1. Scalable Web Search by Adaptive Online Agents: An InfoSpiders Case Study

Filippo Menczer[1] and Alvaro E. Monge[2]

[1] Management Sciences Department
University of Iowa
Iowa City, IA 52245, USA
email: filippo-menczer@uiowa.edu

[2] Computer Science Department
University of Dayton
Dayton, OH 45469, USA
email: monge@cps.udayton.edu

The trend of the recent years in distributed information environments is a good example of the *life-like* complexity that we expect to observe in most aspects of information and computational science. The explosion of the Web and electronic mail, multiplying the number of information providers and consumers many times over and bringing the Internet inside the average home, has created formidable new opportunities and challenges in almost every area of computer and information science.

In an effort to address such problems, researchers in artificial intelligence and information retrieval have already been successful in developing agent-based techniques to automate many tedious tasks and facilitate the management of the growing amounts of information flooding users. But the work has just begun. There is still much need for tools to assist users in ways that scale with the growth of the Web, and adapt to both the personal preferences of the user and the changes in user and environmental conditions.

This chapter discusses an agent-based approach to building scalable information searching algorithms. For systems designed to let users locate relevant information in highly distributed and decentralized databases, such as the Web, we argue that *scalability* is one of the main limitations of the current state of the art. Given such an ambitious goal, it probably comes as no surprise that the solution proposed here draws on many ideas and issues discussed in other parts of this book: cooperation in multi-agent systems, information chain economy in rational agents, and spawning and security in mobile agents.

1.1 Introduction

The complexities emerging in networked information environments (decentralization, noise, heterogeneity, and dynamics) are not unlike those faced

by ecologies of organisms adapting in natural environments. The capabilities of such *natural* agents — local adaptation, internalization of environmental signals, distributed control, integration of externally driven and endogenous behaviors, etc. — represent desirable goals for the next generation of *artificial* agents: autonomous, intelligent, distributed, and adaptive. These considerations, along the lines of the artificial life approach, inspired us to base our model upon the metaphor of an *ecology* of agents.

In this sense, the *multi-agent* system is not composed of a few agents with distinct and clearly defined functions, but rather by a (possibly very) large number of agents collectively trying to satisfy the user request. The number of agents is determined by the environment, in turn shaped by the search task. This does not mean that all agents responding to a specific search are identical; each will adapt to both the local context set by the environment and the global context set by the user.

Cooperation results from the indirect interaction among agents, mediated by the environment. If there are sufficient resources to sustain multiple agents in a given environmental neighborhood, then new agents will spawn and collaborate with the existing ones. If resources are scarce, on the contrary, the agents will compete and some of them will be eliminated.

The ecological metaphor thus induces a rational use of the information resource chain. Computational resources (CPU time) and network resources (bandwidth) are allocated in proportion to the recently perceived success of an agent, estimated from the relevance of the consumed information resources. Ideally, each agent would browse the Web neighborhood in which it is situated like its human master would, given her finite resources — time and attention.

The approach discussed in this chapter assumes that, in exchange for improved bandwidth or payments, servers may allow trusted mobile agents to execute and possibly even spawn new agents using their hardware (CPU, memory, disk storage) in a controlled operating environment. If such an assumption holds, agents can take advantage of the distributed nature of our algorithm. They can execute in a parallel, asynchronous fashion, resulting in a great potential speedup. Security and distributed systems issues are central to the implementation of such mobile agents. If the assumption fails, however, the agents can still execute in a client-based fashion. The algorithm becomes essentially sequential (although possibly multi-threaded), and thus simpler to implement. The decrease in performance can be partially offset by the use of a central cache. While the distributed execution of our mobile agents has been simulated successfully, the case study illustrated in this chapter assumes a client-based implementation.

The next section provides background on the current state of the art in information gathering from networked and distributed information sources. To be sure, the field is evolving very rapidly and agent research is playing a key role in generating the technological advances that may soon allow us to tame

the complexity of the Web. Therefore we have no aspiration at completeness, but merely intend to set a stage in which to identify, in the following section, the limitations of the current methodologies. We then give an overview of the *InfoSpiders* system, an implementation that embodies the approach suggested here to complement the scale limitations of search engines. InfoSpiders have been introduced and described elsewhere [27, 29, 30, 28], so our purpose is to summarize those aspects of the model that are relevant to the issue of scalability. Section 1.5 illustrates through a case study how these adaptive online search agents can complement search engines to achieve scalability. The chapter concludes with an outline of InfoSpiders' previous results and a discussion of the possible applications of the InfoSpiders approach.

1.2 Engines and agents

Exploiting the proven techniques of information retrieval, search engines have followed the growth of the Web and provided users with much needed assistance in their attempts to locate and retrieve information from the Web. Search engines have continued to grow in size, efficiency, performance, and diversity of services offered. Their success is attested by both their multiplication 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 the word weighting schemes.

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. Indices are thus reduced to "snapshots" of the Web. They are continuously updated by *crawlers* that attempt to exhaustively visit and periodically revisit every Web page. At any given time an index will be somewhat inaccurate (e.g., contain stale information about recently deleted or moved documents) and somewhat incomplete (e.g., missing information about recently added or changed documents).

The above observations are quantified in Sect. 1.3. The problem, compounded by the huge size of the Web, is one of scalability. As a result, search engines' capability to satisfy user queries is hindered. Users are normally faced with very large hit lists, low *recall* (fraction of relevant pages that are retrieved), even lower *precision* (fraction of retrieved pages that are relevant),

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.

A way to partially address the scalability problems posed by the size and dynamic nature of the Web is by decentralizing the index-building process. 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 [5]. The success of this approach has been hindered by the need for cooperation between information providers and indexing crawlers.

A step toward enriching search engines with topological information about linkage to achieve better precision has been suggested by the CLEVER group at IBM Almaden Research Labs. The idea is to use hyperlinks to construct "hub" and "authority" nodes from the Web graph and it has proven effective in improving document retrieval and classification performance [7, 6].

Autonomous agents, or semi-intelligent programs making automatic decisions on behalf of the user, are viewed by many as a way of decreasing the amount of human-computer interaction necessary to manage the increasing amount of information available online [26]. Many such information agents, more or less intelligent and more or less autonomous, have been developed in the recent years. The great majority of them suffer from a common limitation: their reliance on search engines. The limited coverage and recency of search engines cannot be overcome by agents whose search process consists of submitting queries to search engines. However, many agents partially improve on the quality of any search engine's performance by submitting queries to many different engines simultaneously. This technique, originally called *metasearch* [32], has indeed proven to increase recall significantly [42].

Typical examples of agents who rely on search engines to find information on behalf of the users are homepage or paper finders. CiteSeer [4] is an autonomous Web agent for automatic retrieval and identification of publications. Ahoy [38] is a homepage finder based on metasearch engine plus some heuristic local search. WebFind [33] 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 provided their starting points, and usually limited to servers known to them.

A different class of agents are designed to learn user interests from browsing for recommendations purposes. Syskill & Webert [34] is a system that identifies interesting Web sites from large domain-specific link lists by learning to rate them based on relevance feedback. WebWatcher [1, 18] is a tour guide agent that learns from experience of multiple users by looking over their shoulders while browsing. Then it provides users with suggestions about what links to follow next. Similarly, Letizia (Chap. 12) is an autonomous interface

agent that assists the user in browsing the Web by performing look-ahead searches and making real-time recommendations for nearby pages that might interest the user. WebMate [8] assists browsing by learning user preferences in multiple domains, and assists searching by automatic keyword extraction for query refinement. All these agents learn to predict an objective function online; they can also track time-varying user preferences. However, they need supervision from the user in order to work; no truly autonomous search is possible.

Amalthaea (Chap. 13) and Fab [3] share many features with the model described in this chapter. They 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). These systems can learn to divide the problem into simpler subproblems, dealing with the heterogeneous and dynamic profiles associated with long-standing queries. However they share the weak points of other agents who perform no active autonomous search, and therefore cannot improve on the limitations of the metasearch engines they exploit.

Fish Search [10] is a search system inspired by some of the same ideas from artificial life that motivated the research in this chapter. Fish Search is based on a population of search agents who browse the Web autonomously, driven by an internally generated energy measure based on relevance estimations. The population is client-based, and uses a centralized cache for efficiency. Each agent has 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 is in determining appropriate cutoff levels *a priori,* possibly resulting in load-unfriendly search behaviors. Therefore Fish Search suffers from limitations that are in a sense opposite to those of all the previously discussed agents; all it does is search, but it cannot adapt to user or environmental conditions.

1.3 Scalability

As was discussed in the previous section, 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 rate, and its highly dynamic nature. The scalability problem is quantified in a recent study by Lawrence and Giles [20]. Their estimates of the current size (over 320 million pages) and growth rate (1000% in a few years) of the Web attest to this environment's increasing complexity.

Lawrence and Giles also measure the *coverage* and *recency* of six among the most popular search engines. The coverage achieved by these search engines varies approximately between 3% and 34% of the Web's indexable pages.

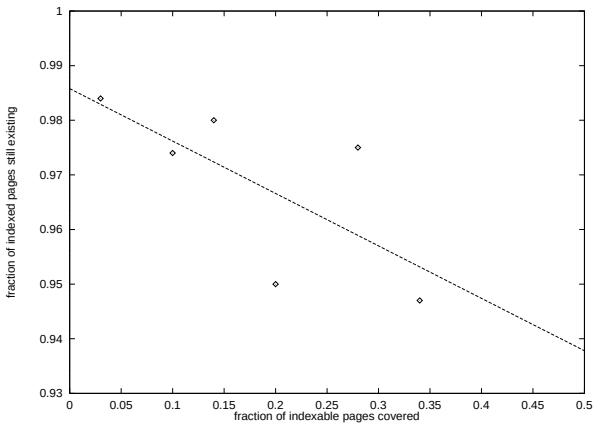

Fig. 1.1. Scatter plot of coverage versus recency in six popular search engines: Alta Vista, HotBot, Northern Lights, Excite, InfoSeek, and Lycos. Data from [20]. Linear regression is also shown. The correlation coefficient is -0.7.

An estimate of recency was obtained by counting the fraction of returned hits corresponding to broken URLs, i.e., pages that have been deleted or moved.[1] Among the search engines considered, the one with highest coverage also has the most broken links (5%), and vice versa — the engine with lowest coverage is the one with highest recency. Such a trade-off between coverage and recency is illustrated in Fig. 1.1. Coverage and recency are indeed anti-correlated, as expected. Increasing the coverage of an index, given some limited bandwidth resource, imposes a search engine's crawler to "spread itself thin" and update pages less frequently, thus increasing the amount of stale information in the index.

In order to keep indices as up-to-date as possible, crawlers have to revisit documents often to see if they have been changed, moved, or deleted. Further, crawlers have to try to exhaustively visit every new document to keep indices as complete as possible. Such crawler behaviors impose significant loads on the net, as documents must be examined periodically. 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. The network load scales as n/τ, where n is the number of documents in the Web and τ is the time scale of the index, i.e. the mean time between visits to the same document. The longer τ, the more stale information found 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 *online* do not have a scalability problem because they search through the *current* environment and therefore do not run into stale information. On the other hand, they are less efficient than search engines because they cannot amortize the cost of a search over many queries. Assuming that users may be willing to cope with the longer wait for certain

[1] URLs with changed content do not appear broken, therefore this method measures a lower bound on the amount of stale information in an index.

queries that search engines cannot answer satisfactorily, one might ask, *What is the impact of online search agents on network load?*

In our opinion, because of the scalability effect, making an index less up-to-date can free up sufficient network resources to completely absorb the impact of online 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 online 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 online 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. For Alta Vista, we recently estimated $n/q\tau \approx 5$ [11, 28]; even at a conservative growth rate of a doubling per year, the condition would be met within at most 5 years.[2] This simple argument, in our opinion, shifts the question: we should not ask what is the network impact of online search agents, but rather, *What ϵ achieves an appropriate balance between the network loads imposed by search engines' crawlers and online agents?*

We make the assumption in this chapter that as the Web continues to grow and its dynamics become even more life-like, users will increasingly rely on personalized tools in addition to global search engines. Under this assumption, we envision that the relative load of the network will shift from "dumb" crawlers to "smart" browsing agents, while users will develop a more distal relationship with the information sources by way of trusted (agent) intermediaries. But this vision can become reality only when agents will offer the capability to add the value of scalability to search engines. We must therefore prove that an agent-based solution can indeed reach beyond search engines and effectively locate information unknown to them.

1.4 InfoSpiders

Let us operationalize the ideas discussed in the previous section into an agent framework. The goal addressed in this chapter is to achieve scalable Web search by complementing search engines with an agent-based algorithm. The agent collective is endowed with a distributed adaptive representation, aiming to take advantage of both the statistical (word) and structural (link)

[2] If we consider the coverage factor of 3 due to the discrepancy between the n of the search engine and the actual size of the Web, the condition will be met a lot sooner.

topology of the Web. We have argued that the agents in such collective must be *autonomous, online, situated, personal browsers* [30].

Our approach is therefore 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 online, 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 might be prone to premature convergence. Genetically evolved solutions would also reflect an inappropriate coarseness of scale, due to individual agents' incapability to learn during their lives. Incidentally, these are the same reasons that have motivated the hybridization of genetic algorithms with local search [16], and reflect the general problem of machine learning techniques in environments with very large feature space dimensionalities [21, 22].

The approach and methods introduced above have been applied in the construction of populations of adaptive information agents. The InfoSpiders system was implemented to test the feasibility, efficiency, and performance of adaptive, online, browsing, situated, personal agents in the Web. In this section we outline the InfoSpiders implementation and briefly describe the distributed evolutionary algorithm and agent representation used. A more detailed account can be found elsewhere [28].

1.4.1 Algorithm

Distributed search in networked environments is a multimodal problem that presents many of the characteristics making it an ideal target for *local selection* algorithms [31]. This task requires a heterogeneous cover of the search space rather than a convergence to the perceived global optimum. Indeed it can easily be cast into a graph search framework, in which local selection algorithms have proven very effective [31, 28].

InfoSpiders search online for information relevant to the user, by making autonomous decisions about what links to follow. 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 would appear satisfactory.

```
InfoSpiders(query, starting_urls, MAX_PAGES, REL_FBK_FLAG) {
   for agent (1..INIT_POP) {
      initialize(agent, query);
      situate(agent, starting_urls);
      agent.energy := THETA / 2;
   }
   while (pop_size > 0 and visited < MAX_PAGES) {
      foreach agent {
         pick_outlink_from_current_document(agent);
         agent.doc := fetch_new_document(agent);
         agent.energy +=  benefit(agent.doc) - cost(agent.doc);
         apply_Q_learning(agent, benefit(agent.doc));
         if (agent.energy >= THETA) {
            offspring := mutate(recombine(clone(agent)));
            offspring.energy := agent.energy / 2;
            agent.energy -= offspring.energy;
            birth(offspring);
         }
         elseif (agent.energy <= 0) death(agent);
      }
      if (REL_FBK_FLAG) process_relevance_feedback(input);
   }
}
```

Fig. 1.2. Pseudocode of the InfoSpiders algorithm for distributed information agents. This is an instance of an evolutionary algorithm based on local selection.

Fortunately, the local selection algorithm provides us with a way to remain agnostic about these questions. Such an algorithm is shown in Fig. 1.2.

The user initially provides a list of keywords (query) and a list of starting points, in the form of a bookmark file. This list could typically be obtained by consulting a search engine. First, 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. The user also provides a maximum number of pages that the population of agents are allowed to visit, collectively. This would depend on how long the user is willing to wait, or how much bandwidth she is willing to consume. Finally, the user may specify whether and how often she is willing to provide the population with relevance feedback, to help focus or shift the search toward relevant areas by "replenishing" the resources in those areas. This chapter does not discuss the use of relevance feedback in depth, as relevance feedback is not used in the reported case study.

In the innermost loop of Fig. 1.2, an 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, the agent "moves" by choosing and following one of the links from the current document.

The agent's energy is then 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 `benefit()` function is used by an agent to estimate the relevance of documents. In the absence of relevance feedback, this function return a non-zero value only if the document had not been previously visited by any agent, and according to a standard measure of similarity between the query and the document. The "marking" of visited pages models the consumption of finite information resources and is implemented via a cache, which also speeds up the process by minimizing duplicate transfers of documents.[3]

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

Instantaneous changes of energy are used as reinforcement signals. This way agents adapt during their lifetime by Q-learning [44]. This adaptive process allows an agent to modify its behavior based on prior experience, by learning to predict the best links to follow.

Local selection means that an agent is selected for reproduction based on a comparison between its current energy level and a constant that is independent of the other agents in the population. Similarly, an agent is killed when it runs out of energy. At reproduction, agents may be recombined by the use of one of two types of crossover. In the case study illustrated later in the chapter, an agent may recombine with any other agent in the population, selected at random. Offspring are also mutated, providing the variation necessary for adapting agents by way of evolution. Energy is conserved at all reproduction events.

The output of the algorithm is a flux of links to documents, ranked according to estimated relevance. The algorithm stops when the population goes extinct for lack of relevant information resources, visits `MAX_PAGES` documents, or is terminated by the user.

1.4.2 Agent architecture

Figure 1.3 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

[3] While in the current client-based implementation of InfoSpiders this poses no problem, caching is a form of communication and thus a bottleneck for the performance of distributed agents. In a distributed implementation, we imagine that agents will have local caches.

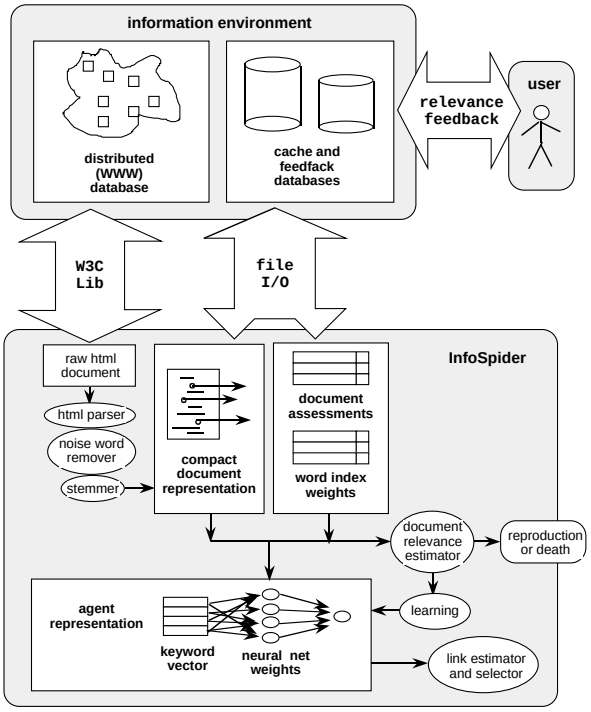

Fig. 1.3. Architecture of an InfoSpiders agent. From [30].

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 after the initial submission of the query and starting points.

The InfoSpiders prototype runs on UNIX and MacOS platforms. The Web interface is based on the `W3C` library [43]. Agents employ standard information retrieval tools such as a filter for noise words [14] and a stemmer based on Porter's algorithm [15]. They store an efficient representation of visited documents in the shared cache on the client machine. Each document is represented by a list of links and stemmed keywords. If the cache reaches its size limit, the LRU (least recently used) replacement strategy is used. In the case study illustrated later in the chapter, the cache can grow arbitrarily.

1.4.3 Adaptive representation

Figure 1.3 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

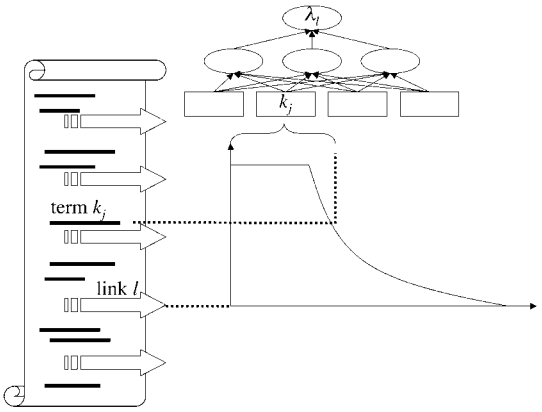

Fig. 1.4. How an agent estimates each link from the current document. For each link in the document, each input of the neural net is computed by counting the document words matching the keyword corresponding to that input, with weights that decay with distance from the link.

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 or 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 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 $[-1, +1]$.

An agent performs action selection by first computing the relevance estimates for each outgoing 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:

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

where k_i is the ith occurrence of k in D and $\mathrm{dist}(k_i,l)$ is a simple count of other, intervening links (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 activation

$$\tanh(b_j + \sum_k w_{jk} in_k^l)$$

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 Fig. 1.4 and is repeated for each link in the current document. Then, the agent uses a stochastic selector to pick a link with probability distribution:

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

After a link has been chosen and the corresponding new document has been visited, the agent has to determine the corresponding energy gain. For a previously unvisited document,

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

where $\mathrm{freq}(k,D)$ is the frequency of term k in document D normalized by document size, and I_k is the weight of term k. In the absence of relevance feedback, $I_k = 1$ if k is in the query and $I_k = 0$ otherwise.[4]

The agent then compares the (estimated) relevance of the current document with the estimate of the link that led to it. By using the connectionist version of Q-learning [23], the neural net can be trained online to predict values of links based on local context. After the agent visits document D, the value returned by the `benefit()` function is used as an internally generated reinforcement signal to compute a teaching error:

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

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 [36]. Learned changes to the weights are "Lamarckian" in that they are inherited by offspring at reproduction. In the absence of relevance feedback this learning scheme is completely unsupervised, in keeping with the autonomy of InfoSpiders.

[4] If the user provides relevance assessments, I_k becomes an algebraic extension of the TFIDF (term frequency-inverse document frequency) index weighting scheme, allowing for negative relevance feedback and consequent energy losses.

InfoSpiders adapt not only by learning neural net weights, but also by evolving all of the genotype components — β, the neural net, and the keyword representation. At reproduction, the offspring clone is 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 mutations are applied. If a' is an offspring of a:

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

The values of β are clipped to $\beta_{\max}$ to maintain some exploratory behavior. The neural net is mutated by adding random noise to a fraction ζ_w of the weights. For each of these network connections, i:

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

U is the uniform distribution and $\kappa_\beta, \kappa_w \in [0,1]$ are parameters.

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. Ties are broken randomly. 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

$$\begin{aligned} \Pr[k] &\propto \mathrm{freq}(k, D) \cdot \delta_{<1}(|I_k| + \chi) \\ \delta_{<1}(x) &\equiv \begin{cases} x & \text{if } x < 1 \\ 1 & \text{otherwise} \end{cases} \end{aligned}$$

where D is the document of birth and $\chi \in [0,1]$ is a parameter. The first factor captures the local context by selecting a word that describes well the document that led to the energy increase resulting in the reproduction. The second factor, in the presence of relevance feedback, captures the global context set by the user by selecting a word that discriminates well the user's preferences. The parameter χ regulates the amount of supervised (small χ) versus unsupervised (large χ) keyword mutation; if $\chi = 0$, only keywords important to the user can be internalized, while if $\chi > 0$ new keywords can be internalized based on local environmental context alone. 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 can adapt over time and across different places. The population of agents thus 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.

1.5 Case study

In this section we want to illustrate how scalability in Web search can be achieved using InfoSpiders as a front-end to a traditional search engine. We do this with a case study, corresponding to a query that search engines cannot satisfy alone.

Table 1.1. InfoSpiders parameter descriptions and values.

Parameter	Value	Description
`MAX_PAGES`	200	Max number of new pages visited per query
`REL_FBK_FLAG`	`FALSE`	Relevance feedback disabled
`INIT_POP`	10	Initial population size
`THETA`	2.0	Reproduction threshold
`CACHE_SIZE`	`MAX_PAGES`	Effectively unbounded cache
c_n	0.01	Energy cost per new document
c_o	0.0001	Energy cost per cached document
β_0	4.0	Initial β
κ_β	0.5	β mutation range
$\beta_{\max}$	5.0	Max β
ρ	5	Half-size of link estimation sliding window
ζ_k	0.0	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

In order to keep the analysis of our case study as simple as possible, agents are disallowed to differentiate on the basis of keyword vectors; all InfoSpiders focus on the same terms over time. The neural nets are allowed to adapt by evolution and reinforcement learning, and the β gene can evolve as well. The cache size is large enough to contain all the visited documents. Table 1.1 shows the values of the main parameters discussed in Sect. 1.4 and their default values used in the case study.

1.5.1 Search engine defeat

Pat is a prospective student in a database class offered at the University of Iowa, and has heard from a friend the name of the instructor who will be teaching the course. Pat wants to find out about the class in order to make an informed decision before registering for the class. Having access to the Web, Pat submits a query to a search engine, say, Excite [13]. The query is reasonable: `SYLLABUS OF DATABASE CLASS PROF FILIPPO`

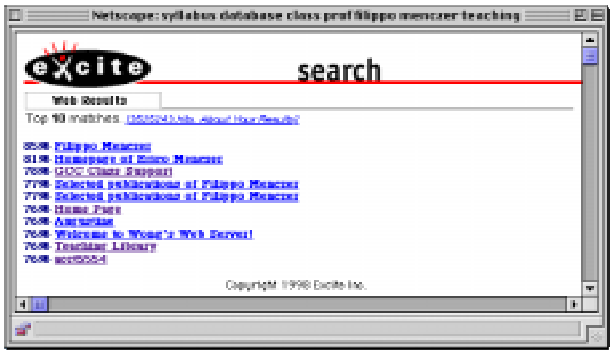

Fig. 1.5. The top ten hits returned by Excite in response to Pat's query and fed to InfoSpiders as a list of starting URLs. The HTML source file has been edited for readability.

`MENCZER IS TEACHING.` Unfortunately, Excite does not return any page that seems relevant.[5]

In fact, the class syllabus has been available on the Web for a couple of weeks already, and there is a pointer to it from the homepage of the department offering the course. However, because of the limited coverage and recency of search engines, the three documents with the information sought by Pat (the relevant set) have not yet been found by any crawler and therefore the pages are not indexed in any search engine. All major search engines, including MetaCrawler [32], return large hit lists with zero precision and zero recall.

1.5.2 InfoSpiders to the rescue

For the purpose of this case study, we assume that Pat is a very busy student who cannot afford to spend much time manually surfing the net in search of class information. Before giving up, however, Pat decides to try launching a search with the InfoSpiders tool that (coincidence!) is available on the local computer. InfoSpiders need a list of starting points, in the form of a bookmark or hit list file. Pat uses the first page of hits returned by Excite, shown in Fig. 1.5.

Ten InfoSpiders are initialized, one at each of the top ten URLs returned by Excite. They all share the same keyword vector, whose components correspond to the query terms (after filtering out noise words and stemming). The log file created during the InfoSpiders search is partially shown in Fig. 1.6. It shows that some of the starting points are perceived as dead ends by InfoSpiders, and discarded.[6] This happens, for example, for broken links yielding a "404 Not Found" error. Such dead ends need to be discarded only

[5] Unbeknownst to Pat, none of the other major search engines would have done any better.

[6] The crude parser in the current prototype fails to recognize relative URLs, as well as anchors with incorrect syntax.

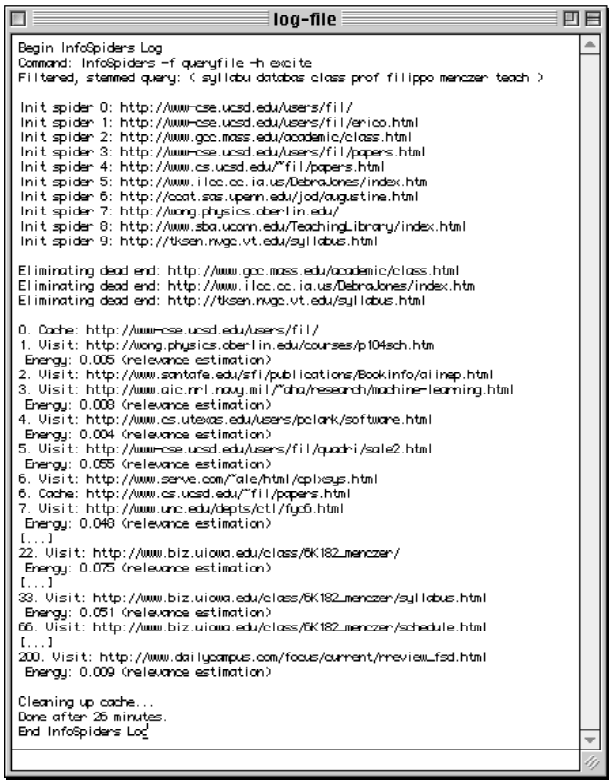

Fig. 1.6. InfoSpiders log file for the search launched by Pat. The file has been edited for brevity. The relevant URLs appear as entries 22, 33, and 66 in the log sequence.

from the starting points; later on they pose no problem because InfoSpiders can always follow a special back link.

After fetching the starting URLs, InfoSpiders begin to browse autonomously. Certain pages contain the query words and yield energy used by the agents to prolong their survival in those neighborhoods. Eventually, after 66 pages have been visited, InfoSpiders locate all three relevant documents. This takes less than 9 minutes of search time. Of course InfoSpiders do not know the size of the relevant set, and continue to search as long as they live (or until, in this case, 200 pages are visited over night).

The next morning, Pat finds the report of the InfoSpiders search. The result is shown in Fig. 1.7. The visited pages are ranked by estimated relevance — in this case simply by their similarity to the query; the cosine matching score is listed next to each visited page. Two of the three relevant documents (those containing query terms) are ranked in the top ten positions (3 and 7) while the third appears later in the list.

By combining the starting points provided by the search engine and the online search provided by InfoSpiders, Pat has found all of the needed information and decides to enroll in the class. This result could not have been

Fig. 1.7. Report of the InfoSpiders search launched by Pat. The original HTML document created by InfoSpiders has been edited for brevity and is viewed through a browser.

reached through search engines alone, at the time when Pat's information need arose. It could have been achieved by manually browsing through the top pages returned by Excite, but this is a very time-consuming activity — one better delegated to intelligent information agents!

1.6 Discussion

If InfoSpiders overcome the scalability limitation of search engines, why use search engines at all? We want to briefly discuss the issue of topology, in support of our view that the two approaches are really complementary to each other, and either one alone is insufficient to achieve scalable search. In this section we also summarize some more quantitative results previously obtained by InfoSpiders, since the case study described above is merely intended as a qualitative illustration of the scalability argument. We conclude with a look at the future.

1.6.1 Links vs. words

Indexing can be described as the process of building a *statistical topology* over a document space. 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, networked information environments contain additional structure information, which can be used to provide browsing users (or agents) with helpful cues. Here we focus on linkage information that is at the basis of hypertext markup languages such as those used in the Web. 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 would scale exponentially with k.[7]

While much linkage information is lost in the construction of indices, it is there to be exploited by browsing users, who in fact navigate from document to document following links. We have argued 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, as confirmed by bibliometric studies of the Web [19]. 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) [28]. Imagine a browsing user or agent following a random walk strategy.[8] First define R as the conditional probability that following a random link from the current document will lead to a relevant document, given that the current document is relevant. We call R *relevance autocorrelation*. Then define G as the probability that any document is relevant, or equivalently the fraction of relevant documents. We call G *generality* (of the query) [37].

For the random browser, the probability of finding a relevant document is given by

$$\nu = \eta R + (1 - \eta) G \ ,$$

where η is the probability that the current document is relevant. 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

$$\nu / G > 1$$

must hold, which upon simplifying for η is equivalent to

$$R / G > 1 \ .$$

[7] Several search engines now allow such queries for $k = 1$.

[8] We make the conservative assumption of random walk to obtain a lower bound for the value added of linkage topology.

This linkage topology conjecture is equivalent to the cluster hypothesis [41] under a hypertext derived definition of association. We can then express the linkage topology value added by defining the quantity

$$\Theta \equiv R/G - 1 \ .$$

As a reality check, we measured Θ for a few queries from a couple of search engines [27]. Relevance autocorrelation statistics were collected by counting the fraction of links, from documents in each relevant set, pointing back to documents in the set. Generality statistics were collected by normalizing the size of the relevant sets by the size of the collections. These were quite gross measurements, but their positive values support our conjecture about the value added by linkage topology: in Lycos [25], for example, we found $\Theta = (9 \pm 3) \times 10^3 \gg 0$.

Linkage topology also has been considered by others in the context of the Web, with different motivations. Links have been used for enhancing relevance judgments [35, 45], incorporated into query formulation to improve searching [2, 39], and exploited to determine "hub" and "authority" pages for document categorization and discovery [7, 6].

If links constitute useful cues for navigation, they can be exploited by autonomous browsing agents just as they are by browsing users — indeed, even the dumbest of agents (random walkers) can exploit linkage information. In fact, the random walk model may turn out to be more than just a lower bound for browsing behavior. Huberman *et al.* [17] argue that it is a very good predictive model of human browsing behavior. They assume that the *value* (e.g., relevance) 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 equation is stronger than our linkage conjecture, since it implies a positive correlation between V_L and V_{L-1} (equivalent to our relevance autocorrelation) for any $\mu > 0$. Huberman *et al.* find that the inverse-gaussian distribution of surfing depth (clicks per Web site) derived from the above random walk equation accurately fits experimental data on surfing behavior, and therefore they call such a distribution a universal *law of surfing.* Although our conjecture on the value of linkage topology is more modest, it finds strong support in these findings. Furthermore, the random-walk assumption implies normative models 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 [24]. Thus we feel justified in our confidence that browsing is not an unreasonable task for autonomous agents.

Linkage topology is not a sufficient condition for an effective search, however. For a given query q, it seems plausible that relevance autocorrelation decays rapidly for distances greater than some correlation distance $\Delta_{R,q}$. If

an agent is farther than $\Delta_{R,q}$ links away from a relevant document, the search is blind. Since $\Delta_{R,q}$ is unknown, there is no way to estimate the necessary amount of energy with which to initially endow agents at any arbitrary starting point. If such amount is underestimated, extinction will ensue before any relevant document is located. And if it is overestimated, resources will be unnecessarily wasted searching through unlikely neighborhoods.

It appears then crucial to launch InfoSpiders from "good" starting points. By this we mean that a starting point should be within a radius $\Delta_{R,q}$ of a target page. Our previous experiments (see the next subsection) quantitatively confirm this necessity. A naive way to meet this condition would be to start from pages with very high numbers of out-links. This is not a solution, however, because the probability of choosing an appropriate link from such a page decreases in proportion to its fan-out.

Search engines, on the other hand, can rely on statistical topology to provide for good starting points. Even if the index does not contain the target, it probably contains pages within a radius $\Delta_{R,q}$ of it. We are assured that the pages returned by the engine contain the query words. Linkage topology can then lead the agents in the right direction. We conclude that the information encoded by statistical and linkage topologies are complementary — search engines and browsing agents should work together to serve the user most effectively.

1.6.2 Previous results

The InfoSpiders system was evaluated, at both the level of the population and of single agents, on a more limited and controlled subset than the whole Web. A subset of the Encyclopaedia Britannica [12] was chosen, among other methodological reasons, because of its reliable and readily available relevant sets corresponding to a large number of test queries. Each query had a *depth* describing the minimum distance between the starting points and the relevant set. Depth was roughly inverse to generality — deeper queries were more specific and their smaller relevant sets added to their difficulty.

The collective performance of InfoSpiders was assessed, and compared to other search algorithms, by a variation of the *search length* metric [9]. We measured the total number of pages visited by InfoSpiders before some fraction of the relevant set was discovered. As a sanity check, InfoSpiders were first compared against a simple non-adaptive algorithm, *breadth-first-search* [29]. In this experiment the use of the centralized cache was allowed, so that only new pages counted toward the search length. Encouragingly, the search length of InfoSpiders across the whole depth spectrum was as much as an order of magnitude shorter than that achieved by breadth-first-search.

In a second series of experiments, distributed InfoSpiders were compared against a priority-queue implementation of *best-first-search* [30]. This algorithm used the InfoSpiders machinery at the agent level, combined with a

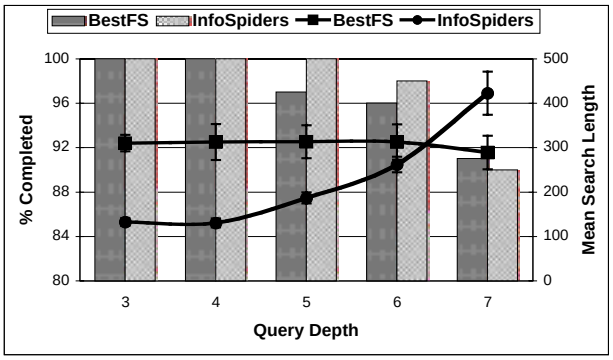

Fig. 1.8. Performance of unsupervised InfoSpiders versus best-first-search. Non-completed queries were those for which InfoSpiders ran out of time, or went extinct; and for which best-first-search ran out of time, or became engulfed in previously visited areas. Search length was averaged over same-depth queries, and error bars correspond to standard errors. The implementation used in this experiment simulated distributed execution of agents across servers. Therefore InfoSpiders had no cache and counted all (new and old) pages visited; but search length was measured concurrently, i.e., taking the maximum parallel length across all agent lineages.

globally-optimum heuristic to decide the order in which to visit the pages.[9] The results, partly reproduced in Fig. 1.8, were illuminating. For the more general queries InfoSpiders had a significant advantage over best-first-search, while for the deepest queries the situation was reversed. Furthermore, both algorithms degraded in performance with increasing depth, i.e., they succeeded less frequently at locating the required fraction of relevant documents. These results support our argument of the previous section, in favor of using search engines to provide InfoSpiders with good starting points.

Focusing our analysis at the level of single agents, we also observed individual agents in the course of single search runs to evaluate whether they were capable of internalizing spatially and temporally local features (words) from the environment into their adaptive search behaviors [30, 28]. In fact, we found that two agents born at the same time but in different places (pages) had adapted to the spatial context in which they had evolved. Their keyword vectors contained different words that were locally correlated with relevance, in their respective neighborhoods. Similarly, along the temporal dimension, two agents born on the same page but at a different times were subject to different user assessments; the different keywords appearing in their representations were consistent with the shifting importance associated with locally occurring terms in their respective temporal contexts. Finally, two agents born at the same time and in the same place learned during their lifetimes different neural net weights for the same keywords, reflecting the reinforcement

[9] InfoSpiders can implement a search strategy similar to best-first-search by evolving high values for the β gene, but only from the local "perspective" of single agents; best-first-search is an upper bound for global search algorithms.

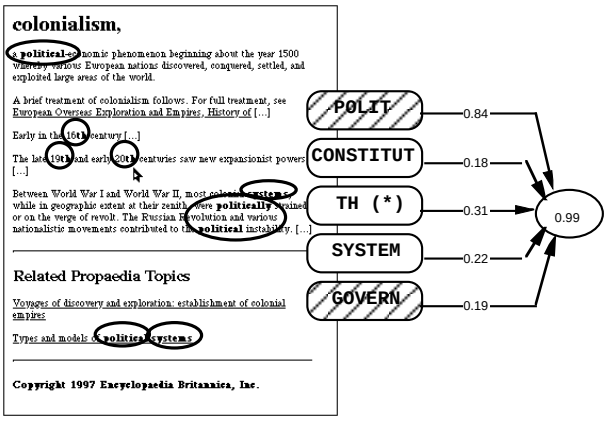

Fig. 1.9. Internalization of spatial, temporal, and experiential context. An agent's representation is shown along with its page of birth. This agent retained some of the original query words (shown as shaded input units) that were locally relevant, but eliminated others in favor of terms that better correlated locally with relevance. One of these (marked by a star) was the product of a new mutation, capturing the occurrence of this term in the page whose relevance resulted in the agent's birth. Relevance was part of the temporal context because it was determined by user feedback. The weights of the neural net (in this case a simple perceptron) will eventually encode the relative importance attributed to the keywords.

signals provided to Q-learning by their respective life experiences. Figure 1.9 illustrates these observations.

1.6.3 The role of context

The results just described point to some other limitations of search engines, besides those of scalability. Adaptive information agents can further improve on search engines with their capability to deal with *context*.

All samples of language, including the Web pages indexed by search engines, depend heavily on shared context for comprehension. An author must make assumptions about the intended audience of a document, and when the latter appears in a "traditional" medium (conference proceedings, academic journal, etc.) it is likely that typical readers will understand it as intended. But the Web brings to the document a huge new audience, a good part of which will not share the author's intended context.

These vague linguistic concerns have concrete manifestation in the global word frequency statistics collected by 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 [40].

Situated agents, on the other hand, can rely on local coherence in keyword distributions by exploiting their linkage topology. Over time, agents may

come to internalize the features that best describe the current documents and discriminate relevant pages. For example, agents browsing through pages about "rock climbing" and "rock'n'roll" should attribute different weights to the word "rock" depending on whether the query they are trying to satisfy is about music or sports. The neighborhood where an agent is situated in the environment provides it with the local context within which to analyze word meanings. Conversely, the words that surround links in a document provide an agent with valuable information to guide its path decisions.

Indices are also constructed without knowledge of the particular queries that they will answer, or of the users posing them. A universal ranking scheme may be generally good but probably will not be the best for each specific query or particular user. Conversely, personal agents may adapt to a user's interests, even if they change over time. They can internalize the user's preferences with respect to, e.g., vocabulary, word disambiguation, and relative importance of terms.

1.6.4 The future

This chapter has discussed the scalability limitation of search engines and suggested a solution based on populations of adaptive information agents. The case study of Sect. 1.5 has illustrated the potential search scalability achievable through the synergy between search engines and online browsing agents.

The viability of adaptive information agents in achieving scalable Web search, however, cannot be demonstrated with anecdotal evidence. Quantitative confirmation of the ideas discussed in this chapter must be sought through extensive testing on the Web. One experiment would have human browsers and InfoSpiders compete in locating documents not indexed by search engines. Another approach would be to create a new page and measure how long it takes, on average, until it is found by a crawler (provided it is not directly submitted to it by the author); this time can be compared to the average time it takes InfoSpiders to find the page, starting from appropriate queries and different hit lists derived from search engines.

Continued development of the InfoSpiders prototype (starting with an urgent upgrade of the HTML parser) is a precondition for such experiments and thus represents an important goal for the near future. Many aspects of the model also remain to be explored in the "real world," from unsupervised query expansion to shifting relevance feedback under long-standing queries; from parameter optimization to the role of recombination; and from the effect of cache size to differential costs under distributed implementations.

Beyond such explorations, we envision that in the growing and increasingly complex Web of information, users will have to rely heavily on adaptive personal agents. People will need to trust their agents and delegate more and more of their tedious tasks to them. This will shift the load of the network from today's bulk of human and "dumb" crawlers to more intelligent

agents, possibly exchanging information autonomously on their owners' behalf. Agents will thus shift the boundary between our brain and the world; hopefully we will be able to make a better use of our precious time and cognitive skills.

1.7 Acknowledgments

The authors wish to thank their respective departments at U. Iowa and U. Dayton for support. Parts of the InfoSpiders code are ©1997 W3C (MIT, INRIA, Keio), ©1993 Free Software Foundation, Inc., 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. Graziano Obertelli provided for assistance with software engineering issues. The InfoSpiders project originated from a collaboration with Richard K. Belew, whose advice is behind many of the ideas discussed in this chapter. Finally, we are grateful to Charles Elkan, Gary Cottrell, Russell Impagliazzo, and all the members of the Cognitive Computer Science Research Group in the CSE Department at U. C. San Diego for helpful discussions and suggestions.

1. R Armstrong, D Freitag, T Joachims, and T Mitchell. Webwatcher: A learning apprentice for the world wide web. In *Working Notes of the AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments*, 1995.
2. GO Arocena, AO Mendelzon, and GA Mihaila. Applications of a web query language. In *Proc. 6th International World Wide Web Conference*, 1997.
3. M Balabanović. An adaptive web page recommendation service. In *Proc. 1st International Conference on Autonomous Agents*, 1997.
4. KD Bollacker, S Lawrence, and CL Giles. CiteSeer: An autonomous web agent for automatic retrieval and identification of interesting publications. In *Proc. 2nd International Conference on Autonomous Agents*, 1998.
5. CM Bowman, PB Danzig, U Manber, and MF Schwartz. Scalable internet resource discovery: Research problems and approaches. *Communications of the ACM*, 37(8):98–107, 1994.
6. S Chakrabarti, B Dom, and P Indyk. Enhanced hypertext categorization using hyperlinks. In *Proc. ACM SIGMOD*, Seattle, WA, 1998.
7. S Chakrabarti, B Dom, P Raghavan, S Rajagopalan, D Gibson, and J Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. In *Proc. 7th International World Wide Web Conference*, 1998.
8. L Chen and K Sycara. WebMate: A personal agent for browsing and searching. In *Proc. 2nd International Conference on Autonomous Agents*, 1998.
9. WS Cooper. 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, 1968.
10. PME De Bra and RDJ Post. Information retrieval in the world wide web: Making client-based searching feasible. In *Proc. 1st International World Wide Web Conference*, Geneva, 1994.
11. Digital Equipment Corporation. http://altavista.digital.com.
12. Encyclopaedia Britannica, Inc. http://www.eb.com.
13. Excite. http://www.excite.com.
14. C Fox. Lexical analysis and stop lists. In *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.
15. WB Frakes. Stemming algorithms. In *Information Retrieval: Data Structures and Algorithms*. Prentice-Hall, 1992.
16. WE Hart and RK Belew. Optimization with genetic algorithm hybrids that use local search. In RK Belew and M Mitchell, editors, *Adaptive Individuals in Evolving Populations: Models and Algorithms*, Santa Fe Institute Studies in the Sciences of Complexity. Addison Wesley, Reading, MA, 1996.
17. BA Huberman, PLT Pirolli, JE Pitkow, and RM Lukose. Strong regularities in world wide web surfing. *Science*, 280(5360):95–97, 1998.

18. T Joachims, D Freitag, and T Mitchell. WebWatcher: A tour guide for the world wide web. In *Proc. International Joint Conference on Artificial Intelligence*, 1997.
19. RR Larson. Bibliometrics of the world wide web: An exploratory analysis of the intellectual structure of cyberspace. In *Proc. 1996 Annual ASIS Meeting*, 1996.
20. SR Lawrence and CL Giles. Searching the world wide web. *Science*, 280:98–100, 1998.
21. DD Lewis. Information retrieval and the statistics of large data sets. In *Proc. NRC Massive Data Sets Workshop*, Washington, DC, 1996.
22. DD Lewis. Challenges in machine learning for text classification. In *Proc. 9th Annual Conference on Computational Learning Theory*, New York, NY, 1997.
23. L-J Lin. Self-improving reactive agents based on reinforcement learning, planning, and teaching. *Machine Learning*, 8:293–321, 1992.
24. RM Lukose and BA Huberman. Surfing as a real option. In *Proc. 4th International Conference on Computational Economics*, 1998.
25. Lycos. http://www.lycos.com.
26. P Maes. Agents that reduce work and information overload. *Comm. of the ACM*, 37(7):31–40, 1994.
27. F Menczer. ARACHNID: Adaptive retrieval agents choosing heuristic neighborhoods for information discovery. In *Proc. 14th International Conference on Machine Learning*, 1997.
28. F Menczer. *Life-like agents: Internalizing local cues for reinforcement learning and evolution*. PhD thesis, University of California, San Diego, 1998.
29. F Menczer and RK Belew. Adaptive information agents in distributed textual environments. In *Proc. 2nd International Conference on Autonomous Agents*, Minneapolis, MN, 1998.
30. F Menczer and RK Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the web. Technical Report CS98–579, University of California, San Diego, 1998.
31. F Menczer and RK Belew. Local selection. In *Evolutionary Programming VII*, number 1447 in Lecture Notes in Computer Science. Springer, 1998.
32. Metacrawler. http://www.metacrawler.com.
33. AE Monge and CP Elkan. The WEBFIND tool for finding scientific papers over the worldwide web. In *Proceedings of the 3rd International Congress on Computer Science Research*, 1996.
34. M Pazzani, J Muramatsu, and D Billsus. Syskill & Webert: Identifying interesting web sites. In *Proc. National Conference on Artificial Intelligence (AAAI96)*, 1996.
35. E Rivlin, R Botafogo, and B Shneiderman. Navigating in hyperspace: designing a structure-based toolbox. *Communications of the ACM*, 37(2):87–96, 1994.
36. DE Rumelhart, GE Hinton, and RJ Williams. Learning internal representations by error propagation. In DE Rumelhart and JL McClelland, editors, *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 1. Bradford Books (MIT Press), Cambridge, MA, 1986.
37. G Salton. The 'generality' effect and the retrieval evaluation for large collections. *Journal of the American Society for Information Science*, 23:11–22, 1972.
38. J Shakes, M Langheinrich, and O Etzioni. Dynamic reference sifting: A case study in the homepage domain. In *Proc. 6th International World Wide Web Conference*, 1997.
39. E Spertus. Parasite: Mining structural information on the web. In *Proc. 6th International World Wide Web Conference*, 1997.

40. AM Steier and RK Belew. Exporting phrases: A statistical analysis of topical language. In R Casey and B Croft, editors, *2nd Symposium on Document Analysis and Information Retrieval*, 1994.
41. CJ van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979. Second edition.
42. CC Vogt and GW Cottrell. Predicting the performance of linearly combined ir systems. In *Proceedings of the ACM SIGIR Conference*, 1998.
43. W3C Reference Library. Libwww version 5.1. http://www.w3.org/Library, 1997.
44. CJCH Watkins. *Learning from delayed rewards*. PhD thesis, King's College, Cambridge, UK, 1989.
45. R Weiss, B Velez, M Sheldon, C Nemprempre, P Szilagyi, and DK Giffor. Hypursuit: A hierarchical network search engine that exploits content-link hypertext clustering. In *Proc. Seventh ACM Conference on Hypertext*, 1996.