, IRI-9525790, 1995–1998 (H. Chen, ''Concept-Based Categorization and Search on Internet: A Machine Learning, Parallel Computing Approach'');
- AT&T Foundation Special Purpose Grants in Science and Engineering, 1994–1996 (H. Chen); and
- National Center for Supercomputing Applications (NCSA), High-Performance Computing Resources Grants, 1994–1996 (H. Chen).

Larry Smarr, Bruce Schatz, Joseph Harding, Charles Herring, and Kevin Powell of NCSA were instrumental in stimulating a lot of creative ideas for our research. We would like to thank the University of Arizona Artificial Intelligence Group members for their participation in our experiment. We also thank Prof. Jerome Yen of Hong Kong University and Prof. Pai-chun Ma of Hong Kong University of Science and Technology for their comments and involvement during system development and testing.

References

Belew, R. K. (1989). Adaptive information retrieval. In N. J. Belkin & C. J. van Rijsbergen (Eds.), *Proceedings of the Twelfth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval,* June 25–28, 1989, Cambridge, MA (pp. 11–20). New York: ACM Press.

Berners-Lee, T., Cailliau, R., Luotonen, A., Nielsen, H. F., & Secret, A. (1994, August). The World-Wide Web. *Communications of the ACM, 37(8),* 76–82.

Blosseville, M. J., Hebrail, G., Monteil, M. G., & Penot, N. (1992). Automatic document classification: Natural language processing, statistical analysis, and expert system techniques used together. In N. Bekin, P. Ingwersen, & A. M. Pejtersen (Eds.), *Proceedings of the Fifteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval,* June 21–24, 1992, Copenhagen, Denmark (pp. 51–57). New York: ACM Press.

Borgida, A., & Williamson, K. E. (1985). Accommodating exceptions in a database, and refining the schema by learning from them. In *Proceedings of the 11th International VLDB Conference,* August 1985, Stockholm, Sweden (pp. 72–81).

Bowman, C. M., Danzig, P. B., Manber, U., & Schwartz, F. (1994, August). Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM, 37(8),* 98–107.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression tree.* Monterey, CA: Wadsworth.

Cai, Y., Cercone, N., & Han, J. (1991). Attribute-oriented induction in relational databases. In G. Piatetsky-Shapiro & W. J. Frawley (Eds.), *Knowledge discovery in databases* (pp. 213–228). Cambridge, MA: The MIT Press.

Carmel, E., Crawford, S., & Chen, H. (1992, September/October). Browsing in hypertext: A cognitive study. *IEEE Transactions on Systems, Man and Cybernetics, 22(5),* 865–884.

Chen, H. (1994, May). Collaborative systems: Solving the vocabulary problem. *IEEE Computer,* Special Issue on Computer-Supported Cooperative Work (CSCW), *27(5),* 58–66.

Chen, H. (1995). Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science, 46,* 194–216.

Chen, H., Houston, A., Yen, J., & Nunamaker, J. F. (1996, August). Toward intelligent meeting agents. *IEEE Computer, 29(8),* 62–70.

Chen, H., & Kim, J. (1994–1995, Winter). GANNET: A machine learning approach to document retrieval. *Journal of Management Information Systems, 11(3),* 7–41.

Chen, H., & Lynch, K. J. (1992, September/October). Automatic construction of networks of concepts characterizing document databases. *IEEE Transactions on Systems, Man and Cybernetics, 22(5),* 885–902.

Chen, H., Lynch, K. J., Basu, K., & Ng, D. T. (1993, April). Generating, integrating, and activating thesauri for concept-based document retrieval. *IEEE Expert, Special Series on Artificial Intelligence in Text-based Information Systems, 8(2),* 25–34.

Chen, H., & Ng, D. T. (1995). An algorithmic approach to concept exploration in a large knowledge network (automatic thesaurus consultation): Symbolic branch-and-bound vs. connectionist Hopfield net activation. *Journal of the American Society for Information Science, 46,* 348–369.

Chen, H., & Schatz, B. R. (1994). Semantic retrieval for the NCSA Mosaic. In *Proceedings of the Second International World Wide Web Conference '94,* October 17–20, 1994, Chicago, IL. Urbana-Champaign, IL: NCSA.

Chen, H., Schatz, B. R., Ng, T. D., Martinez, J. P., Kirchhoff, A. J., & Lin, C. (1996, August). A parallel computing approach to creating engineering concept spaces for semantic retrieval: The Illinois Digital Library Initiative Project. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 18(8),* 771–782.

Chen, H., & She, L. (1994). Inductive query by examples (IQBE): A machine learning approach. In T. N. Mudge & B. D. Shriver (Eds.), *Proceedings of the 27th Annual Hawaii International Conference on System Sciences (HICSS-27), Information Sharing and Knowledge Discovery Track,* January 4–7, 1994, Maui, HI. Piscataway, NJ: IEEE.

Cheong, F. (1996). *Internet agents.* Indianapolis, IN: New Riders Publishing.

Crawford, S. L., Fung, R., Appelbaum, L. A., & Tong, R. M. (1991). Classification trees for information retrieval. In *Proceedings of the 8th International Workshop on Machine Learning,* (pp. 245–249). Morgan Kaufmann.

Crawford, S. L., & Fung, R. M. (1992, June). An analysis of two probabilistic model induction techniques. *Statistics and Computing, 2(2),* 83–90.

DeBra, P., & Post, R. (1994). Information retrieval in the World-Wide Web: Making client-based searching feasible. In *Proceedings of the First International World Wide Web Conference '94,* Geneva, Switzerland.

Doszkocs, T. E., Reggia, J., & Lin, X. (1990). Connectionist models and information retrieval. *Annual Review of Information Science and Technology (ARIST), 25,* 209–260.

Etzioni, O., & Weld, D. (1994, July). A softbot-based interface to the Internet. *Communications of the ACM, 37(7),* 72–79.

Frieder, O., & Siegelmann, H. T. (1991). On the allocation of documents in multiprocessor information retrieval systems. In *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval,* October 13–16, 1991, Chicago, IL (pp. 230–239).

Fuhr, N., Hartmann, S., Knorz, G., Lustig, G., Schwantner, M., & Tzeras, K. (1990). AIR/X—A rule-based multistage indexing system for large subject fields. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* July 29–August 3, 1990, Boston, MA (pp. 789–795).

Fung, R., & Crawford, S. L. (1990). Constructor: A system for the induction of probabilistic models. In *Proceedings of the Eighth National Conference on Artificial Intelligence (AAAI-90),* July 29–August 3, 1990, Boston, MA (pp. 762–769).

Furnas, G. W., Landauer, T. K., Gomez, L. M., & Dumais, S. T. (1987, November). The vocabulary problem in human-system communication. *Communications of the ACM, 30(11),* 964–971.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization, and machine learning.* Reading, MA: Addison-Wesley.

Gordon, M. (1988, October). Probabilistic and genetic algorithms for document retrieval. *Communications of the ACM, 31(10),* 1208–1218.

Gordon, M. D. (1991). User-based document clustering by redescribing subject descriptions with a genetic algorithm. *Journal of the American Society for Information Science, 42,* 311–322.

Han, J., Cai, Y., & Cercone, N. (1993, February). Data-driven discovery of quantitative rules in relational databases. *IEEE Transactions on Knowledge and Data Engineering, 5(1),* 29–40.

Ioannidis, Y. E., Saulys, T., & Whitsitt, A. J. (1992, July). Conceptual learning in database design. *ACM Transactions on Information Systems, 10(3),* 265–293.

Koza, J. R. (1992). *Genetic programming: On the programming of computers by means of natural selection.* Cambridge, MA: The MIT Press.

Kwok, K. L. (1989). A neural network for probabilistic information retrieval. In *Proceedings of the Twelfth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval,* June 25–28, 1989, Cambridge, MA (pp. 21–30).

Li, Q., & McLeod, D. (1989). Object flavor evolution through learning in an object-oriented database system. In L. Kerschberg (Ed.), *Expert Database Systems, Proceedings from the Second International Conference,* (pp. 469–495). Menlo Park, CA: Benjamin/Cummings.

Lin, X., Soergel, D., & Marchionini, G. (1991). A self-organizing semantic map for information retrieval. In A. Bookstein, Y. Chiaramella, G. Salton, & V. V. Raghavan (Eds.), *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval,* October 13–16, 1991, Chicago, IL (pp. 262–269). New York: ACM.

MacLeod, K. J., & Robertson, W. (1991). A neural algorithm for document clustering. *Information Processing & Management, 27(4),* 337–346.

Maes, P. (1994, July). Agents that reduce work and information overload. *Communications of the ACM, 37(7),* 30–40.

Maron, M. E., & Kuhns, J. L. (1960, July). On relevance, probabilistic indexing and information retrieval. *Journal of the ACM, 7(3),* 216–243.

Mauldin & Leavitt. (1994). Web-agent related research at the CMT. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval (SIGNIDR-94),* August 1994.

Michalewicz, Z. (1992). *Genetic algorithms + data structures = evolution programs.* Berlin/Heidelberg: Springer-Verlag.

Pearl, J. (1984). *Heuristics: Intelligent search strategies for computer problem solving.* Reading, MA: Addison-Wesley.

Petry, F., Buckles, B., Prabhu, D., & Kraft, D. (1993). Fuzzy information retrieval using genetic algorithms and relevance feedback. In *Proceedings of the ASIS Annual Meeting,* (pp. 122–125).

Pinkerton, B. (1994). Finding what people want: Experiences with the WebCrawler. In *Proceedings of the Second International World Wide Web Conference '94,* October 17–20, 1994, Chicago, IL. Urbana-Champaign, IL: NCSA.

Raghavan, V. V., & Agarwal, B. (1987). Optimal determination of user-oriented clusters: An application for the reproductive plan. In *Proceedings of the Second International Conference on Genetic Algorithms and Their Applications,* (pp. 241–246). July 1987, Cambridge, MA.

Rasmussen, E. (1992). Clustering algorithms. In W. B. Frakes & R. Baeza-Yates (Eds.), *Information retrieval: Data structures and algorithms.* Englewood Cliffs, NJ: Prentice Hall.

Riecken, D. (1994, July). Intelligent agents. *Communications of the ACM, 37(7),* 18–21.

Salton, G. (1989). *Automatic text processing.* Reading, MA: Addison-Wesley.

Schatz, B. R., Bishop, A., Mischo, W., & Hardin, J. (1994). Digital library infrastructure for a university engineering community. In J. L. Schnase, J. J. Leggett, F. Furuta, & T. Metcalfe (Eds.), *Proceedings of Digital Libraries '94,* June 1994 (pp. 21–24).

Schatz, B. R., & Chen, H. (1996, May). Building large-scale digital libraries. *IEEE Computer, 29(5),* 22–27.

Spetka, S. (1994). The TkWWW robot: Beyond browsing. In *Proceedings of the Second World Wide Web Conference,* October 17–20, 1994.

Waldrop, M. M. (1994, August 12). Software agents prepare to sift the riches of cyberspace. *Science, 265,* 882–883.

Weld, D. (1995, Fall). The role of intelligent systems in the national information infrastructure. *AI Magazine,* pp. 45–64.

Yang, J., & Korfhage, R. R. (1993). Effects of query term weights modification in document retrieval: A study based on a genetic algorithm. In *Proceedings of the Second Annual Symposium on Document Analysis and Information Retrieval,* April 26–28, 1993, Las Vegas, NV (pp. 271–285).

Yang, J., Korfhage, R. R., & Rasmussen, E. (1993). Query improvement in information retrieval using genetic algorithms: A report on the experiments of the TREC project. In *Text Retrieval Conference (TREC-1),* November 4–6, 1993, Gaithersburg, MD (pp. 31–58).