Machine Learning, 39, 203–242, 2000.

Adaptive Retrieval Agents: Internalizing Local Context and Scaling up to the Web

FILIPPO MENCZER filippo-menczer@uiowa.edu
Management Sciences Department, University of Iowa, Iowa City, IA 52242-1000, USA

RICHARD K. BELEW rik@cs.ucsd.edu
Computer Science and Engineering Department, University of California San Diego, La Jolla, CA 92093-0114, USA

Editors: Jaime Carbonell and Yiming Yang

Abstract. This paper discusses a novel distributed adaptive algorithm and representation used to construct populations of adaptive Web agents. These *InfoSpiders* browse networked information environments on-line in search of pages relevant to the user, by traversing hyperlinks in an autonomous and intelligent fashion. Each agent adapts to the spatial and temporal regularities of its local context thanks to a combination of machine learning techniques inspired by ecological models: evolutionary adaptation with local selection, reinforcement learning and selective query expansion by internalization of environmental signals, and optional relevance feedback. We evaluate the feasibility and performance of these methods in three domains: a general class of artificial graph environments, a controlled subset of the Web, and (preliminarly) the full Web. Our results suggest that InfoSpiders could take advantage of the starting points provided by search engines, based on global word statistics, and then use linkage topology to guide their search on-line. We show how this approach can complement the current state of the art, especially with respect to the scalability challenge.

Keywords: InfoSpiders, distributed information retrieval, evolutionary algorithms, local selection, internalization, reinforcement learning, neural networks, relevance feedback, linkage topology, scalability, selective query expansion, adaptive on-line Web agents

1. Introduction

As information environments become more complex, there is a need for tools that assist users in ways that scale with the growth of such environments and adapt to both the personal preferences of the user and the changes in user and environmental conditions. The situation is not unlike the one faced by organisms adapting in physical environments. Such natural agents have to adapt to the topology of their environments, internalizing into their behaviors (via evolution or learning during life) those environmental signals and cues that they perceive as leading to their well-being. The environmental features that are best correlated (or anti-correlated) with fitness are the most useful internalizations.

Consider for example an ant's environment: its association between pheromone and food is internalized into the ant's pheromone-following behavior. Further, this behavior can be implemented by any ant without need for centralized control; finally, the ant can still

resort to a behavior, such as a random walk, that can be implemented in the absence of any appropriate environmental signals. These capabilities of an ant colony—internalization of environmental signals, distributed control, and integration of externally driven and endogenous behaviors—are also highly desirable properties for agents carrying out tasks in complex information environments. It turns out, in fact, that such local, agent-based strategies provide an excellent complement to initial searches seeded from the global perspective of Web search engines.

This paper focuses on several machine learning abstractions springing from ecological models, and on their applications to the intelligent retrieval of information distributed across networked environments. In Section 2 we discuss some of the limitations of the current state of the art; in Section 3 we suggest ways in which these limitations can be overcome by employing adaptive on-line agents. These ideas are first tested on a graph abstraction of the information discovery problem, in Section 4. Their application to the real problem of locating information on the Web and the implementation of actual Web retrieval agents required many design decisions, which are illustrated in Section 5. The system is evaluated in Section 6. Section 7 discusses the main contributions of this work, contrasting them with other approaches in the domains of machine learning and information retrieval. Section 8 outlines some directions for future research.

2. Beyond the state of the art

Exploiting proven techniques from information retrieval, search engines have followed the growth of the Web and provided users with much needed assistance in their attempts to locate an retrieve information from the Web (Eichmann, 1994; Pinkerton, 1994). The success of search engines is attested by both their proliferation and popularity.

The model behind search engines draws efficiency by processing the information in some collection of documents once, producing an *index*, and then amortizing the cost of such processing over a large number of queries which access the same index. The index is basically an inverted file that maps each word in the collection to the set of documents containing that word. Additional processing is normally involved by performance-improving steps such as the removal of noise words, the conflation of words via stemming and/or the use of thesauri, and the use of word weighting schemes (Belew, forthcoming).

This model, which is the source of search engines' success, is also in our opinion the cause of their limitations. In fact it assumes that the collection is static, as was the case for earlier information retrieval systems. In the case of the Web, the collection is highly dynamic, with new documents being added, deleted, changed, and moved all the time. Indexes are thus reduced to "snapshots" of the Web. At any given time an index will be somewhat inaccurate (e.g., contain stale information) and somewhat incomplete (e.g., missing recent information).

These problems, compounded by the huge size of the Web, hinder search engines' capability to satisfy user queries. Users are often faced with very large hit lists, yet low *recall* (fraction of relevant pages that are retrieved), and stale information. These factors make it necessary for users to invest significant time in manually browsing the neighborhoods of (some subset of) the hit list.

So our question is: *How can we extend traditional information retrieval techniques to address such limitation?* The answer we suggest here is that populations of intelligent, autonomous, adaptive information agents can effectively deal with these problems by automating much of the work that currently must be done by users. In the following subsections we discuss two ways in which agents can improve on the current state of the art.

2.1. Linkage topology

Indexing can be described as the process of building a topology over a document space, based on the distribution of words in the space. In the vector space model (Salton & McGill, 1983), documents and queries are viewed as vectors in very large feature spaces where each word corresponds to a dimension. Two documents are similar if the angle between their respective vectors is small. A search engine will show similar documents next to each other, effectively creating on the fly a topology based on their word statistics. This is a very useful model because the user can immediately make assumptions about the contents of retrieved documents, for example about the fact that they contain certain words.

However, hypertext information environments such as the Web contain additional structure information (Chakrabarti et al., 1998b). While this linkage information could be used to provide browsing users (or agents) with helpful cues, one cannot submit to search engines queries like "Give me all documents k links away from this one," because the space to store such information, or the time to compute it on the fly, would scale exponentially with k.[1]

While much linkage information is lost in the construction of indexes, it is there to be exploited by browsing users, who in fact navigate from document to document following links. We argue that linkage topology—the spatial structure in which two documents are as far from each other as the number of links that must be traversed to go from one to the other—is indeed a very precious asset on the Web. Even in unstructured portions of the Web, authors tend to cluster documents about related topics by letting them point to each other via links. Such linkage topology is useful inasmuch as browsers have a better-than-random expectation that following links can provide them with guidance. If this were not the case, browsing would be a waste of time.

Let us quantify the notion of value added by linkage topology. We have conjectured that such value can be captured by the extent to which linkage topology "preserves" relevance (with respect to some query). To obtain a lower bound for the value added of linkage topology, imagine a browsing user or agent following a random walk strategy. Define R as the conditional probability that following a random link from a relevant document will lead to another relevant document. We call R *relevance autocorrelation.* And define G as the fraction of relevant documents. We call G *generality* (of the query). If linkage topology has any value for the random browser, then browsing will lead to relevant documents with higher than random frequency. In order for this to occur the inequality

$$\frac{R}{G} = \frac{\Pr[rel(D_2) \mid rel(D_1) \wedge link(D_1, D_2)]}{\Pr[rel(D_2)]} > 1$$

must hold. We can then express the linkage topology value added by the quantity $\Theta = R/G - 1$.

Table 1. Measures of Θ for ten queries submitted to Lycos (lycos.http://www.lycos.com) and Britannica Online (Encyclopaedia Britannica, Inc. http://www.eb.com). The minimum score parameter used for Lycos was 0.1. Only full (absolute) HTTP references were considered for Lycos, and only Micropaedia and Macropaedia article references for EB. Multiple-term queries represent AND Boolean searches.

Query	Θ (Lycos)	Θ (Britannica Online)
`red wine`	2.5×10^4	3.7×10^1
`evolution selection`	2.3×10^3	1.8×10^1
`photography`	3.5×10^3	1.8×10^2
`color`	2.8×10^3	6.1×10^3
`blindness`	2.1×10^4	1.1×10^2
`einstein`	5.2×10^3	6.2×10^1
`bach`	7.5×10^3	7.8×10^1
`carcinoma`	1.5×10^4	1.3×10^2
`cinema`	1.1×10^3	1.0×10^2
`internet`	4.0×10^3	7.2×10^1
Mean	$(9 \pm 3) \times 10^3$	$(7 \pm 6) \times 10^2$

As a reality check, we have measured Θ for a few queries from a couple of search engines (Menczer, 1997). Relevance autocorrelation statistics were collected by counting the fraction of links, from documents in each retrieved set, pointing back to documents in the set. Generality statistics were collected by normalizing the size of the retrieved sets by the size of the collections. These are quite gross measurements, since they assume a correspondence between retrieved and relevant sets.[2] Our conjecture about the value added by linkage topology is confirmed by the large values of Θ shown in Table 1. Independent evidence for the value of linkage topology can be found in bibliometric studies of the Web (Larson, 1996) as well as link-based approaches to Web page categorization or discovery (Chakrabarti et al., 1998c).

If links constitute useful cues for navigation—even for random walkers—then they can be exploited by autonomous *browsing* agents as they are by browsing users. This observation suggests that browsing is not an unreasonable task for autonomous agents.

2.2. Scalability

Scalability is a major issue limiting the effectiveness of search engines. The factors contributing to the problem are the large size of the Web, its rapid growth, and its highly dynamic nature. In order to keep indexes up-to-date, *crawlers* periodically revisit every indexed document to see what has have been changed, moved, or deleted. Heuristics are used to estimate how frequently a document is changed and needs to be revisited, but the accuracy of such statistics is highly volatile. Moreover, crawlers attempt to find newly added documents either exhaustively or based on user-supplied URLs. Yet Lawrence and Giles have shown that the coverage achieved by search engines is at best around 33%, and that coverage is anti-correlated with currency—the more complete an index, the more stale

links (Lawrence & Giles, 1998). More importantly, such disappointing performance comes at high costs in terms of the load imposed on the net (Eichmann, 1994).

The network load scales as n/τ, where n is the number of documents in the Web and τ if the time scale of the index, i.e. the mean time between visits to the same document. The longer τ, the more stale information in the index. If q is the number of queries answered by the search engine per unit time, then the amortized cost of a query scales as $n/q\tau$.

Agents searching the Web *on-line* do not have a scale problem because they search through the *current* environment and therefore do not run into stale information. On the other hand, they are of course less efficient than search engines because they cannot amortize the cost of a search over many queries.[3] Assuming that users may be willing to cope with the longer wait for certain queries that search engines cannot answer satisfactorily, one might ask, *What is the impact of on-line search agents on network load?*

We argue that because of the scale effect, making an index less up-to-date can free up sufficient network resources to completely absorb the impact of on-line searches. Consider increasing the τ of a search engine by a factor of $(1+\epsilon)$, allowing the information in the index to become correspondingly more stale. Maintaining a constant amortized cost per query, we could now refine the results of each query with an on-line search using an amount of network resources scaling as

$$\frac{n}{q\tau} - \frac{n}{q\tau(1+\epsilon)} \sim \frac{n}{q\tau}\frac{\epsilon}{1+\epsilon}.$$

As an example, imagine visiting 100 Web pages on-line for each query, and accepting $\epsilon = 1$ (bringing τ, say, from one to two weeks). This could be achieved without impacting network load by satisfying the condition $n/q\tau = 200$. Assuming $q\tau$ (the number of queries posed over a constant time interval) is a constant, the current growth of the Web assures that the condition will be met very soon.[4] This simple argument, in our opinion, shifts the question: we should not ask what is the network impact of on-line search agents, but rather, *What ϵ achieves an appropriate balance between the network loads imposed by search engines crawlers and on-line agents.*

3. On-line search agents

Let us operationalize the ideas discussed in the previous section. Our goal is to address the scalability limitation of search engines by an adaptive framework able of taking advantage of both the word and linkage topology of the networked information environment. Our approach is based on the idea of a multi-agent system. The problem is decomposed into simpler subproblems, each addressed by one of many simple agents performing simple operations. The divide-and-conquer philosophy drives this view. Each agent will "live" browsing from document to document on-line, making autonomous decisions about which links to follow, and adjusting its strategy to both local context and the personal preferences of the user. Population-wide dynamics will bias the search toward more promising areas.

In this framework both individual agents and populations must adapt. Individually learned solutions (e.g., by reinforcement learning) cannot capture global features about the search space or the user. They cannot "cover" heterogeneous solutions without complicated internal

models of the environment; such models would make the learning problem more difficult. On the other hand, if we allowed for population-based adaptation alone (e.g., by an evolutionary algorithm), the system would be prone to premature convergence. Genetically evolved solutions might also reflect an inappropriate coarseness of scale, due to individual agents' incapability to learn during their life. These are the same reasons that have motivated the hybridization of genetic algorithms with local search (Hart & Belew, 1996), and reflect the general problem of machine learning techniques in environments with very large feature space dimensionalities.

3.1. Local selection

How long should an agent live before being evaluated? What global decisions can be made about which agents should die and which should reproduce, in order to bias the search optimally? No answer to these questions appears satisfactory. Fortunately, algorithms motivated by the modeling of populations of organisms adapting in natural environments provide us with ways to remain agnostic about these questions (Menczer & Belew, 1996). Such an algorithm is illustrated in figure 1.

In step (1), each agent in the population is initialized with some random search behavior and an initial reservoir of "energy." If the algorithm is implemented sequentially, parallel execution of agents can be simulated with randomization of call order. Steps (4) and (5) depend on the sensory and motor apparatus of an agent. In general we can assume that actions will incur in work (energy loss) and possibly energy intake. The internal energy reservoir is updated accordingly in step (6). Interactions with the environment may also result in environmental cues that can be used as reward or penalty signals and exploited by reinforcement learning in step (7), so that an agent can adapt during its lifetime.

Steps (8–12) are the central point where this algorithm differs from most other evolutionary algorithms. Here an agent may be killed or be selected for reproduction. Energy is conserved in both events. The selection threshold θ is a constant independent of the rest of the population—hence selection is *local*. This fact reduces communication among agent

```
1.  initialize population of agents, each with some energy E
2.  loop:
3.      foreach alive agent:
4.          get and process sensory input
5.          produce motor action
6.          update energy and environmental state
7.          optionally learn by reinforcement
8.          if (E ≥ θ)
9.              reproduce
10.             split energy with offspring
11.         elsif (E ≤ 0)
12.             die
13.     replenish environmental resources
```

Figure 1. Pseudocode of an evolutionary algorithm based on local selection.

processes to a minimum and has several important consequences. First, agents compete only if they are situated in portions of the environment where they have to share resources. No centralized decision must be made about how long an agent should live, how frequently it should reproduce, or when it should die. The search is biased directly by the environment. The size of the population, rather than being determined a priori, emerge from the *carrying capacity* of the environment. This is determined by the replenishment of resources that occurs in step (13) and is independent of the population.

Second, environment-driven adaptation makes the search algorithm more amenable to *cover* optimization—all good solutions being represented in the population—than to standard convergence criteria. The bias is to exploit all resources in the environment, rather than to locate the single best resource. This is particularly appropriate in multi-criteria optimization applications, such as information search, where the goal is to locate as many relevant sources of information as possible.

Finally, the algorithm is embarrassingly parallelizable and therefore lends itself ideally to distributed implementations, in which case agents can execute on remote servers. If servers are available as computational resources, the algorithm can achieve a speedup proportional to the size of the population.

Local selection of course has disadvantages and limitations as well. Imagine a population of agents who can execute code on remote servers in a distributed information environment, but have to look up data on the client machine for every page they access. A typical example of such a situation would be a centralized page cache. Because of communication overhead and synchronization issues, the parallel speedup achievable in this case would be seriously hindered. As this scenario indicates, the feasibility of distributed implementations of evolutionary algorithms based on local selection requires that agents be provided by their execution environment with access to local storage. Such capabilities may require the addressing of such system-related issues as security and service payment transactions.

3.2. Internalization

Another limitation of local selection is its weak selection pressure. Consider the energy level throughout the lifetime of the hypothetical agent depicted in figure 2. The agent reproduces around time 35, giving half of its energy to the offspring. Finally the agent runs out of energy and dies shortly after time 90. Such selection events are rare. As long as the energy level fluctuates between 0 and θ, there is no selective pressure.

Reinforcement signals from the environment that are correlated with an agent's performance can be used as reward or penalty signals to adjust the agent's behavior during its life. This is the basis of the *reinforcement learning* framework (Pack Kaelbling, Littman, & Moore, 1996). In particular, if the sensors of the agent in figure 2 could pick up the signal encoding instantaneous changes in its energy level, the agent could clearly use that information to assess the appropriateness of its actions. Such signal corresponds to the energy change computed in step (6) of the algorithm in figure 1. It could be computed, for example, as the time derivative of the agent's energy level. Its sign represent a perfect reward/penalty cue from the environment. Any reinforcement learning scheme can be used to adjust the behavior of the agent so that actions perceived to lead to rewards are reinforced, and action

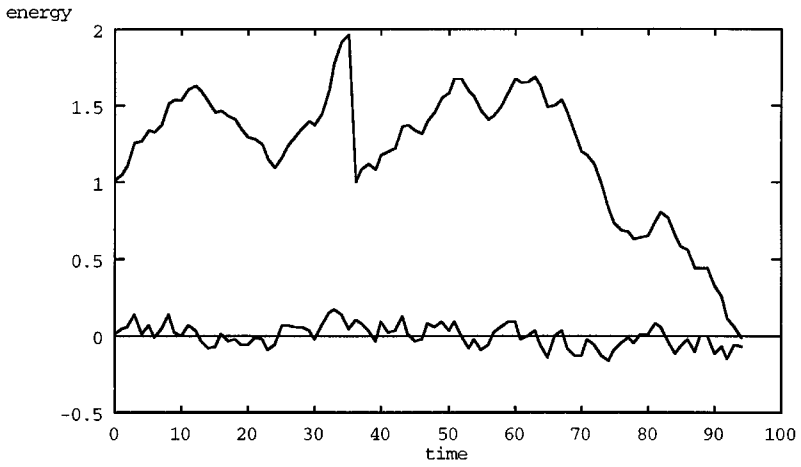

Figure 2. Illustration of energy dynamics in a typical agent's lifetime. The higher curve plots the level of accumulated energy as a function of time, resulting from the instantaneous changes in energy plotted by the lower curve. The selection threshold is $\theta = 2$. With the exception of the reproduction event, the energy level is the integral of the lower curve.

perceived to lead to penalties are discouraged. For example, the prevailing penalties incurred between time 65 and 90 might have warranted changes leading to a delayed death.

An effective *internalization* of environmental signals via reinforcement learning during the lifetime of an agent can obviate to the problem of weak local selective pressure. But more generally, it can allow agents to pick up cues that are just not detectable at the time scales of evolutionary adaptation. Evolution by local selection and internalization by reinforcement learning are two sides of the same coin. Each process uses information from the environment to improve on its "unit of adaptation." For reinforcement learning, the unit is the agent and the signals are sampled instantaneously. For evolution, the unit is the population and the signals are averaged over time to eliminate noise and inconsistent cues. The two forms of unsupervised adaptation can be integrated together to cover the different temporal and spatial scales at which useful cues can be detected.

On-line agents performing in complex environment, such as networked information environment, have access to many signals. The problem is not so much whether some of these signals contain useful information, but rather how to identify those among the many cues that best correlate with performance and may allow the agent to discriminate between good and bad behaviors. We will discuss in Section 5 how information agents can deal with this problem.

4. Searching random graphs

In this section we outline a number of experiments aimed at evaluating the algorithm discussed above in an abstract class of environments. The problem can be broadly described

as searching large graphs in sublinear time. Imagine a very large graph, where each node is associated with some payoff. The population of agents visits the graph as agents traverse its edges. The goal is to maximize the collective payoff of visited nodes, given that there is only time to visit a fraction of the nodes in the graph. The framework is well studied theoretically in the case of random-walkers. The problem of finding the optimal path to an unknown node in a weighted graph is NP-complete (Koutsoupias, Papadimitriou, & Yannakakis, 1996). Agents can do no better than heuristically searching on-line through the graph.

The problem is interesting because typically the graph is distributed, so that agents are charged costs for using its resources, e.g., traversing edges and evaluating nodes' payoff. More importantly, the graph search task is general enough that it can be reduced to some interesting special cases. E.g., if we use nodes to model hypertext documents, edges for hyperlinks, and payoff for some measure of relevance, then the problem can be used to simulate networked information retrieval; we can explore different search strategies in artificial information environments, given a model of relevance.[5]

In our instances of the graph search task, each node is assigned a payoff p from a uniform probability distribution in the unit interval. Furthermore, each link l is annotated with a "feature vector" with N_f real components $f_1^l, \ldots, f_{N_f}^l \in [0, 1]$. These features are generated from an *ad-hoc* distribution to model, e.g., textual annotations around anchors. If properly interpreted, the features of a link can allow an agent to predict the payoff of the node pointed to by that link.

To make this possible, each agent's genotype comprises a single-layer neural net or perceptron, i.e., a vector of weights $w_1, \ldots, w_{N_f+1} \in \Re$. An agent receives in input, for each outgoing link from the node where it is currently situated, the link's feature vector (step (4) of the algorithm in figure 1). It then uses its neural net to compute

$$o(l) = \frac{1}{1 + \exp\left\{-\left(w_{N_f+1} + \sum_{i=1}^{N_f} w_i f_i^l\right)\right\}}$$

i.e., its prediction of the payoff $p(l)$ of the node that l points to. Finally (step (5)) the agent follows a link that is picked by a stochastic selector among the links from the current node, with probability distribution

$$\Pr[l] = \frac{e^{\beta o(l)}}{\sum_{l' \in node} e^{\beta o(l')}}$$

where the β parameter is a component of the agent's genotype describing the importance attributed to link predictions.

The feature vectors are constructed in such a way that there exists an optimal perceptron weight vector that predicts payoff within accuracy A (a parameter). Agents with such a genotype can follow the best links and thus achieve optimal fitness (maximum payoff intake). The energy benefit of an action is the payoff of the newly visited node, provided it had

not been previously visited by any agent (step (6)). A node yields energy only once; nodes are "marked" to keep track of used resources and the environment is not replenished. A constant energy cost is charged for any new node visited. A smaller cost is also charged for previously visited nodes, to prevent endless paths through visited nodes. At reproduction (steps (8–10)), an agent's genotype is cloned and mutated to obtain the offspring genotype. Both β and some of the weights are mutated by additive uniform noise (with the constraint $\beta \geq 0$). To study the effect of local selection in isolation from other factors, no recombination operator is applied.

Random graphs are generated according to a number of distinct parameterizations, aimed at modeling different aspects of a networked information environment with respect to some query:

generality G equals the density of "relevant" nodes, i.e. those whose payoff is above some threshold (cf. Section 2 and carrying capacity in Section 3.1);

ambiguity H is the number of clusters in which relevant nodes are grouped, each with a distinct optimal perceptron (the irrelevant background has yet another optimal weight vector);

autocorrelation R is defined as the conditional probability that a relevant node is linked to other nodes in the same cluster ($R \geq G$; cf. relevance autocorrelation in Section 2);

accuracy $A = 1 - noise$, where *noise* is the minimum achievable error in payoff prediction (by construction).

Unless otherwise stated, the graphs constructed for the experiments described in this section have 1000 nodes, an average fan-out of 5, and $N_f = 16$ features constructed with an accuracy $A = 0.99$. β is initialized with uniform distribution in the range $[0, 6]$.

4.1. Local versus global selection

We have first used the graph environments to compare local and global selection (Menczer & Belew, 1998b). Binary deterministic tournament selection was chosen as the global scheme for the comparison because of its steady-state nature. Steps (8) and (11) of the algorithm are modified for tournament selection by using the energy level of a randomly chosen member of the population in place of both θ for reproduction and 0 for death.

The algorithm is stopped when 50% of the nodes have been visited. Figure 3 illustrates the difference in performance typically observed between the two selection schemes for a few example graphs. The *recall* level (fraction of relevant nodes visited so far) is plotted versus the fraction of all nodes visited so far. Local selection populations continue to discover a constant rate of good nodes, while tournament populations tend to converge prematurely.

The same experiment was repeated for a wide range of graph parameters:

$$(G, R, H \mid G \in \{0.025, 0.05, 0.1, 0.2\},\ R \in \{0.2, 0.4, 0.6, 0.8\},\ H \in \{1, 2, 3, 4\})\,.$$

Across all graph parameterizations, local selection significantly and consistently outperforms tournament selection. The improvement varies depending on the graph parameters, but is generally between two- and ten-fold.

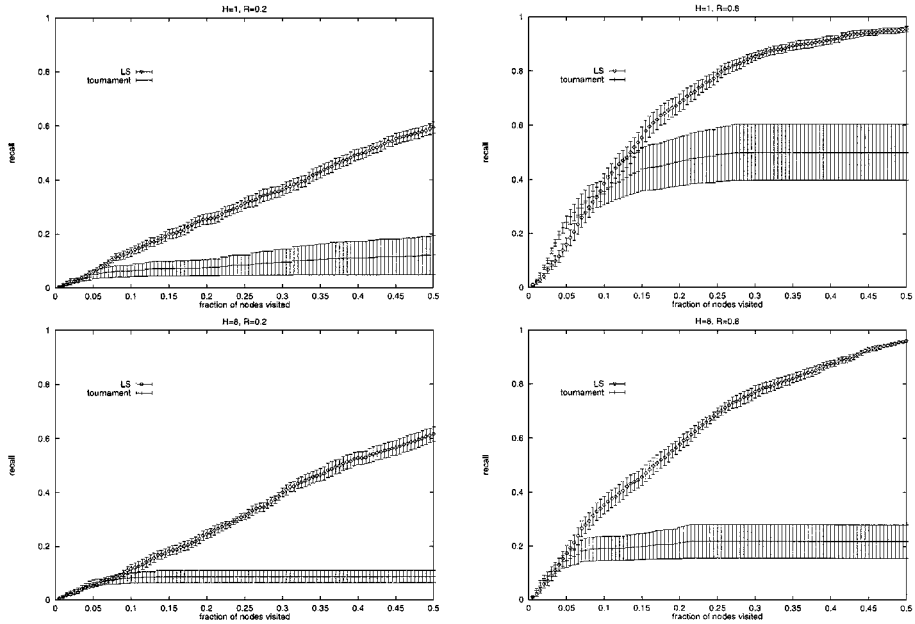

Figure 3. Performance of local selection (LS) vs. tournament selection on typical random graphs with $G = 0.1$ and various values of H and R (shown above the plots). In these and the following plots, error bars indicate standard errors across multiple runs with the same algorithm and graph parameters.

Two main trends are observed in this data. Increasing R, the correlation among good nodes, is equivalent to increasing the importance of locality; where an agent is situated has greater consequence in determining how well it will do in the future. In agreement with our expectation, we find the performance of local selection to increase with R at least as much as that of tournament selection, yielding a consistent advantage in favor of local selection. Increasing H makes for a multi-criteria problem, modeling ambiguous queries. Correspondingly, we observe that tournament selection degrades in performance due to premature convergence. Since local selection is not prone to premature convergence, the advantage in favor of local selection increases with H. These results tell us that if we apply an evolutionary algorithm to search the Web, we are justified in using a robust selection method such as LS, especially in the face of linkage topology (modeled by R) and ambiguous queries (modeled by H).

4.2. Internalization of global cues

In a second set of experiments, the goal was to test the capability of agents evolving by the local selection algorithm to internalize global environmental cues (Menczer, 1997). The signal considered was the accuracy of payoff predictions based on link cues, i.e., the potential accuracy of optimally evolved agents. For high A, the optimal agent strategy is best-first-search; for low A, it is random-walk. Thus internalization of link prediction accuracy implies

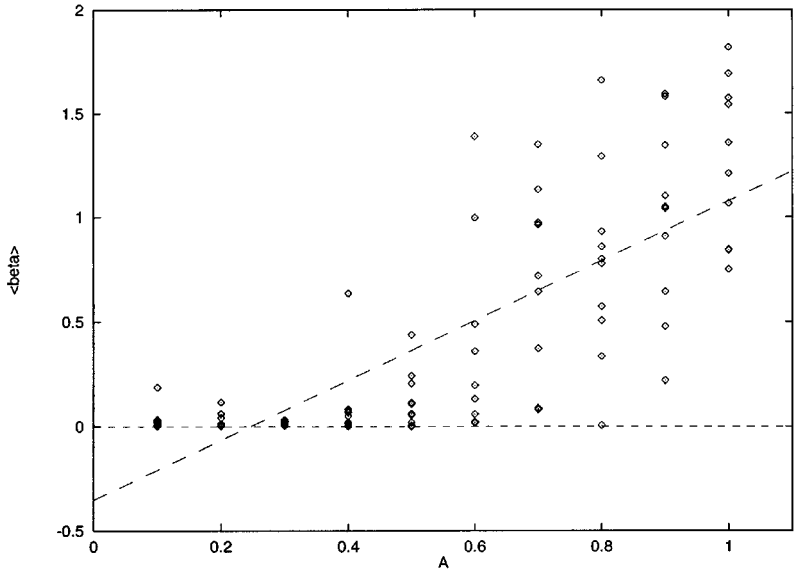

Figure 4. Scatter plot between link accuracy A and population averages of evolved β parameters. Linear regression is also shown.

evolving β values corresponding to the appropriate strategies as implemented by agents via their stochastic selectors.

We have run ten experiments with graphs having an average fan-out of 10 links, $G = 0.1$, $R = 0.75$, and $H = 1$. Each experiment, consisting of 10 runs, used a different value of A between 0.1 (very noisy predictions) and 1.0 (perfectly accurate predictions). In each of these runs, β was initialized with uniform distribution in the range $[0, 5]$ and measured after 750 node accesses. As figure 4 shows, the β values evolved by the population are indeed well correlated with the accuracy of the environmental cues. The correlation coefficient is 0.77. This indicates that Web browsing agents could successfully internalize environmental cues about accuracy into their behaviors, thus adapting to local noise levels.

4.3. Internalization of local cues

The last experiment with graphs is aimed at testing whether local environmental cues can be internalized by reinforcement learning occurring over the lifetime of individual agents. To this end, we have endowed agents with the capability to adjust their neural nets by Q-learning (step (7) of the algorithm in figure 1). This algorithm was chosen because it is model-free and easy to implement within the connectionist framework of the agent representation (Lin, 1992); agents' neural nets are naturally used as Q-value function approximators. An agent compares the payoff of the current node with the prediction based on the features of the link that was followed to visit the node. Perceptron weights are adjusted by the delta rule to improve the accuracy of the link predictor. The net instantaneous energy change (payoff

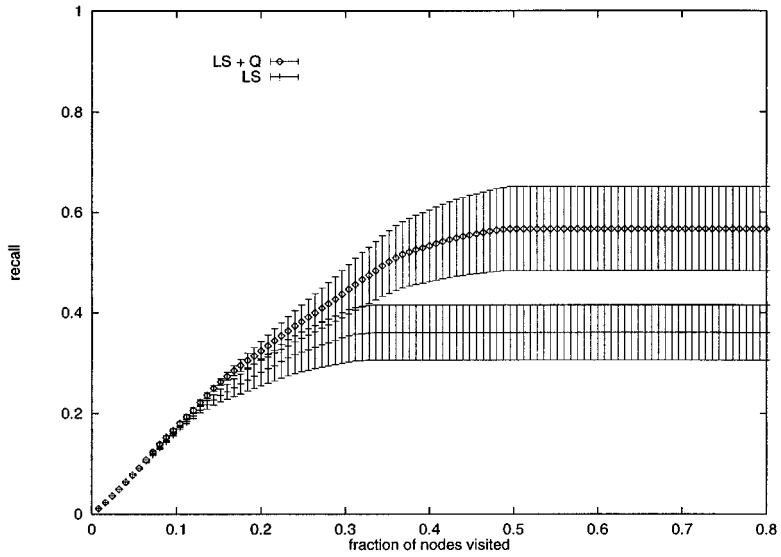

Figure 5. Recall levels achieved with and without Q-learning, averaged over multiple runs. Standard errors are also shown.

minus cost) is used as reinforcement signal, with future values discounted by a factor of 0.5. Learned changes to the weights are inherited by offspring at reproduction.

We have run two experiments with graphs having 10000 nodes, $G = 0.25$, $R = 0.5$, and $H = 1$. In the two experiments Q-learning is disabled and enabled, respectively. Figure 5 shows that Q-learning allows for a significant improvement in performance. Agents are able to learn, based on where they are situated, the appropriate network weights allowing them to correctly predict payoff. This experiment supports the argument that Web agents could internalize the different local features of relevant document clusters, and discriminate them from irrelevant background.

5. InfoSpiders implementation

The methods discussed above have been applied in the construction of populations of information agents. The *InfoSpiders* system was implemented to test the feasibility, efficiency, and performance of adaptive, on-line, browsing Web agents. In this section we describe the InfoSpiders design and implementation and in particular discuss the details of the distributed evolutionary algorithm and agent representation used.

5.1. Algorithm

InfoSpiders search on-line for information relevant to the user, by making autonomous decisions about what links to follow. Figure 6 shows the InfoSpiders implementation of the

```
1.  initialize p_0 agents, each with energy E = θ/2
2.  loop:
3.      foreach alive agent a:
4.          pick link from current document
5.          fetch new document D
6.          E_a ← E_a − c(D) + e(D)
7.          Q-learn with reinforcement signal e(D)
8.          if (E_a ≥ θ)
9.              a' ← mutate(recombine(clone(a)))
10.             E_a, E_a' ← E_a/2
11.         elsif (E_a ≤ 0)
12.             die(a)
13.     process optional relevance feedback from user
```

Figure 6. High-level pseudocode of an evolutionary algorithm based on local selection for distributed information agents.

local selection algorithm. A central part of the system is the use of *optional* relevance feedback. The user may assess the relevance of (some of) the documents visited by InfoSpiders up to a certain point. Such relevance assessments take place asynchronously with respect to the on-line search, and alter the subsequent behaviors of agents on-line by changing the energy landscape of the environment. The process is akin to the replenishment of environmental resources; the user interacts with the environment to bias the search process. Let us first overview the algorithm at a high level; representation-dependent details will be given in the next subsections and experimental parameter values in the following section.

The user initially provides a list of keywords and a list of starting points, in the form of a bookmark file.[6] In step (1), the population is initialized by pre-fetching the starting documents. Each agent is "positioned" at one of these document and given a random behavior (depending on the representation of agents) and an initial reservoir of energy.

In step (4), each agent "senses" its local neighborhood by analyzing the text of the document where it is currently situated. This way, the relevance of all neighboring documents—those pointed to by the hyperlinks in the current document—is estimated. Based on these link relevance estimates, in step (5) the agent "moves" by choosing and following one of the links from the current document.

In step (6), the agent's energy is updated. Energy is needed in order to survive and move, i.e., continue to visit documents on behalf of the user. Agents are rewarded with energy if the visited documents appear to be relevant. The $e(\,)$ function is used by an agent to evaluate the relevance of documents. If a document had previously been visited and assessed by the user, the user's assessment is used; if the document had not been visited before, its relevance must be estimated. This mechanism is implemented via a cache, which also speeds up the process by minimizing duplicate transfers of documents. While in the current, client-based implementation of InfoSpiders this poses no problem, caching is a form of communications and thus a bottleneck for the performance of distributed agents. In a distributed implementation, we imagine that agent will have local caches. When using

the current implementation to simulate the performance of distributed InfoSpiders, we will simply set the cache size to zero.

Agents are charged energy costs for the network load incurred by transferring documents. The cost function $c(\,)$ should depend on used resources, for example transfer latency or document size. For simplicity we will assume a constant cost for accessing any new document, and a (possibly smaller) constant cost for accessing the cache; this way stationary behaviors, such as going back and forth between a pair of documents, are discouraged.

Just as for graph search, instantaneous changes of energy are used, in step (7), as reward/penalty signals. This way agents adapt during their lifetime by Q-learning. This adaptive process allows an agent to modify its behavior based on prior experience, by learning to predict the best links to follow.

In steps (8–12), an agent may be killed or be selected for reproduction. In the latter case offspring are recombined by the use of *local crossover,* whereby an agent can only recombine with agents residing on the same document, if there are any.[7] Offspring are also mutated, providing the variation necessary for adapting agents by way of evolution.

Finally, in step (13), the user may provide the system with relevance feedback. It is important to stress that this process is entirely optional—InfoSpiders can search in a completely unsupervised fashion once they are given a query and a set of starting points. Relevance feedback takes place without direct on-line interactions between user and agents. The user may assess any visited document D with feedback $\phi(D) \in \{-1, 0, +1\}$. All the words in the document are automatically assessed by updating a "feedback list" of encountered words. Each word in this list, k, is associated with a signed integer ω_k that is initialized with 0 and updated each time any document is assessed by the user:

$$\forall k \in D : \omega_k \leftarrow \omega_k + \phi(D).$$

The word feedback list is maintained to keep a global profile of which words are relevant to the user.

The algorithm terminates when the population goes extinct for lack of relevant information resources, or if it is terminated by the user.

5.2. Agent architecture

Figure 7 illustrates the architecture of each InfoSpiders agent. The agent interacts with the information environment, that consists of the actual networked collection (the Web) plus data kept on local disks (e.g., relevance feedback data and cache files). The user interacts with the environment by accessing data on the local client (current status of a search) and on the Web (viewing a document suggested by agents) and by making relevance assessments that are saved locally on the client and will be accessed by agents as they subsequently report to the user/client. There is no direct interaction between the user and the agents.

The InfoSpiders prototype is written in C and runs on UNIX and MacOS platforms. The Web interface is based on the `W3C` library. Agents employ standard information retrieval tools such as a filter for noise words (Fox, 1992) and a stemmer based on Porter's algorithm (Frakes, 1992). Finally, agents store an efficient representation of visited documents in the shared cache on the client machine. Each document is represented by a list of stemmed

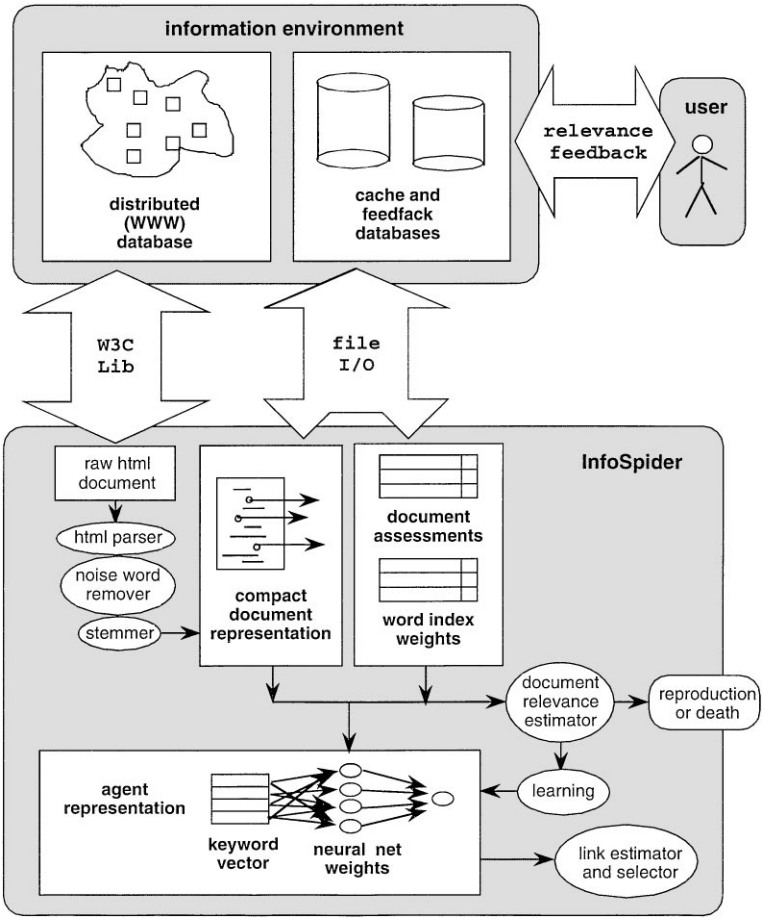

Figure 7. Architecture of an InfoSpider. Interactions between the user and the agents (relevance feedback) are indirect and mediated by the environment. The agent's representation (keywords and neural net) is used to decide which link to follow from the current document, while documents are evaluated based on the user's assessments (or initial query) to provide energy updates and reinforcement signals.

keywords and links (with their relative positions). If the cache reaches its size limit, the least recently used page is replaced.

5.3. Adaptive agent representation

Figure 7 highlights the central dependence of the InfoSpiders system on agent representation. The adaptive representation of InfoSpiders consists of the genotype, that determines the behavior of an agent and is passed on to offspring at reproduction; and of the actual mechanisms by which the genotype is used for implementing search strategies.

The first component of an agent's genotype consists of the parameter $\beta \in \Re^+$. Roughly, it represents the degree to which an agent trusts the descriptions that a page contains about its outgoing links. β is initialized with β_0.

Each agent's genotype also contains a list of keywords, initialized with the query terms. Since feed-forward neural nets are a general, versatile model of adaptive functions, we use them as a standard computation device. Therefore genotypes also comprise a vector of real-valued weights, initialized randomly with uniform distribution in a small interval $[-w_0, +w_0]$. The keywords represent an agent's opinion of what terms best discriminate documents relevant to the user from the rest. The weights represent the interactions of such terms with respect to relevance. The association of an agent's keyword vector with its neural net highlights the significant difference between the representation in this model and the vector space model (Salton & McGill, 1983).

The neural net has a real-valued input for each keyword in its genotype and a single output unit. We want to allow the inputs and activation values of the network to take negative values, corresponding to the possibly negative correlations perceived between terms and relevance. For this reason the network uses the hyperbolic tangent as its squashing function, with inputs and activation values in $[-1, +1]$. Let us now see how the different parts of the system are implemented, based on this representation.

5.3.1. Action selection. An agent performs action selection by first computing the relevance estimates for each link from the current document. This is done by feeding into the agent's neural net activity corresponding to the small set of (genetically specified) keywords to which it is sensitive. Each input unit of the neural net receives a weighted count of the frequency with which the keyword occurs in the vicinity of the link to be traversed. In the experiments reported here, we use a distance weighting function which is biased towards keyword occurrences most close to the link in question. More specifically, for link l and for each keyword k, the neural net receives input:

$$in_{k,l} = \sum_{i:dist(k_i,l)\leq\rho} \frac{1}{dist(k_i, l)}$$

where k_i is the ith occurrence of k in D and $dist(k_i, l)$ is a simple count of intervening links (including l and up to a maximum window size of $\pm\rho$ links away). The neural network then sums activity across all of its inputs; each unit j computes a logistic activation function

$$o_j = \tanh\left(b_j + \sum_k w_{jk} in_k^l\right)$$

where b_j is its bias term, w_{jk} are its incoming weights, and in_k^l its inputs from the lower layer. The output of the network is the activation of the output unit, λ_l. The process is illustrated in figure 8 and is repeated for each link in the current document. Then, the agent uses a stochastic selector to pick a link with probability distribution:

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

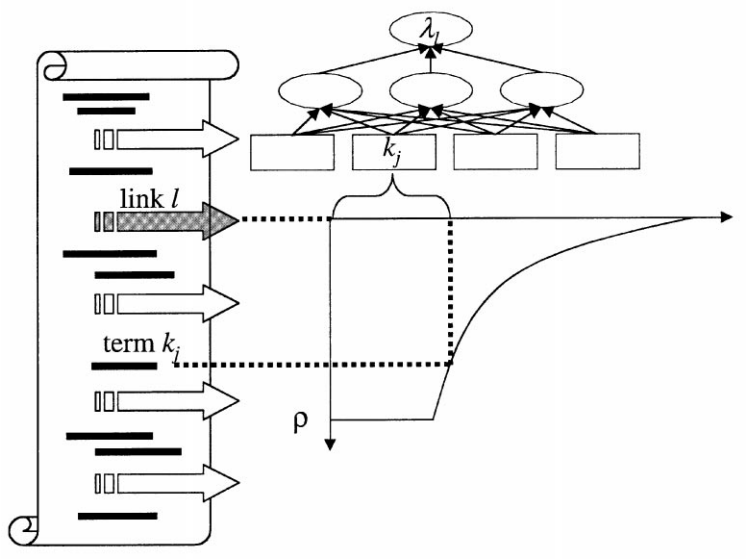

Figure 8. How an agent estimates each link from the current document. For each link in the document, each input of the neural net is computed by counting the document words matching the keyword corresponding to that input, with weights that decay with distance from the link up to the window size ρ.

5.3.2. Relevance estimation and feedback. After a link has been chosen and the corresponding new document has been visited, the agent has to determine the corresponding energy gain and loss; both depend on whether or not the document had been visited previously. If the document is in the cache, and the user has assessed its relevance, then the agent receives energy $e(D) = \phi(D)$, after which $\phi(D)$ decays according to

$$\phi(D) \leftarrow \gamma\phi(D)$$

where the decay factor γ is a parameter. This is done to avoid a population explosion due to non-conserved energy from relevance feedback.

If the user provided the system with relevance assessments, the word feedback list represents a profile of his/her interests that is both more current and more accurate than the original query. This list is used to estimate the relevance of previously unvisited or not assessed documents, so that the corresponding energy intake can be computed:

$$e(D) = \tanh\left(\sum_{k\in D} freq(k, D) \cdot I_k\right)$$

where $freq(k, D)$ is the frequency of term k in document D normalized by document size, and I_k is the weight of term k based on relevance feedback. The latter is an extension of the TF-IDF (term frequency-inverse document frequency) index weighting scheme. If k

is not in the word feedback list, then $I_k = 0$. The list is initialized with the query terms and weights $I = 1$. It changes only if/when the user provides InfoSpiders with relevance feedback. In these cases the weights are updated according to the rule:

$$I_k \leftarrow \alpha \cdot I_k + (1 - \alpha) \cdot \omega_k \cdot \left[1 + \log\left(\frac{1}{C_k}\right)\right]$$

where C_k is the fraction of cache documents containing k. α is an inertia term. Such a weighting formula differs from more traditional TF-IDF schemes (Sparck Jones, 1972) in at least two respects. First, it is not aimed at weighting terms based on how well they describe documents, but rather on how well they correlate with relevance. Therefore it employs algebraic term frequencies (ω_k) to account for terms found in documents that are anti-correlated with relevance. Second, it is computed on-line and therefore uses document frequencies based on the contents of the cache rather than the entire collection. The hyperbolic tangent is used to normalize energy intakes into the appropriate range $[-1, +1]$—the same range as the corresponding neural nets prediction.

5.3.3. Q-learning. The agent then compares the relevance (assessed or estimated) of the current document with the estimate of the link that led to it. By using the connectionist version of Q-learning (Lin, 1992), the neural net can be trained on-line to predict values of links based on local context. After the agent visits document D, $e(D)$ is used as an internally generated reinforcement signal to compute a teaching error:

$$\delta(D) = e(D) + \mu \cdot \max_{l \in D}\{\lambda_l\} - \lambda_D$$

where μ is a future discount factor and λ_D the prediction from the link that was followed to get to D. The neural net's weights are then updated by back-propagation of error (Rumelhart, Hinton, & Williams, 1986). Learned changes to the weights are "Lamarckian" in that they are inherited by offspring at reproduction (see Belew & Mitchell (1996) for a treatment of this issue).

In the absence of relevance assessments, this reinforcement learning algorithm is unsupervised because it is the environment that provides the reinforcement signal $e(D)$. However, relevance feedback alters the function $e(\,)$ under the supervision of the user, who modifies the environment by providing examples of relevance. Therefore InfoSpiders integrate unsupervised and supervised adaptation in the form of evolution, Q-learning, and relevance feedback.

5.3.4. Reproduction. At reproduction, the offspring clone may be recombined with another agent. Two-point crossover is applied to the keywords of the clone, so that a subset of the mate's keywords is spliced into the offspring's keyword vector.

Then the offspring is mutated to provide the evolutionary algorithm with the necessary power of exploration. If a' is an offspring of a:

$$\beta_{a'} \leftarrow U[\beta_a(1 - \kappa_\beta), \beta_a(1 + \kappa_\beta)]$$

where $\kappa_\beta \in [0, 1]$ is a parameter and U is the uniform distribution. The values of β are clipped to β_{max} to maintain some exploratory behavior. The neural net is mutated by adding random noise to a fraction ζ_w of the weights. For each network connection i:

$$w^i_{a'} \leftarrow U\big[w^i_a(1-\kappa_w), w^i_a(1+\kappa_w)\big].$$

The keyword vector is mutated with probability ζ_k. The least useful (discriminating) term $\arg\min_{k\in a'}(|I_k|)$ is replaced by a term expected to better justify the agent's performance with respect to the user assessments. In order to keep any single keyword from taking over the whole genotype, this mutation is also stochastic; a new term is selected with probability distribution

$$\Pr[k] \propto freq(k, D) \cdot |I_k|$$

where D is the document of birth. The first factor captures the local context by selecting a word that well describes the document that led to the energy increase resulting in the reproduction. The second factor captures the global context set by the user by selecting a word that well discriminates the user's preferences. Learning will take care of adjusting the neural net weights to the new keyword.

The evolution of keyword representations via local selection, mutation and crossover implements a form of *selective query expansion.* Based on relevance feedback and local context, the query adapts over time and across different places. The population of agents embodies a distributed, heterogeneous model of relevance that may comprise many different and possibly inconsistent features. But each agent focuses on a small set of features, maintaining a well-defined model that remains manageable in the face of the huge feature dimensionality of the search space.

6. InfoSpiders evaluation

In this section we report on results of experiments and analysis aimed at evaluating the performance of InfoSpiders in responding to queries by searching the Web on-line. In so doing we extend previous, preliminary results (Menczer & Belew, 1998a).

In Section 5 several algorithm parameters have been mentioned. Table 2 shows the values taken by all parameters in the experiments described in this section. The cost c is such that an agent can visit 1000 irrelevant documents before it will run out of energy. The effects of cache size are not considered in the experiments described here; although for efficiency purposes the cache is large enough to contain all of the visited documents, the same cost c is assessed whether or not a document is in the cache. Therefore the algorithm effectively simulates the behavior of distributed InfoSpiders.[8]

For each query, the search is stopped when the population meets the success criterion described in the next subsections (unless it goes extinct or visits a total of T_{max} new pages).

Table 2. InfoSpiders parameter descriptions and values for the experiments reported in this paper. Some parameter values are determined empirically, while others are explored elsewhere (Menczer, 1998).

Parameter	Value	Description
p_0	21	Initial population size
θ	2.0	Reproduction threshold
c	0.001	Energy cost per document
T_{max}	10,000	Max number of new pages visited per query
β_0	2.0	Initial β
κ_β	0.5	β mutation range
β_{max}	5.0	Max β
ρ	5	Half-size of link estimation sliding window
ζ_k	0.5	Keyword mutation rate
N_{layers}	2	Neural net layers (excluding inputs)
w_0	0.5	Initial neural net weight range
ζ_w	0.2	Neural net weight mutation rate
κ_w	0.25	Neural net weight mutations range
η	0.05	Neural net Q-learning rate
μ	0.5	Q-learning discounting factor
α	0.5	Inertia of word feedback weights
γ	0.9	Decay factor for document assessments
F_{max}	64	Max number of word feedback entries
$\|C\|$	T_{max}	Max cache size

6.1. The EB search graph

The difficulty of evaluating on-line retrieval systems stems from multiple factors. The lack of a ranking function over all documents (in particular those not seen) is one problem that will be addressed in the next subsection. A more general difficulty is the lack of queries with available well-defined relevant sets. To overcome this problem, a special chunk of the Web has been selected as a test environment (Steier, 1994): the Encyclopaedia Britannica (EB) (Encyclopaedia Britannica, Inc. http://www.eb.com.). The advantage is that we can make use of readily available relevant sets of articles associated with a large number of queries.

Here we use a subset of the EB corpus, corresponding to the "Human Society" topic—roughly one tenth of the whole collection. Any links to other parts of the EB are removed, and so are the documents left without outlinks as a result. The final environment is made of $N = 19427$ pages, organized in a hypertext graph (the EB is already in HTML format). 7859 of these pages are full articles constituting the *Micropaedia*. These, together with 10585 *Index* pages (containing links to articles and pointed to by links in articles), form a graph with many connected components. The remaining 983 nodes form a hierarchical topical tree, called *Propaedia*. These nodes contain topic titles and links to children nodes, ancestor nodes, and articles. Micropaedia articles also have links to Propaedia nodes. Propaedia and

Table 3. Propaedia query statistics and examples. Multiple measurements are obtained from a single query by restarting the search with different initial conditions (i.e., different seeds for the random number generator).

Depth	# Queries	# Measurements	$\langle G \rangle$	Example
1	12	84	0.02	`Branches of private law, substantive and procedural`
2	72	144	0.003	`Laws governing economic transactions`
3	100	200	0.002	`Law of commercial transactions`
4	100	200	0.001	`Principal forms of business associations`
5	12	84	0.0007	`State and municipal corporations, quasi-public enterprises and utilities`

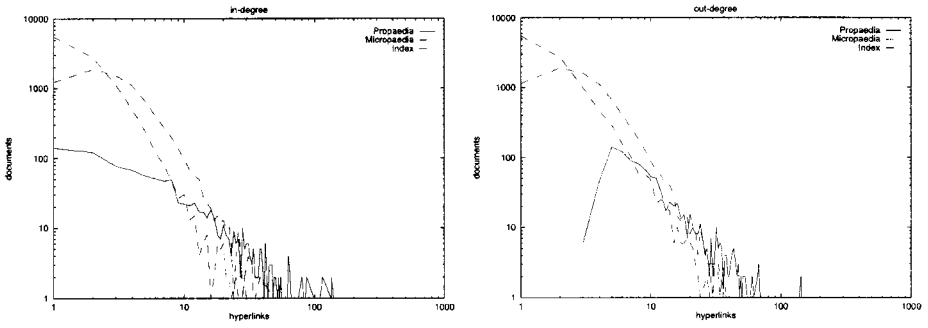

Figure 9. In-degree and out-degree distribution histograms for the EB Human Society graph. There is only one document with zero in-degree (the Propaedia root), and no document with zero out-degree.

Index pages are included in the search set to ensure a connected graph and to be faithful to the EB information architecture—an actual subset of the Web. Figure 9 shows the in- and out-degree statistics of the search graph. These illustrate the main disadvantage of using the EB dataset—its careful design yields an unrealistically well-structured hypertext environment in comparison to the noisier full Web.

Articles are manually classified according to the Propaedia hierarchy by skilled human editors. By using the title of any Propaedia node as a query we have access to all the articles classified into that category by the editor, who has exhaustive knowledge of all the documents in that subject. Therefore we will use the set of Micropaedia articles associated with the *subtree rooted at the query node* as the relevant set corresponding to that query.

Since the Propaedia topology is used to define relevance, its "visibility" to agents during search would make the problem of navigating through the relevant set quite easy. Therefore one modification is made to the search space for each query. We remove all the Propaedia nodes in the relevant subtree, so that agents cannot access relevant nodes directly from the Propaedia, but only from Index or other Micropaedia nodes. This is illustrated schematically in figure 10 for the example query used in Section 6.3.

Table 3 shows some statistics and examples for the queries used in the experiments. Queries corresponding to nodes at the same depth in the Propaedia tree are grouped together.

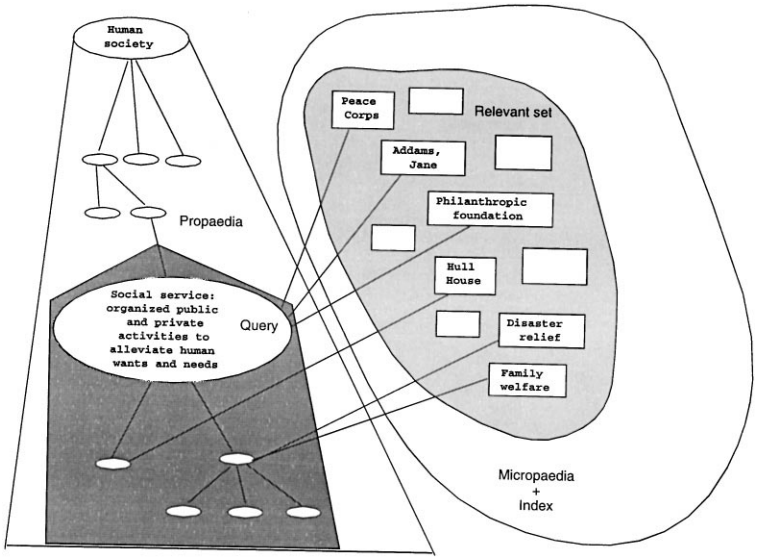

Figure 10. Schematic representation of the EB search space for an actual query, showing the titles of a few relevant articles. The relevant set is depicted in light gray. The subtree in dark gray is removed from the search space.

Generality statistics are obtained dividing the size of the relevant set by N, and averaging over same-depth queries.[9] Queries span five different depths. If d is the depth of a query, the minimal distance from the staring node (the Human Society root) to the root of the relevant Propaedia subtree is $d + 1$. However, since the latter is not part of the search graph, the actual shortest path to a relevant article is at least $d + 3$. Deeper queries are harder for InfoSpiders because they are less general (lower carrying capacity) and their relevant sets are farther from the starting point. The number of different queries available also varies with depth. The small number of queries available at the maximum depth, corresponding to maximum variance, will yield the greatest standard error in performance measurements.

6.2. Performance evaluation

Let us quantitatively analyze the behavior of the InfoSpiders algorithm on the EB corpus. Since InfoSpiders do not have access to the whole corpus, but only to the subset of documents they actually visit, it is difficult to impose a ranking over the whole collection. Furthermore, we want to focus on the novel aspect of the InfoSpiders approach—the location of documents by autonomous on-line browsing—rather than the ranking performance of the algorithm, which is not addressed in this paper. Therefore we choose to avoid using standard information retrieval measures such as precision-recall curves, which require rank or similar measures as a control parameter (van Rijsbergen, 1979).

Search length is an alternative metric that better lends itself to assess the performance of on-line retrieval systems. It is normally defined as the number of irrelevant documents that appear, in some ordered list of retrieved documents, in front of some fraction of the relevant set (Cooper, 1968). We can easily extend this method by imagining that only visited documents appear in the list of retrieved documents, and that their ordering is given by visit time rather than rank—length then refers to waiting time. For an on-line retrieval system this means that we only need to wait until some fraction of the relevant set is visited, and count the total number of documents visited up until that time. For each query in the experiments discussed here, search length is measured for an arbitrary recall level of 10%, i.e., when $\lceil 0.1 \cdot G \cdot N \rceil$ relevant articles have been visited.

We want to compare InfoSpiders with the best global search algorithm. When searching an annotated graph, the optimal strategy is given by the A^* algorithm. However, in the case of the Web there is no admissible criterion to apply A^*, since the only suitable lower bound on the distance from the closest relevant node is trivially 0. Therefore A^* does not have an admissible heuristic other than the trivial one, and reduces to *best-first-search* (BFS). InfoSpiders can implement a search strategy similar to BFS by evolving high values for the β gene, but only from the local "perspective" of single agents, rather than with respect to the global search space. Nevertheless, assuming BFS as an upper bound for global search algorithms, we compare its performance with InfoSpiders. We implemented BFS by means of a priority queue. Links are estimated in the same way in which they are by InfoSpiders, but then they are placed in a global queue, sorted by estimated relevance of the pointed documents; then documents are visited by following the queued links, in the order determined by their priority.[10]

Another complication arises due to potential distributed implementations. In a sequential algorithm, all visited documents are retrieved from the servers on which they reside and analyzed locally on the client machine. In this case the cache helps minimize the number of repeated document requests over the network. Since this is also the case for BFS, we allow BFS to use an arbitrarily long cache, so that *only previously unvisited documents* contribute to the measured search length. For InfoSpiders, we want to simulate a distributed implementation of the algorithm. This is why those parts of the system that require communication among agents (shared access to the global cache and differential costs for new and previously visited documents) are disallowed. Therefore we include *each and every document visited* into the search length statistics, be it requested across the network or loaded from the cache.[11] Agents then effectively execute in parallel, and search length is given by the maximum number of documents visited by any agent lineage. This is measured in the experiments by keeping track of the number of links traversed by each agent, accumulated over all generations in the agent's ancestry since the start of a run.

We have run experiments by initializing the agents at the Human Society root of the Propaedia and comparing the search length achieved by BFS with that of two variants of InfoSpiders, one without relevance feedback and one with relevance feedback every 50 newly visited documents. In the latter case, $\phi(D) = 1$ assessments are automatically generated for all documents D visited so far that belong to the editor-defined relevant set.

Figure 11 shows the percentages of queries successfully completed by the different algorithms. These are the cases in which 10% of the relevant pages are found within the T_{max} limit on visited pages. Non completed queries are those for which InfoSpiders run out

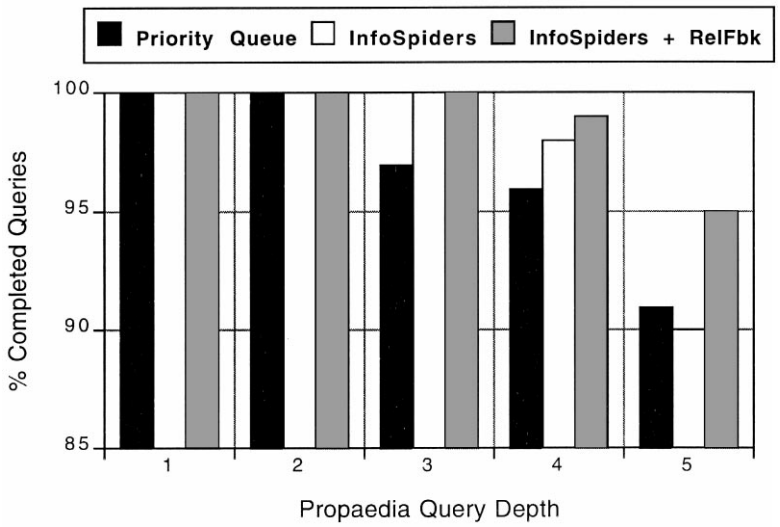

Figure 11. Percentage of queries successfully completed by BFS (Priority Queue) and different versions of InfoSpiders.

of time, or go extinct; and for which priority queue runs out of time, or becomes engulfed in previously visited areas. Performance degrades for queries of increasing depth. This is expected: like humans, agents can "get lost" or "get tired and give up." InfoSpiders however tend to have a higher success rate than BFS (with the only exception of depth 5 for which InfoSpiders without relevance feedback do worse than BFS by 1%). Relevance feedback affords an improvement where unsupervised InfoSpiders fail to achieve 100% completion rate. This is due to the evolutionary reinforcement provided by relevance feedback: those agents that are moving in the right direction receive energy boosts and have a greater chance to reach the relevant clusters.

Figure 12 plots search length versus query depth. Search length is averaged over same-depth queries that are successfully completed. As the plots demonstrate, InfoSpiders' search length increases with depth while the global heuristic does not show such dependency, at least within the observed depth range. BFS seems to either fail or succeed with depth-independent search length. As a result, BFS outperforms InfoSpiders for very deep completed queries.

This result provides insight into the complementarity between the InfoSpiders approach and traditional search engines. In practical cases we expect the search to start not too far away from the desired pages, thanks to the use of search engines to seed the initial population of InfoSpiders. Under this assumption, figure 12 shows that the distributed nature of InfoSpiders results in a significantly shorter search time than required by BFS. The hypothesis that realistic queries correspond to lower depths requires empirical confirmation. Interestingly, human browsing behavior leads to similar results—people rarely follow more than a few links at any given site. For example, AOL users typically request only one page from any server (the mode of the browsing depth is one click, the average is three clicks per server) (Huberman et al., 1998).

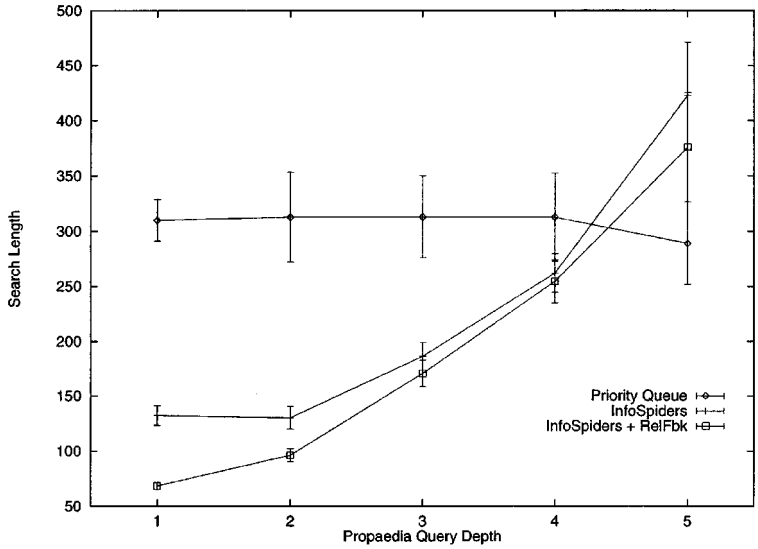

Figure 12. Performance of InfoSpiders (with and without relevance feedback) versus BFS. Error bars correspond to standard errors.

Figure 12 also illustrates that when user relevance feedback is available, it further accelerates the discovery of relevant documents, by pointing agents in the right direction. The improvement becomes less significant statistically for deep queries due to the fewer available measurements.

6.3. Micro analysis

To illustrate how some of the goals that we set out for adaptive information agents are achieved by InfoSpiders, let us now look more closely at a few typical agents adapting within a single search. The query is "`Social service: organized public and private activities to alleviate human wants and needs`" and, after removal of stop words and stemming, it results in the keyword vector shown in Table 4.

To simplify the analysis during this example run we used simple perceptrons to represent agent behaviors, so that an agent is completely described by a vector of 8 keywords and a vector of 9 weights (one per keyword plus a bias term), plus the β parameter. We ran the search until the population had visited 1000 new pages, and provided the population with relevance feedback every 50 new pages.

6.3.1. Adapting to "spatial" context. How do InfoSpiders internalize environmental word features that are spatially local (in the sense of linkage topology)? Can they adapt to the spatial context in which they evolve? To answer these questions consider two agents, A and B, born at the same time but in different places. More precisely, A was born at time 554 in the sequential execution of the algorithm, i.e., after the population had collectively visited

Table 4. Default initial word feedback list corresponding to a query.

k	I_k
ORGAN	1.0
PUBLIC	1.0
PRIVAT	1.0
SERVIC	1.0
SOCIAL	1.0
HUMAN	1.0
ACTIV	1.0
ALLEVI	1.0

554 new pages. B was born at time 580; A and B were effectively contemporaries because they had the same temporal context—the global information resulting from the relevance assessments of time 550, partially shown in Table 5.

As Table 5 shows, the original query words were displaced from their top positions and replaced by new terms. For example, PRIVAT and ALLEVI had relatively low weights, while FOUNDAT appeared to have the highest correlation with relevance feedback at this time (cf. figure 10).

A's and B's keyword vectors are shown in Table 6. In the course of the evolution leading to A and B through their ancestors, some query terms were lost from both genotypes. A was a third generation agent; its parent lost ALLEVI through a mutation in favor of HULL. At A's birth, PRIVAT was mutated into TH. B was a second generation agent; at its birth, both ALLEVI and PRIVAT were replaced by HULL and ADDAM, respectively, via mutation and crossover. These keyword vectors demonstrate how environmental features correlated with relevance were internalized into the agents' behaviors.

The difference between A and B can be attributed to their evolutionary adaptation to spatially local context. A and B were born at documents D_A and D_B, respectively, whose word frequency distributions are partly shown in Table 7. TH represented well the place where A was born, being the second most frequent term there; and ADDAM represented well the place where B was born, being the third most frequent term there. By internalizing these words, the two situated agents are better suited to their respective spatial contexts.

6.3.2. Adapting to "temporal" context. Let us now consider adaptation along the temporal dimension. How do InfoSpiders internalize features appropriate for their time? Can they capture the temporal context in which they evolve? To answer these questions consider again two agents, B and C, born in the same place (D_B; see Table 7) but at different times. More precisely, C was born at time 965, and therefore its temporal context was the global information resulting from the relevance assessments of time 950, partially shown in Table 8. B's temporal context was given in Table 5.

Let us make a few observations about Table 8. After more relevant documents were discovered, the original query terms gained weight, with four of them in the top six positions. Their relative positions also changed; for example, PRIVAT surpassed HUMAN. Other words

Table 5. Part of the word feedback list and weights at time 550. Stars mark new terms not present in the original query. Note that TH does not correspond to the article "the," which is a noise word and thus removed from all documents; rather, it corresponds to the "th" used for ordinal numbers and often associated with centuries.

Rank	New	k	I_k
1	⋆	FOUNDAT	0.335
2	⋆	RED	0.310
3	⋆	MISSION	0.249
4		SOCIAL	0.223
5	⋆	CROSS	0.197
6	⋆	HULL	0.184
7	⋆	HOUS	0.183
8		ORGAN	0.161
		...	
15		SERVIC	0.114
16		ACTIV	0.112
		...	
23	⋆	TH	0.094
		...	
30		PUBLIC	0.087
		...	
32	⋆	ADDAM	0.079
		...	
37		HUMAN	0.075
		...	
41		PRIVAT	0.067
		...	
44		ALLEVI	0.065
		...	

Table 6. Keyword vectors for agents A and B.

A	B
ORGAN	ORGAN
PUBLIC	PUBLIC
TH	ADDAM
SERVIC	SERVIC
SOCIAL	SOCIAL
HUMAN	HUMAN
ACTIV	ACTIV
HULL	HULL

Table 7. Most frequent terms in the documents where agents A and B were born. Word frequencies are normalized by the total number of words in each document.

$rank_{D_A}$	k	$freq(k, D_A)$	$rank_{D_B}$	k	$freq(k, D_B)$
1	WORKHOUS	0.076	1	HOUS	0.043
2	TH	0.038	1	HULL	0.043
2	POOR	0.038	3	ADDAM	0.025
4	SOCIAL	0.030		...	
4	CENTURI	0.030	38	AMERICAN	0.004
	...			...	

Table 8. Part of the word feedback list and weights at time 950. Stars mark new terms not present in the original query. Query words have regained the top three positions.

Rank	New	k	I_k
1		SERVIC	0.273
2		SOCIAL	0.268
3		ORGAN	0.238
4	⋆	FOUNDAT	0.152
5	⋆	NATION	0.148
6		PUBLIC	0.138
		...	
12		ACTIV	0.118
		...	
15	⋆	HULL	0.110
		...	
17		PRIVAT	0.098
		...	
25		HUMAN	0.087
		...	
31		ALLEVI	0.080
		...	
35	⋆	AMERICAN	0.077
		...	
56	⋆	ADDAM	0.053
		...	

lost importance; among those with a presence in B's and C's birth page D_B, both HULL and ADDAM decreased their weights, but while the former maintained a relatively strong position, the latter did not. Finally, some new terms made their first appearance in the list, such as AMERICAN that was also represented in D_B but was not present in the word feedback list at time 550.

Table 9. Keyword vectors for agents B (from Table 6) and C.

B	C
ORGAN	ORGAN
PUBLIC	PUBLIC
ADDAM	PRIVAT
SERVIC	SERVIC
SOCIAL	SOCIAL
HUMAN	AMERICAN
ACTIV	ACTIV
HULL	HULL

Table 9 shows the differences between the representations of agents B and C. Such differences reflect the times in which these agents were born. When B was born, ADDAM appeared better correlated with relevance then PRIVAT, while the converse was true when C was born. The internalization of the two terms by B and C, respectively, is consistent with this change in temporal context. Furthermore, at the time of C's birth AMERICAN had a small but positive global weight, so that the presence of this term in D_B could be picked up by C—something impossible for B due to the term's absence in the earlier word feedback list. By evolving to internalize these words, the two agents adapted to their respective temporal contexts, shaped by the dynamics of the user's personal preferences.

6.3.3. Learning from experience. One last question is: Can InfoSpiders internalize their local context over smaller spatial scales and shorter time scales, during their lives? To answer, consider two agents D and E in the initial population. Both lived until the end of the run and were successful (with three and nine offspring, respectively). Although D and E were born at the same time and in the same place, they searched through different paths and therefore had different life experiences.

Table 10 shows the weight vectors of D's and E's neural nets at the end of the run. The weights were adapted via Q-learning so that each agent would be able to estimate document relevance across links. For example, it is clear that the strategy learned by D paid special attention to PUBLIC. On the contrary, E's predictions were anticorrelated with the presence of PUBLIC. This demonstrates that the local contexts experienced by D and E during their lives were quite divergent with respect to this word. Through reinforcement learning, the two agents were able to internalize into their neural nets the perceived correlations between environmental features and relevance.

6.4. Web case study

We conclude this evaluation section briefly outlining the result of a simple case study described in greater detail elsewhere (Menczer & Monge, 1999). An *ad-hoc* query was constructed in such a way that the small relevant set (a total of four Web pages) was known *a priori*. Three of these pages had been recently published on the Web, so that none of

Table 10. Learned neural net weights for agents D and E.

k	w_k^D	w_k^E
ORGAN	0.22	−0.14
PUBLIC	0.75	−0.42
PRIVAT	0.20	0.81
SERVIC	0.22	−0.05
SOCIAL	−0.01	−0.21
HUMAN	0.07	−0.03
ACTIV	0.12	0.27
ALLEVI	0.29	−0.08
Bias	0.02	0.41

the major search engines had yet indexed any of them. The remaining page was old and previously indexed by all of the major search engines, but it had been recently updated to include, among its many links, an additional link to one of the other three relevant pages. The linked new page included, among others, links to the remaining two pages, so that all the relevant set was within two links from the indexed page.

The query was submitted to the Excite (Excite. http://www.excite.com.) search engine. As expected, Excite returned the only relevant page indexed (a recall of 1/4 and a precision of about 3×10^{-7}). The hit was ranked first. A small population of 10 InfoSpiders was then initialized at the top 10 pages returned by Excite (one agent per page) and allowed to search on-line, adapting by evolution and query-based reinforcement learning, without any user-supplied relevance feedback. After 66 new pages had been visited, all of the relevant pages had been located (a recall of 1 and a precision of about 0.05 if we count the startup pages among those visited). The search took 9 minutes. The purpose of this case study is limited to an illustration of the InfoSpiders potential. The main point is that complementing a search engine with on-line browsing agents can overcome limitations in coverage and currency, and thus address the scalability challenge.

7. Related work

The following discussion of the main connections between our work and that of others in machine learning and information retrieval reflects the fact that distinctions between these two areas are blurred in this paper, due to a problem-oriented rather than a discipline-oriented approach.

7.1. Machine learning

Evolutionary algorithms using local selection are a general adaptive paradigm for distributed agents. While we have shown elsewhere that the approach is not suitable in every domain (e.g., combinatorial optimization (Menczer & Belew, 1998b)), local selection has proven successful in multi-criteria optimization problems requiring a heterogeneous cover of the

search space rather than a convergence to the perceived global optimum. In the evolutionary algorithm community this behavior is often referred to as *niche selection* (De Jong & Sarma, 1995; Mahfoud, 1995).

The most notable selection variations explicitly aimed at niching are *crowding* (De Jong, 1975) and *fitness sharing* (Goldberg & Richardson, 1987). In both of these methods selection is altered to take into account some measure of similarity among individuals, leading to inefficiency; if p is the population size, selection has time complexity $O(p)$ rather than $O(1)$ *per individual*. Moreover, the population size required to maintain a cover across niches grows rapidly with the number of niches (Mahfoud, 1994). Local selection naturally enforces the maintenance of population diversity and is implicitly niched without any communication overhead. Therefore it affords efficiency both in centralized and, especially, distributed tasks.

Local selection and reinforcement learning both allow agents to internalize environmental features in an unsupervised fashion (Pack Kaelbling, Littman, & Moore, 1996). InfoSpiders seamlessly integrate population-based adaptation and individual-based learning, not to accelerate global optimization (Hart & Belew, 1996) but to take advantage of environmental signals at different spatial and temporal scales.

There is plenty of data available on-line, and although it may be noisy and inconsistent compared with manually constructed relevance assessments, adaptive algorithms must take advantage of what is cheap and realistic in the actual search environment (Sutton, 1996). The connectionist model in which InfoSpiders learn to estimate link relevance is also reminiscent of the use of neural networks to learn probability distributions for text retrieval by logistic regression (Mitchell, 1997).

If relevance assessments from the user are available, active learning should take advantage of them because it has been shown that they can considerably improve the performance of retrieval systems (Lewis, 1995). This is why, in a user-oriented system such as InfoSpiders, we have integrated unsupervised adaptation with "adaptation by examples," driven by relevance feedback. The environmental model behind local selection allows relevance feedback to interact asynchronously with the on-line agents. Relevance feedback is a selfish process from the user's standpoint (Sutton, 1996), but it provides agents with modified rewards that improve on their models of relevance and therefore on their performance.

Large, distributed text collections are a typical example of massive data sets that challenge machine learning techniques due to their huge feature space dimensionality (Lewis, 1996). InfoSpiders deal with dimensionality reduction in a localized, situated way. Agents internalize those words that appear maximally correlated (or anticorrelated) with their objective function, in their (temporally and spatially) local context. This model of feature selection keeps the size of the learning problem manageable for each individual agents, while the population as a whole may consider a much larger set of heterogeneous features.

7.2. Information retrieval

Our linkage topology conjecture is equivalent to the cluster hypothesis (van Rijsbergen, 1979) under a hypertext derived definition of association. Linkage topology has been considered by others in the context of the Web and other hypertext environments, with different

motivations. For extending vector space retrieval systems, links have been used to collect index terms across neighboring documents (Salton, 1963; Salton & McGill, 1983; Kwok, 1988; Croft & Turtle, 1993). Links have also been used for enhancing relevance judgments (Rivlin, Botafogo, & Shneiderman, 1994; Weiss et al., 1996) and incorporated into query formulation to improve searching (Savoy, 1996; Arocena, Mendelzon, & Mihaila, 1997; Spertus, 1997).

In the Clever project (Chakrabarti et al., 1998a; Chakrabarti et al., 1998c), the results of a search engine are used to seed a pool of pages that is augmented with all pages pointing to or pointed by this initial set. Links between these pages are then used to distill the best authorities and hubs. Clever shares with InfoSpiders the idea that these links capture the annotative power of the many independent authors of Web pages. And like InfoSpiders, Clever exploits not only the link structure but also the text features of the pages being distilled. Unlike InfoSpiders, Clever is generally limited to the pages already indexed by a search engine.

The most direct application of machine learning to the information retrieval task has been as a type of classification task, separating "relevant" documenents from "irrelevant" ones (Lewis & Hayes, 1994; Lewis et al., 1996; Cohen, 1996). In addition to assuming a previously identified training set of documents that have been classified manually, this approach also holds constant the set of features on which discrimination is based. Learning within the InfoSpiders system is more consistent with what is sometimes called the "on-line" learning task (e.g., within the TREC 1998 competitions), in that the stream of data available to the learning algorithm later in its training changes as a consequence of its earlier performance.

A more significant difference in InfoSpiders learning is that while early generations of agents begin with a shared set of lexical features (taken from the original query), this feature set soon diverges. Evolutionary pressures select for features that help to discriminate relevant from irrelevant documents within the local context of nearby documents, in the link-distance sense. This process of selective query expansion differs from traditional uses of feature selection for information retrieval (Harman, 1992) in the role played by local context.

InfoSpiders are a first step towards applying traditional indexing methods within the *contextual* frame that naturally surrounds Web documents. All samples of language, including the documents indexed by Web search engines, depend heavily on shared context for comprehension. Authors make assumptions, often tacit, about their intended audience and when a document appears in a "traditional" medium (newspaper, academic journal, etc.) it is likely that typical readers will understand it as intended. But one of the many things the Web changes is the huge new audience it brings for documents; much of this audience may not share the author's intended context. These vague linguistic concerns have concrete manifestation in the *global* word frequency statistics collected by Web search engines. The utility of an index term, as a discriminator of relevant from irrelevant items, can become a muddy average of its application across multiple, distinct sub-corpora within which these words have more focused meaning (Steier, 1994; Steier & Belew, 1994). Agents that are *situated* at a particular location within the Web can exploit local coherence in keyword distributions by exploring link proximity. Over time, they may come to internalize those features that best discriminate between relevant and other pages with respect to their local context only.

Context also characterizes the main difference between our use of relevance feedback with TF-IDF-like weighting and the mainstream of text retrieval research in which supervised learning is employed to estimate word probability distributions (Robertson & Spark Jones, 1976; Sparck Jones, 1979). InfoSpiders use cues provided by user assessments only in conjunction with local context. The environment plays a mediating role. An agent will not waste its limited resources paying attention to a word that never appears in the current search area, even if the user likes that word a lot.

The ideas incorporated into the InfoSpiders framework suggest ways to deal with some of the new challenges posed by text classification to machine learning, especially with respect to time-varying documents and user needs (Lewis, 1996) and to large, dynamic, and heterogeneous collections (Lewis, 1997). For example, the InfoSpiders population deals with the curse of dimensionality in a scalable way; more agents can collectively select more features, but each agent's learning task does not become harder. On-line search makes the classification problem hard because the relevant classes can be heterogeneous (think of a long-standing user profile) and because class membership can change over time with the user's shifting interests. Fortunately, these factors can be viewed as assets in the construction of an agent-based retrieval systems.

7.3. Other related projects

The idea of decentralizing the index-building process is not new. Dividing the task into localized indexing, performed by a set of *gatherers*, and centralized searching, performed by a set of *brokers*, has been suggested since the early days of the Web by the Harvest project (Bowman et al., 1994).

Fish Search (De Bra & Post, 1994) was a search system proposed at the same time as InfoSpiders (Menczer, Willuhn, & Belew, 1994) and inspired by some of the same ideas from artificial life. Fish Search was based on a population of search agents who browsed the Web autonomously, driven by an internally generated energy measure based on relevance estimations. The population was client-based, and used a centralized cache for efficiency. While we believe that the algorithm could have been extended to allow for distributed implementations, each agent could not internalize local context. This was due to a fixed, nonadaptive strategy: a mixture of depth-first-, breadth-first-, and best-first-search, with user-determined depth and breadth cutoff levels. One difficulty of the Fish Search approach was in determining appropriate cutoff levels *a priori*, possibly resulting in load-unfriendly search behaviors.

There have been several examples of agents who rely on search engines to find information (e.g., homepages or papers) on behalf of users. CiteSeer (Bollacker, Lawrence, & Giles, 1998) is an autonomous Web agent for automatic retrieval and identification of publications. Ahoy (Shakes, Langheinrich, & Etzioni, 1997) is a homepage finder based on a metasearch engine plus some heuristic local search. WebFind (Monge & Elkan, 1996) is a similar locator of scientific papers, but it relies on a different information repository (*netfind*) to bootstrap its heuristic search. While agents like CiteSeer, Ahoy and WebFind may perform some autonomous search from the pages returned by their initial sources, this is strongly constrained by the repositories that provide their starting points, and usually limited to servers known to them.

WebWatcher (Armstrong et al., 1995) and Letizia (Lieberman, 1997) are agents that learn to mimic the user by looking over his/her shoulder while browsing. Then they perform look-ahead searches and make real-time suggestions for pages that might interest the user. As with InfoSpiders, these agents learn to predict an objective function on-line; they can also track time-varying user preferences. Unlike InfoSpiders, however, WebWatcher and Letizia are single agents, and more importantly they need supervision from the user in order to work; no autonomous search is possible.

Fab (Balabanović, 1997) and Amalthaea (Moukas & Zacharia, 1997) are multi-agent adaptive filtering systems inspired by genetic algorithms, artificial life, and market models. Term weighting and relevance feedback are used to adapt a matching between a set of discovery agents (typically search engine parasites) and a set of user profiles (corresponding to single- or multiple-user interests). Like InfoSpiders, these systems can learn to divide the problem into simpler subproblems, dealing with the heterogeneous and dynamic profiles associated with long-standing queries. However these systems perform no active autonomous search, and therefore cannot improve on the scale limitations of the indexes they exploit.

Finally, the ecology-inspired InfoSpiders algorithm has been contrasted with a normative model for constructing browsing agents who make optimal local decisions about when to stop surfing, in much the same way in which real options are evaluated in financial markets (Lukose & Huberman, 1998). Such model is based on a different linkage topology assumption, in which the *value* of pages along the browsing path of a user follows a random walk of the form $V_L = V_{L-1} + \xi_L$ where L is the depth along the path and ξ_L is a random variable drawn from a normal distribution $\aleph(\mu, \sigma^2)$. This is stronger than our linkage conjecture, since it implies a positive correlation between V_L and V_{L-1} (analogous to our relevance autocorrelation) for any $\mu > 0$. Huberman et al. (1998) find that the distribution of surfing depth (clicks per Web site) derived from the above random walk equation is a very good predictive model of human browsing behavior. Although our conjecture on the value of linkage topology is more modest, it finds strong support in these findings.

8. Conclusion

8.1. Summary

Our results suggest that distributed, adaptive, on-line information browsing agents could complement current indexing technology by starting up where search engines stop. Engines provide global starting points, based on statistical features of the search space (words); agents can use topological features (links) to guide their subsequent search on-line. We have shown how this approach can improve on the current state of the art by dealing effectively with the scalability problem.

Our evaluation of the InfoSpiders collective performance provides us with encouraging support for the approach: the population can locate relevant documents in a large distributed corpus faster than best-first-search, taking advantage of its distributed model and implementation. We have shown elsewhere (Menczer, 1997; Menczer & Belew, 1998a) that InfoSpiders outperform exhaustive (breadth-first) search in this domain by an order

of magnitude, and that their performance can receive a boost from the synergy between individual learning and relevance feedback.

Our micro analysis enables us to determine that single agents can in fact internalize important local features of the environment into their internal representation, while the collective ecology captures a more heterogeneous snapshot of what features best correlate with user relevance. Agent representations and strategies evolve with time and change over an agent's lifetime; they are different from agent to agent depending on the temporal and spatial contexts in which they were born, and on what parts of the environment each has experienced.

8.2. Future directions

In our early experiments we have been most interested in the behavior of agents on a carefully controlled and structured corpus (EB). Therefore the full diversity we can reasonably expect from our agents as they interact with the real Web remains to be demonstrated. We have shown at least some divergence in the features that allow one agent to be successful within one topical area of the Encyclopedia and another, but the real purpose of open-ended evolutionary methods like those we propose is to adapt to the much wider variation found in the Web. We expect there to be roles for many different types of agents, sensitive to widely varying user demands, and effective at searching disparate corpora. Extensive evaluation of our approach on the actual Web, beyond the limited case study outlined above, is necessary to verify whether these goals can be met.

Many aspects of our model are to be explored at a greater detail in the near future. For instance, we have only begun to study the effect of caching and cache size on performance in the sequential InfoSpiders implementation (Menczer, 1998). The role of local, distributed caches in distributed implementations also needs further attention.

Our weighting scheme may be improved in several ways; for example, it has been suggested that the use of IDF in the local relevance estimation mechanism may be inappropriately biased toward global features (Srinivasan, 1998). The use of term weights in the mutation process could also be modified to allow for query expansion in the absence of relevance feedback, based solely on unsupervised correlation detection. This could be easily achieved by replacing the factor $|I_k|$ by $(\chi + |I_k|)$ in the probability distribution used for keyword mutation (cf. Section 5.3.4), where χ could be a fixed or evolved parameter (Menczer, 1998).

The only form of direct agent interaction that we have considered is crossover. An agent at reproduction can recombine its internal representation with that of a nearby agent, perhaps one situated on the same server. The two can internalize experiences that are now relevant to each other because of their proximity. Many other models of interaction among agents are also worth exploration in this domain. Agents learning from other agents, agent collaboration, and agent communication languages are all examples of very active research areas.

In the opposite direction, an important issue with respect to the practical implementations of InfoSpiders concerns the wasted effort in running the entire population of agents serially on a single, centralized user client. From the point of view of distributed implementations,

agent interactions must be kept at a minimum. This calls for further study of the interactions that are implicit in the current model, mainly centralized repositories for caching, resource sharing, and relevance feedback.

The InfoSpiders architecture anticipates a computing environment, already beginning to emerge (Pasquale, 1998), in which remote processes can be run on the hosts serving the documents. In this case, our simple agents need only perform a very light-weight "remote indexing" function. Documents will be scanned locally (on their remote servers) for a few textual features, and only those documents that appear relevant will be sent back to the user's client machine. As Internet bandwidth becomes more and more saturated, and as the communication overhead associated with the crawling activity of global search engines continues to grow, we believe such remote indexing strategies will become essential.

Finally, the feasibility of integrating agent-based on-line search with index-based search engines must be put to better test. The case study shows how to construct hybrid systems in which search engines provide agents with good staring points, based on the statistical (word-based) topology of the search space. This is crucial because, as we have shown in Section 6.2, the performance of on-line distributed search degrades with the distance between starting points and relevant clusters. The hypothesis that search engines can provide InfoSpiders with "good" starting points—within a certain distance from relevant pages—deserves empirical confirmation. If this is achieved, personal agents can continue the search on-line, adapting to both user and current environmental context. Using a population of autonomous browsing agents as a front-end to a search engine can help us better understand the mutual benefits of the two approaches and the potential synergies that may ensue.

Acknowledgments

The authors are grateful to Apple Computers for equipment donations and Encyclopaedia Britannica for making the Britannica CD collection available for the experiments described in this paper; the BCD data is ©1997 Encyclopaedia Britannica, Inc. Parts of the InfoSpiders code are ©1993 Free Software Foundation, Inc., ©1995 Massachusetts Institute of Technology, and ©1992–1997 Matthias Neeracher. We thank these sources for making such software available under the GNU General Public License. Daniel Clouse contributed a software library for associative arrays. Finally, we wish to thank David Lewis, Alvaro Monge, Charles Elkan, Russell Impagliazzo, the members of the Cognitive Computer Science Research Group in the CSE Department at UCSD, and four anonymous reviewers for helpful discussions and suggestions.

Notes

1. Several search engines now allow such queries for $k = 1$.
2. But a retrieved set could be viewed as the relevant set for *some* query.
3. Like search engine crawlers, on-line agents may carry out their search in real-time or be delayed until low-traffic hours, depending on user needs and network resources.
4. For Alta Vista, at the time of this writing we estimate $n/q\tau \approx 5$ (Digital Equipment Corporation. http://altavista.digital.com.); the condition will be met within a few years.
5. This methodology was suggested by van Rijsbergen (van Rijsbergen, 1979).

6. This list would typically be obtained by consulting a search engine.
7. Alternative crossover strategies are explored elsewhere (Menczer, 1998).
8. Other cost settings are explored elsewhere (Menczer, 1998).
9. These statistics do not account for the removal of relevant Propaedia subtree nodes.
10. The length of the priority queue is set equal to the initial InfoSpiders population size, p_0.
11. This is a worst-case scenario for InfoSpiders performance, since we can imagine that each agent could carry along a small local cache.

References

Armstrong, R., Freitag, D., Joachims, T., & Mitchell, T. (1995). Webwatcher: A learning apprentice for the world wide web. *AAAI Spring Symposium Information Gathering from Heterogeneous, Distributed Environments*.

Arocena, G.O., Mendelzon, A.O., & Mihaila, G.A. (1997). Applications of a web query language. *Proc. 6th International World Wide Web Conference*.

Balabanović, M. (1997). An adaptive web page recommendation service. *Proc. 1st International Conference on Autonomous Agents*.

Belew, R.K. (forthcoming). *Finding out about: Search engine technology from a cognitive perspective*. Cambridge University Press.

Belew, R.K. & Mitchell, M. (Eds.) (1996), *Adaptive individuals in evolving populations: models and algorithms*. Santa Fe Institute Studies in the Sciences of Complexity. Reading, MA: Addison Wesley.

Bollacker, K.D., Lawrence, S., & Giles, C.L. (1998). CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. *Proc. 2nd International Conference on Autonomous Agents*.

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

Chakrabarti, S., Dom, B., Gibson, D., Kumar, S.R., Raghavan, P., Rajagopalan, S., & Tomkins, A. (1998a). Experiments in topic distillation. *ACM SIGIR Workshop on Hypertext Information Retrieval on the Web*.

Chakrabarti, S., Dom, B., & Indyk, P. (1998b). Enhanced hypertext categorization using hyperlinks. *Proc. ACM SIGMOD*, Seattle, WA.

Chakrabarti, S., Dom, B., Raghavan, P., Rajagopalan, S., Gibson, D., & Kleinberg, J. (1998c). Automatic resource compilation by analyzing hyperlink structure and associated text. *Proc. 7th International World Wide Web Conference*.

Cohen, W.W. (1996). Learning trees and rules with set-valued features. *Proc. 13th AI Conference* (pp. 709–716).

Cooper, W.S. (1968). Expected search length: A single measure of retrieval effectiveness based on weak ordering action of retrieval systems. *Journal of the American Society for Information Science, 19*, 30–41.

Croft, W.B. & Turtle, H.R. (1993). Retrieval strategies for hypertext. *Information Processing and Management, 29*(3), 313–324.

De Bra, P.M.E. & Post, R.D.J. (1994). Information retrieval in the world wide web: Making client-based searching feasible. *Proc. 1st International World Wide Web Conference*, Geneva.

De Jong, K.A. (1975). An analysis of the behavior of a class of genetic adaptive systems. Ph.D. Thesis, University of Michigan.

De Jong, K.A. & Sarma, J. (1995). On decentralizing selection algorithms. *Proc. 6th International Conference on Genetic Algorithms*.

Digital Equipment Corporation. http://altavista.digital.com.

Eichmann, D. (1994). The RBSE spider—Balancing effective search against Web load. *Proc. 1st International World Wide Web Conference*.

Encyclopaedia Britannica, Inc. http://www.eb.com.

Excite. http://www.excite.com.

Fox, C. (1992). Lexical analysis and stop lists. *Information retrieval: Data structures and algorithms*. Prentice-Hall.

Frakes, W.B. (1992). Stemming algorithms. *Information retrieval: Data structures and algorithms*. Prentice-Hall.

Goldberg, D.E. & Richardson, J. (1987). Genetic algorithms with sharing for multimodal function optimization. *Proc. 2nd International Conference on Genetic Algorithms*.

Harman, D. (1992). Relevance feedback and other query modification techniques. *Information retrieval: Data structures and algorithms*. Prentice-Hall.

Hart, W.E. & Belew, R.K. (1996). Optimization with genetic algorithm hybrids that use local search. *Adaptive individuals in evolving populations: Models and algorithms*. Addison Wesley.

Huberman, B.A., Pirolli, P.L.T., Pitkow, J.E., & Lukose, R.M. (1998). Strong regularities in world wide web surfing. *Science*, *280*(5360), 95–97.

Koutsoupias, E., Papadimitriou, C., & Yannakakis, M. (1996). Searching a fixed graph. *Proc. 23rd International Colloquium on Automata, Languages and Programming*, Berlin, Germany (pp. 280–289). Springer-Verlag.

Kwok, K.L. (1988). On the use of bibliographically related titles for the enhancement of document representations. *Information Processing and Management*, *24*(2), 123–131.

Larson, R.R. (1996). Bibliometrics of the world wide web: An exploratory analysis of the intellectual structure of cyberspace. *Proc. 1996 Annual ASIS Meeting*.

Lawrence, S.R. & Giles, C.L. (1998). Searching the world wide web. *Science*, *280*, 98–100.

Lewis, D.D. (1995). Active by accident: Relevance feedback in information retrieval. *AAAI Fall Symposium on Active Learning*.

Lewis, D.D. (1996). Information retrieval and the statistics of large data sets. *Proc. NRC Massive Data Sets Workshop*, Washington, DC.

Lewis, D.D. (1997). Challenges in machine learning for text classification. *Proc. 9th Annual Conference on Computational Learning Theory*, New York, NY.

Lewis, D.D. & Hayes, P.H. (1994). Special issue on text classification (guest editorial). *ACM Transactions on Information Systems*, *12*(3), 231.

Lewis, D.D., Schapire, R.E., Callan, J.P., & Papka, R. (1996). Training algorithms for linear text classifiers. *Proc. ACM SIGIR* (pp. 298–306).

Lieberman, H. (1997). Autonomous interface agents. *Proc. ACM Conference on Computers and Human Interface*, Atlanta, GA.

Lin, L.-J. (1992). Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8, 293–321.

Lukose, R.M. & Huberman, B.A. (1998). Surfing as a real option. *Proc. 4th Intl. Conf. on Computational Economics*.

Lycos. http://www.lycos.com.

Mahfoud, S.W. (1994). Population sizing for sharing methods. *Foundations of Genetic Algorithms 3*.

Mahfoud, S.W. (1995). A comparison of parallel and sequential niching methods. *Proc. 6th International Conference on Genetic Algorithms*.

Menczer, F. (1997). ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery. *Proc. 14th International Conference on Machine Learning*.

Menczer, F. (1998). Life-like agents: Internalizing local cues for reinforcement learning and evolution. Ph.D. Thesis, University of California, San Diego.

Menczer, F. & Belew, R.K. (1996). Latent energy environments. *Adaptive individuals in evolving populations: Models and algorithms*. Addison Wesley.

Menczer, F. & Belew, R.K. (1998a). Adaptive information agents in distributed textual environments. *Proc. 2nd International Conference on Autonomous Agents*, Minneapolis, MN.

Menczer, F. & Belew, R.K. (1998b). Local selection. *Proc. 7th Annual Conference on Evolutionary Programming*, San Diego, CA.

Menczer, F. & Monge, A.E. (1999). Scalable web search by adaptive online agents: An InfoSpiders case study. In M. Klusch (Ed.), *Intelligent information agents: Agent-based Information discovery and management on the internet*. Springer.

Menczer, F., Willuhn, W., & Belew, R.K. (1994). An endogenous fitness paradigm for adaptive information agents. *CIKM Workshop on Intelligent Information Agents*.

Mitchell, T. (1997). *Machine Learning* (Ch. 4). New York, NY: McGraw-Hill.

Monge, A.E. & Elkan, C.P. (1996). The WEBFIND tool for finding scientific papers over the worldwide web. *Proceedings of the 3rd International Congress on Computer Science Research*.

Moukas, A. & Zacharia, G. (1997). Evolving a multi-agent information filtering solution in amalthaea. *Proc. 1st International Conference on Autonomous Agents*.

Pack Kaelbling, L., Littman, M.L., & Moore, A.W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237–285.

Pasquale, J. (1998). The UCSD Active Web. http://www.cs.ucsd.edu/Department/Infra.

Pinkerton, B. (1994). Finding what people want: Experiences with the Webcrawler. *Proc. 1st International World Wide Web Conference*.

Rivlin, E., Botafogo, R., & Shneiderman, B. (1994). Navigating in hyperspace: Designing a structure-based toolbox. *Communications of the ACM*, *37*(2), 87–96.

Robertson, S.E. & Spark Jones, K. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science* (pp. 129–146).

Rumelhart, D.E., Hinton, G.E., & Williams, R.J. (1986). Learning internal representations by error propagation. In D.E. Rumelhart & J.L. McClelland (Eds.), *Parallel distributed processing: Explorations in the microstructure of cognition* (Vol. 1). Cambridge MA: Bradford Books (MIT Press).

Salton, G. (1963). Associative document retrieval techniques using bibliographic information. *Journal of the ACM*, *10*(4), 440–457.

Salton, G. & McGill, M.J. (1983). *An introduction to modern information retrieval*. New York, NY: McGraw-Hill.

Savoy, J. (1996). An extended vector processing scheme for searching information in hypertext. *Information Processing and Management*, *32*(2), 155–170.

Shakes, J., Langheinrich, M., & Etzioni, O. (1997). Dynamic reference sifting: A case study in the homepage domain. *Proc. 6th International World Wide Web Conference*.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, *28*, 111–121.

Sparck Jones, K. (1979). Experiments in relevance weighting of search terms. *Information Processing and Management*, *15*, 133–144.

Spertus, E. (1997). Parasite: Mining structural information on the web. *Proc. 6th International World Wide Web Conference*.

Srinivasan, P. (1998). Personal communication.

Steier, A.M. (1994). Statistical semantics of phrases in hierarchical contexts. Ph.D. Thesis, Computer Science and Engineering Department, U.C. San Diego.

Steier, A.M. & Belew, R.K. (1994). Exporting phrases: A statistical analysis of topical language. In R. Casey & B. Croft (Eds.), *2nd Symposium on Document Analysis and Information Retrieval*.

Sutton, R. (1996). Reinforcement learning and information access. *AAAI Spring Symposium on Machine Learning and Information Access*.

van Rijsbergen, C.J. (1979). *Information Retrieval*, second edition, London: Butterworths.

Weiss, R., Velez, B., Sheldon, M., Nemprempre, C., Szilagyi, P., & Giffor, D.K. (1996). Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering. *Proc. Seventh ACM Conference on Hypertext*.

Received April 13, 1998
Accepted July 22, 1999
Final manuscript July 22, 1999