The Java Search Agent Workshop

Hsinchun Chen, Marshall Ramsey, and Po Li

MIS Department, Karl Eller Graduate School of Management
University of Arizona
Tucson, AZ 85721, USA

Abstract. As part of the ongoing Illinois Digital Library Initiative project, this paper presents the Java Search Agent Workshop (JSAW), a testbed designed for Java-based information searching. Based on artificial intelligence, neural networks, and G-Search, we implemented several search methods in Java to demonstrate their feasibility in various database, Internet, Intranet, and digital library search tasks. In addition to detailing our design rationale and implementation status, we present several sample Java implementations including a best first search spider and G-Search spider for Internet searching, and a Hopfield neural network based visualizer for database searching. Lessons learned and future directions are also presented.

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 [10]. While Internet services are popular and appealing to many online users, difficulties with searches are expected to worsen as the amount of online information increases. This is mainly due to the problems of information overload and vocabulary differences [7]. Many researchers consider that devising a scalable approach to Web searching is critical to the success of Internet and Intranet services, and other current and future National Information Infrastructure (NII) applications [17,5].

The complex, dynamic, and interconnected Web [21] essentially represents a large and evolving directed graph of connected nodes (homepages). General-purpose search algorithms developed in artificial intelligence, neural networks, and G-Search appear to be well suited for such a complex search task. As part of the ongoing Illinois Digital Library Initiative project, which aims to create a semantic retrieval and analysis environment [18], this research proposes an intelligent agent approach to creating Java-based search agents that can be deployed in database, Internet, Intranet, and digital library applications. After a review of intelligent agent research, we present several sample implementations of Java-based search agents in this paper.

2 Literature Review: Intelligent Agents and Internet/Intranet Searching

Our research is based on an intelligent agent approach to searching and relies significantly on several general-purpose search algorithms.

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 [16]. Such agents might act as “spiders” on the Web and look for relevant information [15], schedule meetings on behalf of executives based on their constraints, filter newsgroup articles based on “induced” (or learned) users’ profiles [11], or assist meeting facilitators in converging ideas [4]. Many researchers have focused on developing scripting and interfacing languages for designers and users such that they can create mobile agents of their own [19]. 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.”

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 [20], 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 within some environment, and act as a personal assistant to the user. In our research, our goal is to create intelligent agents that perform customized, dynamic searches in various Internet, Intranet, and digital library applications for any user.

2.2 Searching in Internet/Intranet

In order to allow our agents to be goal-oriented, cooperative, and customized, we have adopted various search paradigms.

Heuristic AI Search Algorithms: Broadly defined as heuristics-based search methods, artificial intelligence researchers have adopted different search algorithms in the context of human problem solving [13]. Based on the information space principle, human problem solving can be represented as a search in the information space via knowledge states and knowledge operators. Many serial search algorithms have been developed and implemented in various knowledge based systems, including breadth first search, hill climbing, depth first search, best first search, branch and bound, and A^* [13]. All of these serial search methods follow existing link structures based on either known distance/cost measures or some heuristics of expected distances or costs. For example, hill climbing explores the most promising neighbor first based on a predicted distance measure. After each exploration, predicted distances associated with new neighbors are computed to decide the next exploration point.

Most of the existing Internet spiders or databases are created based on this type of search methods. TueMosaic and the WebCrawler are two prominent early examples. Both use variations of conventional best first (local) search strategies [13]. DeBra and Post [7] reported on 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 the 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. At the Second WWW Conference, Pinkerton [14] reported on 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. Both systems were implemented in CGI/HTML.

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 exhaustive 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.

G-Search: G-Search is designed based on the genetic algorithms. G-Search is 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 [12]. G-Search inherits the features of exploration and exploitation of the search space. A crossover operator in the

G-Search is used to exploit local connected links and a mutation operator is used to create variations (random mutations) in the search space (i.e., exploration). A roulette wheel selection can be used to select promising candidates to survive the next iteration (generation) process, i.e., survival of the fittest (interestingness). Despite its global search capability, we did not find any advanced Internet or Intranet search applications employ such a method.

Neural Networks: Neural networks model computations in terms of complex topologies (neurons and synapses) and statistics-based error correction algorithms. Some neural networks exhibit excellent parallel search capabilities, most noticeably the Hopfield net [10].

A Hopfield net may provide a distributed representation for a content-addressable memory in which each structure is stored as a collection of active units. Such a special distributed representation allows the network to be more damage-resistant, a property that exists in animal memory. In addition to the distributed representation property, neural nets perform *parallel relaxation* searches, during which nodes are activated in parallel and are traversed until the network reaches a stable state. This process often is considered more efficient than serial, symbolic searching because it makes use of states that have no analogues in symbolic search and because it maps naturally onto highly parallel hardware [15]. Due to its novelty and computational complexity, researchers only recently started to experiment with these networks in large-scale Internet and Intranet applications [3].

In an attempt to create a dynamic and scalable infrastructure for various Internet, Intranet, and digital library applications [17], several general-purposed search algorithms were recently implemented in Java in our Illinois Digital Library Initiative testbed [18].

3 The Java Search Agent Workshop

The Java Search Agent Workshop (JS AW) was recently created to allow us to implement different search algorithms for various simple and complex Web-related search tasks.

Homepages with dynamic content have traditionally been produced by programs that use the Common Gateway Interface (CGI), a means of connecting client programs with information servers using the web protocol HTTP. However, the communication between the users' browsers and the HTTP servers is not dynamic. The servers are completely event driven, meaning they can only respond to requests from the clients and cannot initiate requests of their own. In addition, CGI programs can only respond to one event before terminating. These limitations lead to relatively static user interfaces for CGI based applications. This is especially frustrating when requests take

more than a few moments to satisfy, causing users to wonder if the program is still running. Also, users are relegated to passive roles during searches, unable to view intermediate results or change parameters in reaction to search events. Our earlier version of an Internet search agent was implemented in CGI/HTML and was severely hampered by such problems. For this reason, Java was chosen for our later search agent implementation.

Java is an object-oriented language that is both compiled and interpreted. The Java compiler converts source code to byte code that is platform independent [9]. This byte code represents instructions for the Java virtual machine which can be easily emulated on most modern hardware platforms. Ports currently exist for Windows NT, Windows 95, Macintosh, HP-UX, Solaris, Digital UNIX, Linux, and others. Our implementations were tested on HP-UX, Digital UNIX, and Windows NT.

Programmers have the option of creating two types of Java applications, those that can be run as stand-alone programs and those (applets) that can be executed in the framework of other software, such as a Web browser. In order to allow a wider access, most of our agents were implemented as applet. In addition, several of our agents were implemented in a client-server design.

In the following sections, several sample Java-based search agents are illustrated.

4 An Internet/Intranet Java Spider

Using the GA-based function searcher as our initial testbed, we proceeded to develop two Java Internet spiders based on best first search and G-Search. The function search problem was expanded to dynamic Internet searching, a complex, global search problem of similar nature.

An Internet search agent architecture was designed, which 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 the 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 homepages.

Users submit their search requests by providing several starting URLs and a few desired search keywords. The agents then perform searches from the starting 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). 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.

The graphical user interface (GUI) provides a link between the submitted task, control parameters, and the search engine. It includes forms, images, scrollbars, and radio buttons that the user employs to fill in the task input and control parameters. It also displays the search results using tables and graphics.

Our client-server architecture was based on UNIX sockets and created in both C and Java. 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 and control parameters, and displays intermediate and final results was written in Java. It can be invoked easily through a Netscape browser.

UNIX sockets are used to build 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. In order to reduce fetching time during the search process, which has 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 within the maximum allowable time.

In order to determine the “appropriateness” (or fitness) of a given new homepage, a Jaccard’s similarity function has been adopted. Each homepage is represented as a weighted vector of keywords and connecting links; the former are automatically extracted by our spider. 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 homepages 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, $X = (x_1, x_2, \dots, x_m)$ and $Y = (y_1, y_2, \dots, y_n)$, 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 [6]. 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:

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

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 equally-weighted summation of above two Jaccard's scores, i.e.,

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

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.

The search engine in our spider embeds both the best first search and a genetic algorithm. Best first search is a serial state space traversal method [13]. 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, $(input_1, input_2, \dots, input_m)$. 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 $H = (h_1, h_2, \dots, h_n)$.
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. 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.

A G-Search algorithm performs a stochastic evolution process toward global optimization based on the crossover and mutation operations in GA. The search space of the problems 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 in 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 G-Search 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 G-Search implementation. A sketch of the generic algorithm adopted for Web client-based searching is presented below:

1. *Initialize the search space.* The G-Search 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, $CG = (cg_1, cg_2, \dots, cg_m)$.
2. *Crossover.* A heuristics-based crossover operation is then used. New homepages connected to starting homepages in CG set are executed. Homepages that have been connected to multiple starting homepages (i.e., multiple parents) are considered Crossover Homepages and saved in a new set, $C = \{c_1, c_2, \dots\}$. 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 G-Search algorithm requests a mutated homepage, the system retrieves the top-ranked homepage from homepages in the user-specified category based in the keywords presented in the anchor 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. $M = \{m_1, m_2, \dots\}$.

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 [12]. 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 [12], homepages which ob-

tain 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.

Figure 1 shows the Java request and control parameter interface. When the genetic algorithm spider is selected, the system displays a dialog block soliciting starting URLs, desired keywords, and search categories (a mutation seed database). In addition, users can set their preferred crossover and mutation probabilities and the desired number of URLs. A timeout mechanism was also introduced to avoid time-consuming, unfruitful connections.

Figure 2 shows the window which displays the intermediate and final search results dynamically and graphically. The 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 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 2, the fetched URLs are displayed during each generation (instead of 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 the genetic algorithm converged) the spider falls asleep, as shown by sleeping spider animation (figure 3).

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.

5 A Neural Network Agent for Vocabulary Switching

Designed for digital library and Intranet applications, we recently developed a Hopfield neural network based Java agent to perform vocabulary switching – a classical problem for searching databases of different subject domains (vocabularies) [2]. Many researchers consider vocabulary switching to be the “holy grail” of information retrieval [17].

The screenshot shows a Java applet window titled "Global Evolution Based Search Input Applet/Application". It contains the following controls:

- Starting URLs:** A list of text boxes containing "http://ai.bpa.arizona.edu", "http://", "http://", "http://", "http://", and "http://".
- Crossover and Mutation Rates:** Two text boxes, both containing "0.5".
- Timeout (seconds):** A text box containing "20.0".
- Starting Keywords:** A text box containing "Digital Library".
- # of URLs return:** A text box containing "10.0".
- Categories:** A grid of radio buttons for selecting a category:
 - Arts, Business, ☒ Computer
 - Education, Entertainment, Government
 - Health, News, Recreation
 - Reference, Regional, Science
 - Social Culture, Social Science
- Buttons:** Submit, Clear, Default, New Search, and Exit.

Fig. 1. The control panel for initiating a Java-based genetic algorithm spider.

The program implemented in our testbed uses the same client-server approach with a C/C++ based server and a Java client. Communication is performed using UNIX sockets and token passing similar to the Internet spider. The same threaded communication class is used to manage the connection. The server performs vocabulary switching using the neural network and passes the results to the client which creates the display.

Fig. 2. The window shows the intermediate result of the search process displayed dynamically. Animation is displayed at the upper right-hand corner. The 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.

The Hopfield network is a neural net that can be used as a content-addressable memory. Since knowledge can be stored in a single-layered interconnected neurons (nodes) and weighted synapses (links), information can be retrieved based on the network's parallel relaxation process. Nodes are activated in parallel and activation values from different sources are combined for each individual node (see figure 4) until the activation levels of nodes on the network reach a stable state (convergence).

The formulas in figure 5 show the parallel relaxation property of the Hopfield net. At each iteration, nodes are activated in parallel and activated values from different sources are combined for each individual node. Neighboring nodes are traversed in order until the activation levels of nodes on the network gradually "die out" and the network reaches a stable state (convergence). The weight computation schema ($net_j = \sum_{i=0}^{n-1} t_{ij} u_i(t)$) is unique to the Hopfield net algorithm. Each newly activated node computes its new

Fig. 3. The display window shows the final search result. A sleeping spider indicates the end of a search session.

Fig. 4. Hopfield net parallel relaxation process.

weight based on the summation of the products of its neighboring nodes' weights and the similarity of its predecessor node to itself.

$$u_i(t) = x_i \quad 0 \leq i \leq n-1$$

$u_i(t)$ is the output of node i at time t . x_i (which has a value between 0 and 1) indicates the input pattern for node i .

$$u_j(t+1) = f_s \left[\sum_{i=0}^{n-1} t_{ij} u_i(t) \right] \quad 0 \leq j \leq n-1$$

where f_s is the continuous SIGMOID transformation function as shown below:

$$f_s(net_j) = \frac{1}{1 + \exp\left[\frac{-(net_j - \vartheta_j)}{\vartheta_0}\right]}$$

where $net_j = \sum_{i=0}^{n-1} t_{ij} u_i(t)$, and ϑ_j serves as a threshold or bias, and ϑ_0 is used to modify the shape of SIGMOID function.

Fig. 5. Hopfield net parallel relaxation formulas.

The above process is repeated until there is no significant change in terms of output between two iterations, which is accomplished by checking the following formula:

$$\sum_{j=0}^{n-1} |u_j(t+1) - u_j(t)| \leq \varepsilon$$

where ε is the maximum allowable error (used to indicate whether there is a significant difference between two iterations). Once the network converges, the output represents the set of terms most relevant to the starting input terms.

Using a parallel relaxation search process, the Hopfield network agent invokes selected knowledge networks in different databases. As shown in figure 6, a user selects Dam database as the starting domain, the Surface Water database as the target domain for vocabulary switching, and enters "Hydraulic Power" and "Foundation Soils" as a search term in an HTML form. That is, the user wishes to switch the Dam term "Hydraulic Power" and "Foundation Soils" to conceptually similar or equivalent terms in Surface Water. Such cross-domain vocabulary switching is common in research collaboration and in scientific retrieval. Using the starting term, the Java client

Fig. 6. The starting screen for the vocabulary switching program

starts and makes a connection to the server which implements the actual Hopfield net search process. As shown in figure 6, search results are shown as a network graph with starting terms ("hydraulic power" and "foundation soils") represented as nodes at the far left. Intermediate and the final results found in the destination domains are shown at the extreme right. The edges (white lines) connecting the nodes show the paths between initial and final terms. Neighboring terms are identified by looking up concept spaces (automatic thesauri) created earlier in our research [6]. Color is used to show the domains in which terms are found. Nodes are green if the term is only in the starting domains, yellow if only in the destination domains, and purple if in both. An effective vocabulary switching session is demonstrated by a search with starting terms in the starting domains and final terms in the target domains.

The Java display is very powerful, allowing users to see the parents and children of any term in the network by clicking on its node. Blue edges indicate connections to parent terms and purple shows the paths to children. Figure 7 shows a selected term with parent and child connections displayed. In addition, areas can be zoomed by clicking the mouse button on a particu-

lar node. The node will be enlarged and the rest of the nodes will be shrunk as in figure 8. Tasks such as this are very difficult for CGI because the server would have to create bitmapped images and send them to the client browser. By using Java, communication with the server is kept to a minimum, thereby decreasing the load on the server and communication network. Also, the response time to user input is vastly improved creating a more satisfactory search experience.

Fig.7. Parent and child paths for the Hopfield neural network agent.

6 Conclusion and Discussion

The results from our current experimentation of various Java-based search agents are promising. With the dynamic nature of Java, we were able to implement various generic search paradigms and methods, including best first search, genetic algorithms, and neural networks, for different Internet, Intranet, and digital library search problems. We believe we are moving closer towards our goal of creating search agents which are integrated,

expressive, goal-oriented, cooperative, and customized. Readers are encouraged to access our Java server for demonstrations of all the above agents (<http://ai.bpa.arizona.edu>).

In our ongoing effort in the Illinois Digital Library Initiative project, we are in the process of including other general-purpose search methods and visualization techniques in our Java Search Agent Workshop. In particular, a fisheye view and a fractal based visualization component are under development for viewing the agent search results. In addition, we are in the process of incorporating our search agents in an engineering digital library testbed (the Hopfield net agent) and in experimental big-ten university engineering Intranet (i.e., the Internet/Intranet spider).

Fig. 8. The zoom view for the vocabulary switching program.

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);
- National Center for Supercomputing Applications (NCSA), High- performance Computing Resources Grants, 1994-1996 (H. Chen).

We would like to thank University of Arizona Artificial Intelligence Group members for their participation in our experiments.

References

1. Berners-Lee T., Cailliau R., Luotonen A., Nielsen H. F., and Secret A. (1994) The World-Wide Web. *Communications of the ACM*, 37(8): 76-82.
2. Chen H. (1994) Collaborative systems: solving the vocabulary problem. *IEEE Computer*, 27(5): 58-66, Special Issue on Computer-Supported Cooperative Work (CSCW).
3. Chen H. (1995) Machine learning for information retrieval: neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3): 194-216.
4. Chen H., Houston A., Yen J. and Nunamaker J. F. (1996) Toward intelligent meeting agents. *IEEE Computer*, 29(8): 62-70.
5. Chen H. and Schatz B. R. (1994) Semantic retrieval for the NCSA Mosaic. In *Proceedings of the Second International World Wide Web Conference 94*, Chicago, IL, October 17-20.
6. Chen H., Schatz B. R., Ng T. D., Martinez J. P., Kirchhoff A. J., and Lin C. (1996) 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.
7. DeBra P. and 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.
8. Etzioni O., and Weld D. (1994) A softbot-based interface to the Internet. *Communications of the ACM*, 37(7): 72-79.
9. Flanagan D. (1996) *Java in a Nutshell*. O'Reilly Associates, Inc., CA.
10. Hopfield J. J. (1982) Neural network and physical systems with collective computational abilities. *Proceedings of the National Academy of Science, USA*, 79(4): 2554-2558.
11. Maes P. (1994) Agents that reduce work and information overload. *Communications of the ACM*, 37(7): 30-40.
12. Michalewicz Z. (1992) *Genetic Algorithms + Data Structures = Evolution Programs*. Springer-Verlag, Berlin Heidelberg.
13. Pearl J. (1984) *Heuristics: Intelligent Search Strategies for Computer Problem Solving*. Addison-Wesely Publishing Company, Reading, MA.

H. Chen, M. Ramsey and P. Li

14. Pinkerton B. (1994) Finding what people want: experiences with the WebCrawler. In *Proceedings of the Second International World Wide Web Conference 94*, Chicago, IL, October 17-20.
15. Rich E. and Knight K. (1991) *Artificial Intelligence*, 2nd Edition. McGraw-Hill, Inc., New York, N.Y.
16. Riecken D. (1994) Intelligent agents. *Communications of the ACM*, 37(7): 18-21.
17. Schatz B. R., and Chen H. (1996) Building large-scale digital libraries. *IEEE Computer*, 29(5): 22-27.
18. Schatz B. R., Micho B., Cole T., Hardin A., Bishop A., and Chen H. (1996) Federating repositories of scientific literature. *IEEE Computer*, 29(5): 28-36.
19. Waldrop M. M. (1994) Software agents prepare to sift the riches of cyberspace. *Science*, 265:882-883, 12.
20. Weld D. (1995) The role of intelligent systems in the national information infrastructure. *AI Magazine*, pages 45-64.
21. Yourdan E. (1996) Java, the Web, and software development. *Communications of the ACM*, pages 25-30.