Topical Web Crawlers: Evaluating Adaptive Algorithms

FILIPPO MENCZER
Indiana University
GAUTAM PANT
University of Utah
and
PADMINI SRINIVASAN
University of Iowa

Topical crawlers are increasingly seen as a way to address the scalability limitations of universal search engines, by distributing the crawling process across users, queries, or even client computers. The context available to such crawlers can guide the navigation of links with the goal of efficiently locating highly relevant target pages. We developed a framework to fairly evaluate topical crawling algorithms under a number of performance metrics. Such a framework is employed here to evaluate different algorithms that have proven highly competitive among those proposed in the literature and in our own previous research. In particular we focus on the tradeoff between exploration and exploitation of the cues available to a crawler, and on adaptive crawlers that use machine learning techniques to guide their search. We find that the best performance is achieved by a novel combination of explorative and exploitative bias, and introduce an evolutionary crawler that surpasses the performance of the best non-adaptive crawler after sufficiently long crawls. We also analyze the computational complexity of the various crawlers and discuss how performance and complexity scale with available resources. Evolutionary crawlers achieve high efficiency and scalability by distributing the work across concurrent agents, resulting in the best performance/cost ratio.

Categories and Subject Descriptors: H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval—*search process*; H.3.4 [**Information Storage and Retrieval**]: Systems and Software—*distributed systems*; *information networks*; *performance evaluation (efficiency and effectiveness)*; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*graph and tree search strategies*; *heuristic methods*; I.2.11 [**Artificial Intelligence**]: Distributed Artificial Intelligence—*intelligent agents*; *multiagent systems*

General Terms: Performance, Measurement, Algorithms, Design

Additional Key Words and Phrases: Topical crawlers, evaluation, efficiency, exploration, exploitation, evolution, reinforcement learning

This work was funded in part by NSF CAREER grant no. IIS-0133124/0348940 to FM.
Current authors' addresses: F. Menczer, School of Informatics and Department of Computer Science, Indiana University, Bloomington, IN 47408, fil@indiana.edu; G. Pant, School of Accounting and Information Systems, The University of Utah, Salt Lake City, UT 84112, actgp@business.utah.edu; P. Srinivasan, School of Library and Information Science, The University of Iowa, Iowa City, IA 52242, padmini-srinivasan@uiowa.edu.

1. INTRODUCTION

Searching the Web is a difficult task. A lot of machine learning work is being applied to one part of this task, namely ranking indexed pages by their estimated relevance with respect to user queries. This is a crucial task because it heavily influences the perceived effectiveness of a search engine. Users often look at only a few top hits, making the precision achieved by the ranking algorithm of paramount importance. Early search engines ranked pages principally based on their lexical similarity to the query. The key strategy was to devise the best weighting algorithm to represent Web pages and queries in a vector space, such that closeness in such a space would be correlated with semantic relevance.

More recently the structure of hypertext links has been recognized as a powerful new source of evidence for Web semantics. Many machine learning techniques have been employed for link analysis, so that a page's linkage to other pages, together with its content, could be used to estimate its relevance [Kleinberg and Lawrence 2001]. The best known example of such link analysis is the PageRank algorithm successfully employed by the Google search engine [Brin and Page 1998]. Other machine learning techniques to extract meaning from link topology have been based on identifying hub and authority pages via eigenvalue analysis [Kleinberg 1999] and on other graph-theoretical approaches [Gibson et al. 1998; Flake et al. 2002; Kumar et al. 1999]. While purely link-based methods have been found to be effective in some cases [Flake et al. 2002], link analysis is most often combined with lexical pre/post-filtering [Brin and Page 1998; Kleinberg 1999].

However, all this research only addresses half of the problem. No matter how sophisticated the ranking algorithm we build, the results can only be as good as the pages indexed by the search engine — a page cannot be retrieved if it has not been indexed. This brings us to the other aspect of Web searching, namely crawling the Web in search of pages to be indexed. The visible Web with its estimated size between 4 and 10 billion "static" pages as of this writing [Cyveillance 2000] offers a challenging information retrieval problem. This estimate is more than double the 2 billion pages that the largest search engine, Google, reports to be "searching." In fact the coverage of the Web by search engines has not improved much over the past few years [Lawrence and Giles 1998; 1999]. Even with increasing hardware and bandwidth resources at their disposal, search engines cannot keep up with the growth of the Web. The retrieval challenge is further compounded by the fact that Web pages also change frequently [Wills and Mikhailov 1999; Cho and Garcia-Molina 2000; Brewington and Cybenko 2000]. For example Cho and Garcia-Molina [2000] in their daily crawl of 720,000 pages for a period of about 4 months, found that it takes only about 50 days for 50% of the Web to change. They also found that the rate of change varies across domains. For example it takes only 11 days in the `.com` domain versus 4 months in the `.gov` domain for the same extent of change. Thus despite the valiant attempts of search engines to index the whole Web, it is expected that the subspace eluding indexing will continue to grow. Hence the solution offered by search engines, i.e., the capacity to answer any query from any user is recognized as being limited. It therefore comes as no surprise that the development of topical crawler algorithms has received significant attention in recent years [Aggarwal et al. 2001; Chakrabarti et al. 1999; Cho et al.

1998; Hersovici et al. 1998; Menczer and Belew 2000; Menczer et al. 2001; Menczer 2003].

Topical crawlers (also known as focused crawlers) respond to the particular information needs expressed by topical queries or interest profiles. These could be the needs of an individual user (query time or online crawlers) or those of a community with shared interests (topical or vertical search engines and portals). Topical crawlers support decentralizing the crawling process, which is a more scalable approach [O'Meara and Patel 2001; Pant et al. 2003]. An additional benefit is that such crawlers can be driven by a rich context (topics, queries, user profiles) within which to interpret pages and select the links to be visited.

Starting with the early breadth first [Pinkerton 1994] and depth first [De Bra and Post 1994] crawlers defining the beginnings of research on crawlers, we now see a variety of algorithms. There is Shark Search [Hersovici et al. 1998], a more aggressive variant of De Bra's Fish Search [1994]. There are crawlers whose decisions rely heavily on link based criteria [Cho et al. 1998; Diligenti et al. 2000]. Diligenti et al. [2000], for example, use backlinks based context graphs to estimate the likelihood of a page leading to a relevant page, even if the source page itself is not relevant. Others exploit lexical and conceptual knowledge. For example Chakrabarti et al. [1999] use a hierarchical topic classifier to select links for crawling. Still others emphasize contextual knowledge [Aggarwal et al. 2001; Menczer and Belew 2000; Najork and Wiener 2001] for the topic including that received via relevance feedback. For example Aggarwal et al. [2001] learn a statistical model of the features appropriate for a topic while crawling. In previous work by one of the authors, Menczer and Belew [2000] show that in well-organized portions of the Web, effective crawling strategies can be learned and evolved by agents using neural networks and evolutionary algorithms.

The present highly creative phase regarding the design of topical crawlers is accompanied by research on the evaluation of such crawlers, a complex problem in and of itself. For example a challenge specific to Web crawlers is that the magnitude of retrieval results limits the availability of user based relevance judgments. In previous research we have started to explore several alternative approaches both for assessing the quality of Web pages as well as for summarizing crawler performance [Menczer et al. 2001]. In a companion paper [Srinivasan et al. 2004] we expand such a methodology by describing in detail a framework developed for the fair evaluation of topical crawlers. Performance analysis is based on both quality and on the use of space resources. We formalize a class of crawling tasks of increasing difficulty, propose a number of evaluation metrics based on various available sources of relevance evidence, analyze the time complexity and scalability of a number of crawling algorithms proposed in the literature, and study the relationship between the performance of various crawlers and various topic characteristics.

Our goal in this paper is to examine the algorithmic aspects of topical crawlers. We have implemented within our evaluation framework a group of crawling algorithms that are representative of the dominant varieties published in the literature. Based on the evaluation of such crawlers in a particular task, we have designed and implemented two new classes of crawling algorithms that currently stand as the best performing crawlers for the task considered. These proposed crawler classes allow

us to focus on two crucial machine learning issues that have not been previously studied in the domain of Web crawling strategies: (i) the role of exploration versus exploitation, and (ii) the role of adaptation (learning and evolutionary algorithms) versus static approaches. Overall, our intention is to obtain a more complete and reliable picture of the relative advantages and disadvantages of various crawling strategies than what can be drawn from the current literature.

In Section 2 we summarize the aspects of our evaluation framework that are relevant to this study. This includes our system architecture, a formalization of the crawling task, a description of the dataset used, and the evaluation metrics employed for assessing both effectiveness and efficiency of the crawlers. Section 3 outlines a number of crawling algorithms proposed in the literature, on which sufficient information is available to allow for our own implementation, and compares their performance with respect to their effectiveness and efficiency. The scalability of the algorithms is also analyzed by varying the resource constraints of the crawlers. A class of best-first crawling algorithms and a class of shark-search algorithms are introduced in Section 4 and used to study the tradeoff between exploration and exploitation. The issue of adaptation is discussed in Section 5, using a multi-agent class of crawling algorithms in which individuals can learn to estimate links by reinforcement based on local link and lexical cues, while the population can evolve to bias the exploration process toward areas of the Web that appear promising. In Section 6 we analyze the robustness of our results in the face of longer crawls, and finally Section 7 discusses our findings and concludes the paper with ideas about further research.

2. EVALUATION FRAMEWORK

In order to empirically study the algorithmic issues outlined above, we need to first illustrate the evaluation framework that we will use in our experiments. Here we focus on just the aspects of the framework that are relevant to the analyses in this paper. For a more complete treatment the reader is referred to a companion paper [Srinivasan et al. 2004].

2.1 System Architecture

Figure 1 illustrates the architecture of the crawler evaluation framework. The system is designed so that all the logic about any specific crawling algorithm is encapsulated in a crawler module which can be easily plugged into the system through a standard interface. All crawling modules share public data structures and utilities to optimize efficiency without affecting the fairness of any evaluations. Examples of common facilities include a cache, an HTTP interface for the Web, a simple HTML parser, a stemmer [Porter 1980], benchmarking and reporting routines. The system is implemented in Perl with Berkeley embedded databases for storing persistent data as well as data structures shared across concurrent processes.

Each tested crawler can visit up to `MAX_PAGES` pages per topic, starting from a *seed set*. We use a timeout of 10 seconds for Web downloads. Large pages are chopped so that we retrieve only the first 10 KB. The only protocol allowed is HTTP GET (with redirection), and we also filter out all but pages with `text/html` content. Stale links yielding HTTP error codes are removed as they are found; only good links are used in the analysis.

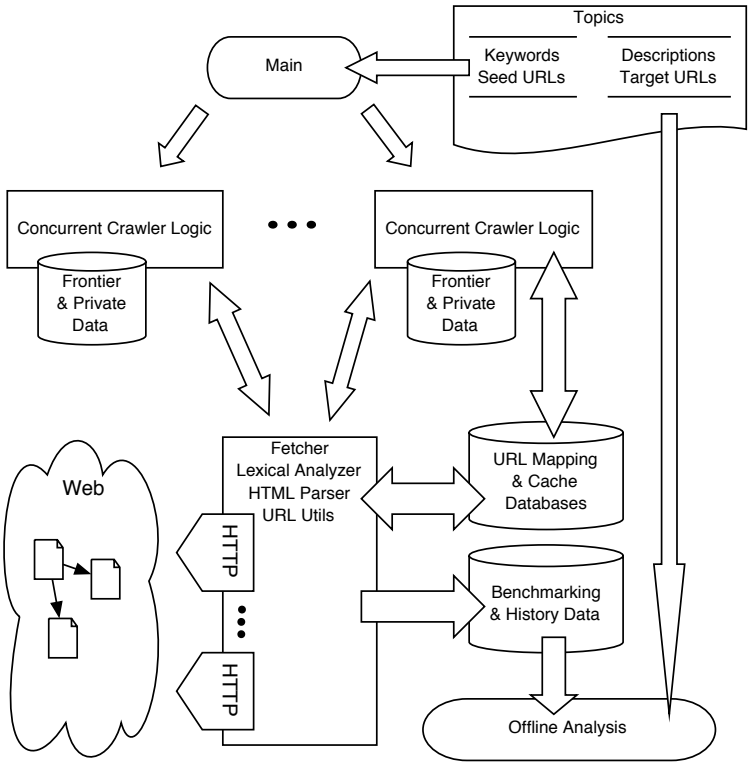

Fig. 1. Architecture of crawler evaluation framework.

2.2 Resource Constraints

Crawlers consume resources: network bandwidth to download pages, memory to maintain private data structures in support of their algorithms, CPU to evaluate and select URLs, and disk storage to store the (stemmed) text and links of fetched pages as well as other persistent data. Obviously the more complex the selection algorithm, the greater the use of such resources. In order to allow for a fair comparison of diverse crawling algorithms, we take two measures:

(1) We track the CPU time taken by each crawler for each page and each topic, ignoring the time taken by the fetch module, which is common to all the crawlers. We do this since it is impossible to control for network traffic and congestion, and we want to benchmark only the crawler-specific operations. The monitored CPU time will be used to compare the complexity of the crawling algorithms.

(2) We limit the memory available to each crawler by constraining the size of its buffer. This buffer is used by a crawler to temporarily store link data, typically a frontier of pages whose links have not been explored. Each crawler is allowed to track a maximum of `MAX_BUFFER` links. If the buffer becomes full, the crawler must decide which links are to be substituted as new ones are added. The value of the `MAX_BUFFER` parameter will be varied between 2^8 and 2^{11} to gauge how the

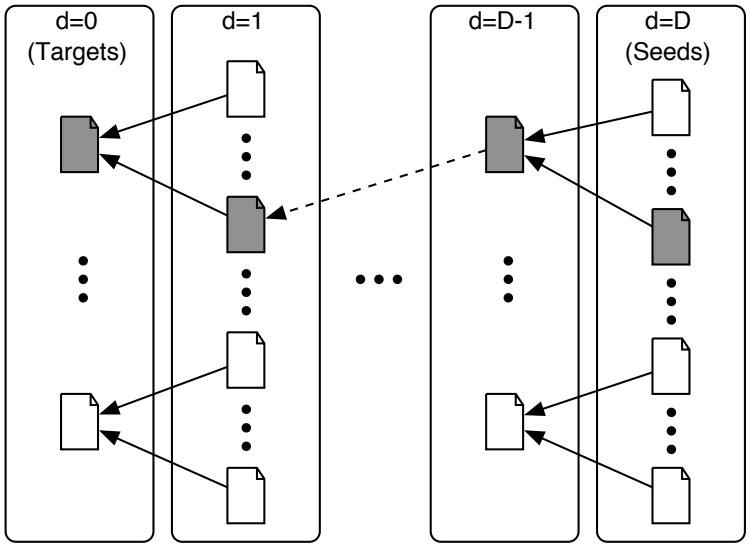

Fig. 2. Generalized crawl task. A seed page and its path to a target are shown in gray.

performance of the crawling algorithms scales with their memory resources. Note that we set the size of the buffer to such small values because we are particularly interested in analyzing crawler performance in the most demanding circumstances, i.e., when memory is very scarce. This would be realistic, for example, if a crawler was deployed as an applet [Pant and Menczer 2002].

2.3 Crawl Task

What should be the starting URLs, i.e., the *seed set* for a crawler? In the discussion that follows, we use the word *targets* to represent a known subset of relevant pages. If we identify a target set for any given topic, we would like our crawlers to reach these target pages, and/or pages similar to the targets, from whatever seed pages are used to begin the crawl.

Figure 2 illustrates a generalized crawl task. Seed pages are chosen so as to guarantee that from each seed there exist at least one path to *some* target page, within a distance of at most D links. In practice we identify the seed pages by first considering the inlinks of each target page. Submitting a `link:` query to a search engine[1] yields up to 20 URLs of pages linking to a page. We then repeat this for each of these first-neighbors, and so on for D steps. Hence, if we had 10 targets to start with, we would end up with a maximum of 10×20^D unique URLs. A subset of 10 seed URLs is then picked at random from this set. Note that this procedure does not guarantee a path from the seed set to *every* target.

Such a generalized crawl task formalizes a class of problems whose difficulty

[1]At the time of the experiments described here we used `http://google.yahoo.com`, which allowed us to comply with the Robot Exclusion Standard. Currently we use the Google Web API (`http://www.google.com/apis/`).

intuitively increases with D. For $D = 0$, the seeds coincide with the targets so that the crawlers start from pages that are assumed to be relevant. This task mimics the "query by example" search mode where the user provides the crawler with a few sample relevant pages. As an alternative strategy these seeds may also be obtained from a Web search engine. The idea is to see if the crawlers are able to find other relevant pages for the topic. This is the approach used in most crawler evaluation research to date (see e.g. Ben-Shaul et al. [1999], Chakrabarti et al. [1999], Diligenti et al. [2000]).

An assumption implicit in the $D = 0$ crawl task is that pages that are relevant tend to be neighbors of each other. Thus the objective of the crawler is to stay focused, i.e., to remain within the neighborhood in which relevant documents have been identified. However, the seeds obviously cannot be used to evaluate the performance of a crawler.[2] Typical applications of the $D = 0$ crawl task are query-time search agents that use results of a search engine as starting points to provide a user with recent and personalized results [Pant and Menczer 2002]. Such a task may also be a part of Web mining or competitive intelligence applications, e.g., a search starting from competitors' home pages [Pant and Menczer 2003].

For $D > 0$, crawl seeds are different from the target pages. So in this case there is little prior information available to the crawlers about the relevant pages when the crawl begins. The idea is to see if the crawlers are able to find the targets, and/or other relevant pages. The $D > 0$ crawl task involves seeking relevant pages while starting the crawl from links that are a few links away from some known relevant pages.

With a few exceptions, $D > 0$ crawl tasks are rarely considered in the literature although they allow to model problems of varying difficulty. A $D > 0$ problem is also realistic since quite commonly users are unable to specify known relevant URLs. The effort by Aggarwal et al. [2001] is somewhat related to this task in that the authors start the crawl from general points such as `Amazon.com`. Although Cho et al. [1998] start their crawls at a general point, i.e., the Stanford Web site, topics have rather primitive roles in their research as no target set is identified.

We believe that it is important to understand crawlers within task contexts corresponding to more than a single D value, i.e., with varying degrees of difficulty since they serve different yet valid search modes. For the purposes of this paper we limit our analyses to one difficult task, $D = 3$. Note that given the large average fanout of Web pages, this problem is a bit like looking for a needle in a haystack. To illustrate this point, observe that a breadth-first crawler would have a chance in ℓ^{-D} to visit a target from a seed assuming an average fanout of ℓ links. Starting from 10 seeds and visiting 1,000 pages, $\ell \approx 10$ and $D = 3$ would imply an expected upper bound of around 10% on the recall of target pages.

2.4 Topics and Targets Dataset

In order to evaluate crawler algorithms, we need *topics* and some corresponding relevant *targets*. One could theoretically generate topics and target sets using frequent queries from search engines and user assessments. However this approach

[2] A variant of this task would be to use only part of the target set as seed set and evaluate the crawler's capability to locate the rest [Pant and Menczer 2003].

would make it very expensive to obtain a large number of topics and target sets, and very cumbersome to keep such a dataset up to date over time given the dynamic nature of the Web. Fortunately this data is already available from Web directories updated by human editors. While in our past research we have used Yahoo![3] due to its popularity [Menczer et al. 2001], in our current work we use the Open Directory Project[4] (ODP) because (i) it has less of a commercial bias, (ii) it is an "open resource," and (iii) it is maintained by a very large and diverse number of volunteer editors.

We collected topics by running randomized breadth-first crawls starting from each of the main categories on the Open Directory site. These crawls identify ODP "leaves," i.e., pages that have no children category nodes. Leaves with five or more external links are then used to derive topics. A topic is represented by three types of information derived from the corresponding leaf page. First, the words in the ODP hierarchy form the topic's *keywords*. Second, the external links form the topic's *targets*. Third, we concatenate the text descriptions and anchor text of the target URLs (written by ODP human editors) to form a topic's *description*. The difference between a topic's keywords and its description is that we give the former to the crawlers, as models of (short) query-like topics; and we use the latter, which is a much more detailed representation of the topic, to gauge the relevance of the crawled pages in our post-hoc analysis. Table I shows a few sample topics. The experiments described in this paper use 50 such topics.

2.5 Performance Metrics

In previous research we explored several alternative methodologies for evaluating crawlers [Menczer 2003; Menczer et al. 2001]. These include methods for assessing page relevance using linear classifiers, and using similarity computations. We also explored different methods for summarizing crawler performance such as plotting the mean similarity of retrieved pages to topic descriptions over time and computing the average fraction of retrieved pages that were assessed as being "relevant" by the classifiers (at the end of the crawl). Based upon our previous experience, in a companion paper we discuss in detail a number of measures that we have selected as the minimal set needed to provide a well rounded assessment of crawler performance [Srinivasan et al. 2004]. In particular we propose to assess both recall and precision, using target and lexical criteria for page relevance. This leads to a set of $4 \cdot D \cdot G$ performance metrics, where D is the task difficulty parameter and G is a topic generality parameter.[5] Each performance metric can be plotted as a function of crawling time to yield a dynamic perspective. In addition we assess the efficiency of the crawling algorithms — an aspect that was not previously considered in the literature.

In this paper we employ only two of the performance metrics proposed in Srinivasan et al. [2004], focusing on the effect of different machine learning techniques on the performance of crawling algorithms over time. This view is especially appropriate when studying issues such as greediness and adaptation, whose effects are

[3] `http://dir.yahoo.com`

[4] `http://dmoz.org`

[5] Considering only ODP leaf topics, as in this paper, is a special case corresponding to $G = 0$.

Table I. A few sample ODP topics. The descriptions are truncated for space limitations.

Keywords	Description	Targets
Computers Software Workflow Consulting	Document Management and Workflow Consulting - EID has been providing document management and workflow consulting for over years and is heavily involved in establishing industry standards Our services range from feasibility studies to implementation management; Fast Guard Software Services - Specialized software sales and consulting for NT and Exchange networks email directory synchronization workflow document management NT print management Internet management Wang and Banyan migration Eastman Software WMX SP; FormSoft Group - Provides enterprise workflow consulting web application development and training to corporate and government entities; [...]	http://www.eid-inc.com/ http://www.fastguard.com/ http://www.formsoftgroup.com/ http://www.gain.ch/ http://www.iicon.co.uk/ http://www.mondas.com/index.htm http://www.pdfsolutions.co.uk/
Society Activism Consumer	Bad Business Bureau - Rip off Report is the nationwide consumer reporting web site to enter complaints about companies and individuals who are ripping people off; Consumer Soundoff - Consumer advocate who offers a free service to help reconcile consumer disputes and problems with businesses; Foundation for Taxpayer and Consumer Rights - A nationally recognized California based non profit consumer education and advocacy organization Deals in insurance reform healthcare reform billing errors and other issues; [...]	http://www.badbusinessbureau.com/ http://soundoff1.tripod.com/ http://www.consumerwatchdog.org/ http://members.tripod.com/~birchmore1/muzak/ http://www.ourrylandhomes.com/ http://www.ripoffrevenge.com/ http://www.konsumentsamverkan.se/english/... http://www.ucan.org/ http://www.nocards.org/
Sports Lumberjack	Always Wanted to Be a Lumbarjack - To grow up and survive in Hayward WI a few things to learn; American Birling Association - Created to promote log rolling boom running and river rafting it is the best source of information for lumberjack water sports; American Lumberjack Association - A group of sportsmen and women dedicated to furthering upgrading and standardizing timbersports; Lumber Jack Jill Festival - Festival held in Avoca New York; Lumberjack Days - Held in July on the beautiful St Croix River; Lumberjack Entertainment And Timbersports Guide - Information on lumberjack contests shows world records and competition equipment; [...]	http://www.subway.com/subpages/namap/usa/lumber/ http://www.logrolling.org/ http://www.americanlumberjacks.com/ http://www.westny.com/logfest.htm http://www.lumberjackdays.com/ http://www.starinfo.com/ljguide/lumberjack.html http://www.lumberjackworldchampionships.com/ http://www.canadian-sportfishing.com/... http://www.starinfo.com/woodsmen/ http://www.orofino.com/Lumberjack.htm http://lumberjacksports.com/ http://www.stihlusa.com/timbersports.html http://www.usaxemen.com/ http://www.hotsaw.com/

likely to vary in the course of the crawl. The dynamic measures provide a temporal characterization of the crawl strategy, by considering the pages fetched while the crawl is in progress.

The first metric is the simple *recall level* based on the target pages:

$$\rho(i) = \frac{|S(i) \cap T|}{|T|} \tag{1}$$

where $S(i)$ is the set of pages crawled by crawler i and T is the target set. This measure allows us to determine how well a crawler can locate a few highly relevant pages. However, there may well be many other relevant pages that are not specified in the target set. To assess a crawler's capability to locate these other relevant pages, we have to estimate the relevance of any crawled page. We do so using the lexical similarity between a page and the topic description. Recall that the crawlers do not have any knowledge of topic descriptions, and that these descriptions are manually compiled by human experts as they describe relevant sites. Therefore pages similar to these descriptions have a good likelihood to be closely related to the topic. Rather than inducing a binary relevance assessment by guessing an arbitrary similarity threshold, in the second metric we measure the *mean similarity* between the topic description and the set of pages crawled:

$$\sigma(i) = \frac{\sum_{p \in S(i)} \cos(p, V)}{|S(i)|} \tag{2}$$

where V is the vector representing the topic description and

$$\cos(p, V) = \frac{\sum_{k \in p \cap V} w_{pk} w_{Vk}}{\sqrt{\left(\sum_{k \in p} w_{pk}^2\right)\left(\sum_{k \in V} w_{Vk}^2\right)}} \tag{3}$$

is the standard cosine similarity function. Here w_{dk} is the TF-IDF weight of term k in d where inverse document frequency is computed from the crawl set $\bigcup_i S(i)$ once `MAX_PAGES` pages have been visited by each crawler. Both recall ρ and mean similarity σ can be plotted against the number of pages crawled to obtain a trajectory over time that displays the dynamic behavior of the crawl.

3. CRAWLING ALGORITHMS

Crawlers exploit the Web's hyperlinked structure to retrieve new pages by traversing links from previously retrieved ones. As pages are fetched, their outward links may be added to a list of unvisited pages, which is referred to as the crawl *frontier.* A key challenge during the progress of a topical crawl is to identify the next most appropriate link to follow from the frontier.

The algorithm to select the next link for traversal is necessarily tied to the goals of the crawler. A crawler that aims to index the Web as comprehensively as possible will make different kinds of decisions than one aiming to collect pages from university Web sites or one looking for pages about movie reviews. For the first crawl order may not be important, the second may consider the syntax of the URL to limit retrieved pages to the `.edu` domain, while the third may use similarity with some source page to guide link selection. Even crawlers serving the same goal may adopt different crawl strategies.

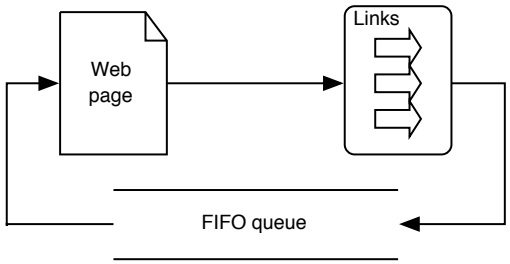

Fig. 3. A Breadth-First crawler.

```
Breadth-First (starting_urls) {
   foreach link (starting_urls) {
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      link := dequeue_link(frontier);
      doc := fetch(link);
      enqueue(frontier, extract_links(doc));
      if (#frontier > MAX_BUFFER) {
         dequeue_last_links(frontier);
      }
   }
}
```

Fig. 4. Pseudocode of the Breadth-First algorithm.

3.1 Crawler Descriptions

In this section we describe and evaluate five different crawling algorithms that we have implemented within our evaluation framework: Breadth-First, Best-First, PageRank, Shark-Search, and InfoSpiders. Our selection is intended to be a broad representation of the crawler algorithms reported in the literature, with an obvious bias toward those that we have been able to reimplement. One of these algorithms, InfoSpiders, derives from our own prior and current research [Menczer 1997; Menczer and Belew 1998; 2000; Menczer and Monge 1999]. In Section 4 we will generalize two of these algorithms (Best-First and Shark-Search) through a parameterization of their greediness level.

3.1.1 *Breadth-First.* A Breadth-First crawler is the simplest strategy for crawling. This algorithm was explored as early as 1994 in the WebCrawler [Pinkerton 1994] as well as in more recent research [Cho et al. 1998; Najork and Wiener 2001]. It uses the frontier as a FIFO queue, crawling links in the order in which they are encountered. Note that when the frontier is full, the crawler can add only one link from a crawled page. The Breadth-First crawler is illustrated in Figure 3 and the algorithm is shown in Figure 4. Breadth-First is used here as a baseline crawler; since it does not use any knowledge about the topic, we expect its performance to provide a lower bound for any of the more sophisticated algorithms.

3.1.2 *Best-First.* Best-First crawlers have been studied by Cho et al. [1998] and Hersovici et al. [1998]. The basic idea is that given a frontier of links, the best link according to some estimation criterion is selected for crawling. Different Best-First

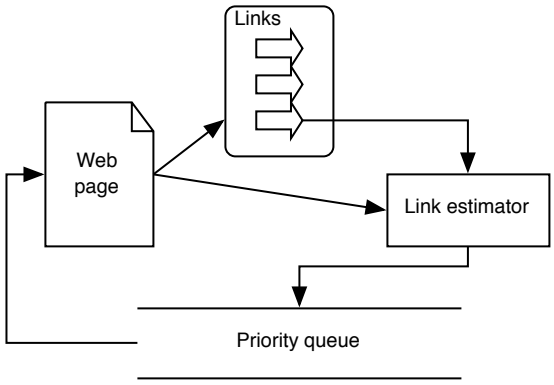

Fig. 5. A Best-First crawler.

```
BFS (topic, starting_urls) {
   foreach link (starting_urls) {
      enqueue(frontier, link, 1);
   }
   while (visited < MAX_PAGES) {
      link := dequeue_top_link(frontier);
      doc := fetch(link);
      score :=  sim(topic, doc);
      enqueue(frontier, extract_links(doc), score);
      if (#frontier > MAX_BUFFER) {
         dequeue_bottom_links(frontier);
      }
   }
}
```

Fig. 6. Best-First-Search algorithm.

strategies of increasing complexity and (potentially) effectiveness could be designed on the bases of increasingly sophisticated link estimation criteria. In our "naive" implementation, the link selection process is guided by simply computing the lexical similarity between the topic's keywords and the source page for the link. Thus the similarity between a page p and the topic keywords is used to estimate the relevance of the pages pointed by p. The URL with the best estimate is then selected for crawling. Cosine similarity is used by the crawler and the links with minimum similarity score are removed from the frontier if necessary in order to not exceed the limit size `MAX_BUFFER`. Figure 5 illustrates a Best-First crawler, while Figure 6 offers a simplified pseudocode of the Best-First-Search (BFS) algorithm. The `sim()` function returns the cosine similarity between topic and page:

$$\texttt{sim}(q,p) = \frac{\sum_{k \in q \cap p} f_{kq} f_{kp}}{\sqrt{\left(\sum_{k \in p} f_{kp}^2\right)\left(\sum_{k \in q} f_{kq}^2\right)}} \tag{4}$$

where q is the topic, p is the fetched page, and f_{kd} is the frequency of term k in d.

Preliminary experiments in a $D = 0$ task have shown that BFS is a competitive crawler [Menczer et al. 2001], therefore here we want to assess how it compares

```
PageRank (topic, starting_urls, frequency) {
   foreach link (starting_urls) {
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      if (multiplies(visited, frequency)) {
         recompute_scores_PR;
      }
      link := dequeue_top_link(frontier);
      doc := fetch(link);
      score_sim :=  sim(topic, doc);
      enqueue(buffered_pages, doc, score_sim);
      if (#buffered_pages >= MAX_BUFFER) {
         dequeue_bottom_links(buffered_pages);
      }
      merge(frontier, extract_links(doc), score_PR);
      if (#frontier > MAX_BUFFER) {
         dequeue_bottom_links(frontier);
      }
   }
}
```

Fig. 7. Pseudocode of the PageRank crawler.

with other algorithms in a more challenging crawling task.

3.1.3 *PageRank.* PageRank was proposed by Brin and Page [1998] as a possible model of user surfing behavior. The PageRank of a page represents the probability that a random surfer (one who follows links randomly from page to page) will be on that page at any given time. A page's score depends recursively upon the scores of the pages that point to it. Source pages distribute their PageRank across all of their outlinks. Formally:

$$PR(p) = (1 - \gamma) + \gamma \sum_{\{d \in in(p)\}} \frac{PR(d)}{|out(d)|} \tag{5}$$

where p is the page being scored, $in(p)$ is the set of pages pointing to p, $out(d)$ is the set of links out of d, and the constant $\gamma < 1$ is a damping factor that represents the probability that the random surfer requests another random page. As originally proposed PageRank was intended to be used in combination with content based criteria to rank retrieved sets of documents [Brin and Page 1998]. This is in fact how PageRank is used in the Google search engine. More recently PageRank has been used to guide crawlers [Cho et al. 1998] and to assess page quality [Henzinger et al. 1999].

In previous work we evaluated a crawler based on PageRank [Menczer et al. 2001]. We used an efficient algorithm to calculate PageRank [Haveliwala 1999]. Even so, as one can see from Equation 5, PageRank requires a recursive calculation until convergence, and thus its computation can be a very resource intensive process. In the ideal situation we would recalculate PageRanks every time a URL needs to be selected from the frontier. Instead, to improve efficiency, we recomputed PageRanks only at regular intervals.

```
Shark (topic, starting_urls) {
   foreach link (starting_urls) {
      set_depth(link, d);
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      link := dequeue_top_link(frontier);
      doc := fetch(link);
      doc_score :=  sim(topic, doc);
      if (depth(link) > 0) {
         foreach outlink (extract_links(doc)) {
            score = (1-r) * neighborhood_score(outlink)
                   +  r   * inherited_score(outlink);
            if (doc_score > 0) {
               set_depth(outlink, d);
            } else {
               set_depth(outlink, depth(link) - 1);
            }
            enqueue(frontier, outlink, score);
         }
         if (#frontier > MAX_BUFFER) {
            dequeue_bottom_link(frontier);
         }
      }
   }
}
```

Fig. 8. Pseudocode of the Shark-Search crawler.

The PageRank crawler can be seen as a variant of Best-First, with a different link evaluation function (see Figure 5). The algorithm is illustrated in Figure 7. The `sim()` function returns the cosine similarity between a topic's keywords and a page as measured according to Equation 4, and the PageRank is computed according to Equation 5. Note that the PageRank algorithm has to maintain a data structure of visited pages in addition to the frontier, in order to compute PageRank. This buffer can contain at most `MAX_BUFFER` pages. We assume pages with 0 out-degree (such as the URLs in the frontier) to be implicitly linked to every page in the buffer, as required for the PageRank algorithm to converge. We use $\gamma = 0.8$ and the threshold for convergence is set at 0.01.

3.1.4 *Shark-Search.* Shark-Search [Hersovici et al. 1998] is a more aggressive version of Fish-Search [De Bra and Post 1994]. In Fish-Search, the crawlers search more extensively in areas of the Web in which relevant pages have been found. At the same time, the algorithm discontinues searches in regions that do not yield relevant pages. Shark-Search offers two main improvements over Fish-Search. It uses a continuous valued function for measuring relevance as opposed to the binary relevance function in Fish-Search. In addition, Shark-Search has a more refined notion of potential scores for the links in the crawl frontier. The potential score of links is influenced by anchor text, text surrounding the links, and inherited score from ancestors (pages pointing to the page containing the links).

The Shark-Search crawler can be seen as a variant of Best-First, with a more sophisticated link evaluation function (see Figure 5). Figure 8 shows the pseudocode

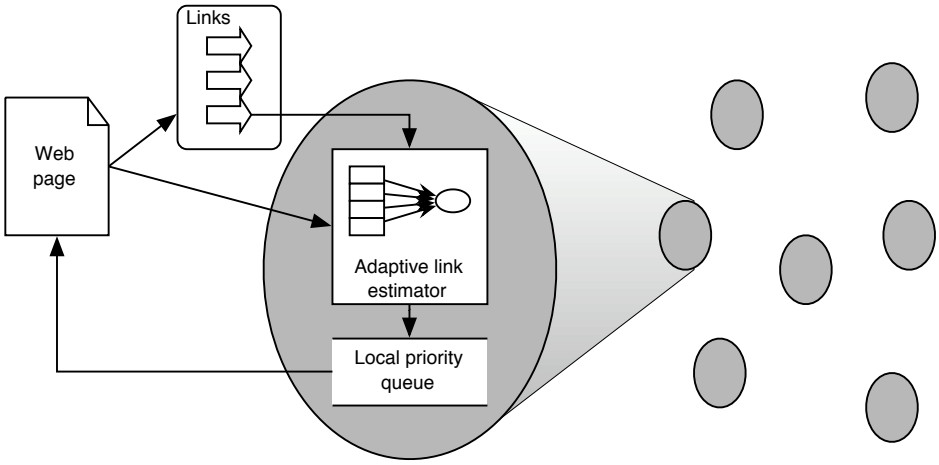

Fig. 9. A population of InfoSpiders.

of the Shark-Search algorithm. The parameters `d` and `r` represent respectively the maximum depth and the relative importance of inherited versus neighborhood scores. In our implementation we set `d` = 3 and `r` = 0.1. This closely follows the details given by Hersovici et al. [1998].

3.1.5 *InfoSpiders.* In InfoSpiders [Menczer 1997; Menczer and Belew 1998; 2000; Menczer and Monge 1999], an adaptive population of agents search for pages relevant to the topic using evolving query vectors and neural nets to decide which links to follow. The idea is illustrated in Figure 9. This evolutionary approach uses a fitness measure based on similarity as a *local* selection criterion. The original algorithm (see, e.g., Menczer and Belew [2000]) was previously simplified and implemented as a crawler module [Menczer et al. 2001]. This crawler was outperformed by BFS due to its extreme local behavior and the absence of any memory — once a link was followed the agent could only visit pages linked from the new page. Since then we have made a number of improvements to the original algorithm while retaining its capability to learn link estimates via neural nets and focus its search toward more promising areas by selective reproduction. The resulting novel algorithm is schematically described in the pseudocode of Figure 10.

InfoSpiders agents are independent from each other and crawl in parallel. The adaptive representation of each agent consists of a list of keywords (initialized with the topic keywords) and a neural net used to evaluate new links. Each input unit of the neural net receives a count of the frequency with which the keyword occurs in the vicinity of each link to be traversed, weighted to give more importance to keywords occurring near the link (and maximum in the anchor text). There is a single output unit. One of the simplifications that we have retained in this implementation is that the neural networks do not have hidden layers, therefore consisting of simple perceptrons. What this implies is that although a neural net's output unit computes a nonlinear sigmoidal function of the input frequencies, the neural network can only learn linear separations of the input space. However we feel

```
IS (topic, starting_urls) {
   agent.topic := topic;
   agent.energy := 0;
   agent.fsize := MAX_BUFFER;
   agent.frontier := starting_urls;
   insert(agent, population);
   while (visited < MAX_PAGES) {
      foreach parallel agent (population) {
         link := pick_and_dequeue(agent.frontier);
         doc := fetch(link);
         newenergy = sim(agent.topic, doc);
         agent.energy += newenergy;
         learn_to_predict(agent.nnet, newenergy);
         foreach outlink (extract_links(doc)) {
            score = ALPHA * newenergy +
                    (1 - ALPHA) * agent.nnet(doc, outlink);
            enqueue(agent.frontier, outlink, score);
         }
         if (#agent.frontier > agent.fsize) {
             dequeue_bottom_links(agent.frontier);
         }
         delta := newenergy - sim(topic, agent.doc);
         if (boltzmann(delta)) {
            agent.doc := doc;
         }
         if (agent.energy > THETA and pop_size < MAX_POP_SIZE) {
            child.topic := expand(agent.topic, agent.doc);
            (child.energy, agent.energy) := (0, 0);
            (child.fsize, agent.fsize) := half(agent.fsize);
            (child.frontier, agent.frontier) := split(agent.frontier);
            child.nnet := extend(agent.nnet, child.topic)
            insert(child, population);
         }
      }
   }
}
```

Fig. 10. Pseudocode of the InfoSpiders algorithm. In the actual implementation, each agent executes as a concurrent process.

that this is a powerful enough representation to internalize the levels of correlation between relevance and the distributions of keywords around links. The output of the neural net is used as a numerical quality estimate for each link considered as input. These estimates are then combined with estimates based on the cosine similarity (Equation 4) between the agent's keyword vector and the page containing the links. A parameter $\alpha, 0 \leq \alpha \leq 1$ regulates the relative importance given to the estimates based on the neural net versus the parent page. Based on the combined score, the agent uses a stochastic selector to pick one of the links with probability

$$\Pr(\lambda) = \frac{e^{\beta\mu(\lambda)}}{\sum_{\lambda' \in \phi} e^{\beta\mu(\lambda')}} \tag{6}$$

where λ is a link from the agent's local frontier ϕ and $\mu(\lambda)$ is its combined score. The β parameter regulates the greediness of the link selector. In the experiments

described in this paper we use $\alpha = 0.5$ and $\beta = 5$.

After a new page has been fetched, the agent receives energy in proportion to the similarity between its keyword vector and the new page (Equation 4). The agent's neural net can be trained to improve the link estimates by predicting the similarity of the new page, given the inputs from the page that contained the link leading to it. We use one epoch of standard error back-propagation [Rumelhart et al. 1986] with a learning rate 0.5. Such a simple reinforcement learning technique provides InfoSpiders with the unique capability to adapt the link-following behavior in the course of a crawl by associating relevance estimates with particular patterns of keyword frequencies around links. While InfoSpiders was the first crawling algorithm to use reinforcement learning [Menczer 1997], others have considerably expanded on this idea [Rennie and McCallum 1999; McCallum et al. 1999; O'Meara and Patel 2001].

An agent moves to the newly selected page only if the `boltzmann()` function returns a true condition (cf. Figure 10). This is determined stochastically based on the probability

$$\Pr(\delta) = \frac{1}{1 + e^{-\delta/T}} \tag{7}$$

where δ is the difference between the similarity of the new and current page to the agent's keyword vector and $T = 0.1$ is a temperature parameter.

An agent's energy level is used to determine whether or not an agent should reproduce after visiting a page. An agent reproduces when the energy level passes the constant threshold `THETA`=2. At reproduction, the offspring receives half of the parent's link frontier. The offspring's keyword vector is also mutated (expanded) by adding the term that is most frequent in the parent's current document. Such a mutation provides InfoSpiders with the unique capability to adapt the search strategy based on new clues captured from promising pages, while reproduction itself allows the population to bias the search toward areas (agents) that lead to good pages.

There can be at most `MAX_POP_SIZE`=8 agents, maintaining a distributed frontier whose total size does not exceed `MAX_BUFFER`. The behavior of individual agents is still based on localized link estimates and keyword expansion, however unlike in the original algorithm, each agent retains a more global representation of the search space in its frontier.

Others have explored the use of evolutionary algorithms for Web search problems. For example a genetic algorithm has been applied to evolve populations of keywords and logic operators, used to collect and recommend pages to the user [Moukas and Zacharia 1997; Nick and Themis 2001]. In such systems the learning process is sustained by meta-search engines and guided by relevance feedback from the user, while in this paper we are interested in agents that can actually crawl the Web and learn from unsupervised interactions with the environment.

3.2 Experimental Results

We now outline the results of a number of crawls carried out to evaluate and compare performance, efficiency and scalability for the crawling algorithms described in Section 3.1.

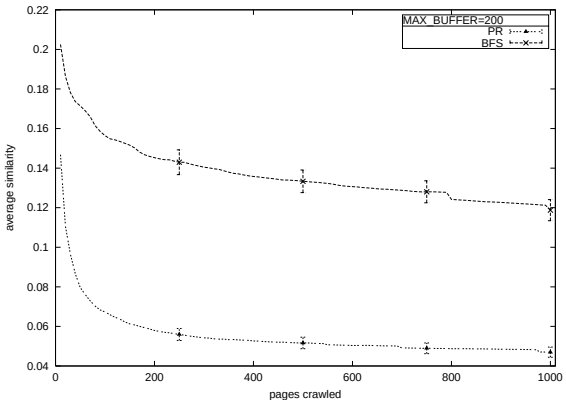

Fig. 11. Average similarity to topic descriptions by PageRank and Best-First crawlers on a $D = 0$ task. The data (from Menczer et al. [2001]) is based on 53 topics.

Unless otherwise stated, the results in this and the following sections are based the $D = 3$ crawling task and the `MAX_PAGES` parameter is set to 1,000. These settings correspond to a particularly difficult task in which the relevant pages are far away and the crawler can only visit a very small number of pages — possibly while the user awaits [Pant and Menczer 2002]. As discussed in Section 2.3 this challenging task is akin to searching for needles in a haystack, making it an interesting test problem on which to evaluate the various crawlers.

Also unless otherwise stated, the results in this and the following sections are based on averages across 50 topics. Throughout the paper, error bars in plots correspond to ± 1 standard error, i.e. standard deviation of the mean across topics. We interpret non-overlapping error bars as a statistically significant difference at the 68% confidence level assuming normal noise.

3.2.1 *Dynamic Performance.* In our prior experiments the PageRank crawler was found not to be competitive (worse than Breadth-First) due to its minimal exploitation of the topic context and its resource limitations [Menczer et al. 2001]. The PageRank metric is designed as a global measure and its values are of little use when computed over a small set of pages. To illustrate, Figure 11 plots average similarity to topic descriptions in a $D = 0$ crawl, i.e., starting from target pages. Given the poor performance of PageRank under the performance measures and tasks we use here, we will not further report on this crawling algorithm in the remainder of this paper.

The performance of the other four crawlers — Breadth-First, Best-First, Shark-Search and InfoSpiders — is compared in Figure 12. We ran four variations of each crawling algorithm for values of `MAX_BUFFER` between $2^8 = 256$ and $2^{11} = 2,048$.[6]

Recall and similarity yield qualitatively consistent results. As expected Breadth-First performs poorly and thus constitutes a baseline. Surprisingly, Shark-Search does not perform significantly better than Breadth-First except for one case, the similarity metric for `MAX_BUFFER`=256. Our intuition for the poor performance of Shark-Search is that this algorithm may not be sufficiently explorative (see Section 4). Best-First and InfoSpiders significantly outperform the other two crawlers

[6]Error bars are a bit larger for `MAX_BUFFER`=2,048 because the averages are based on only 28 topics due to time constraints.

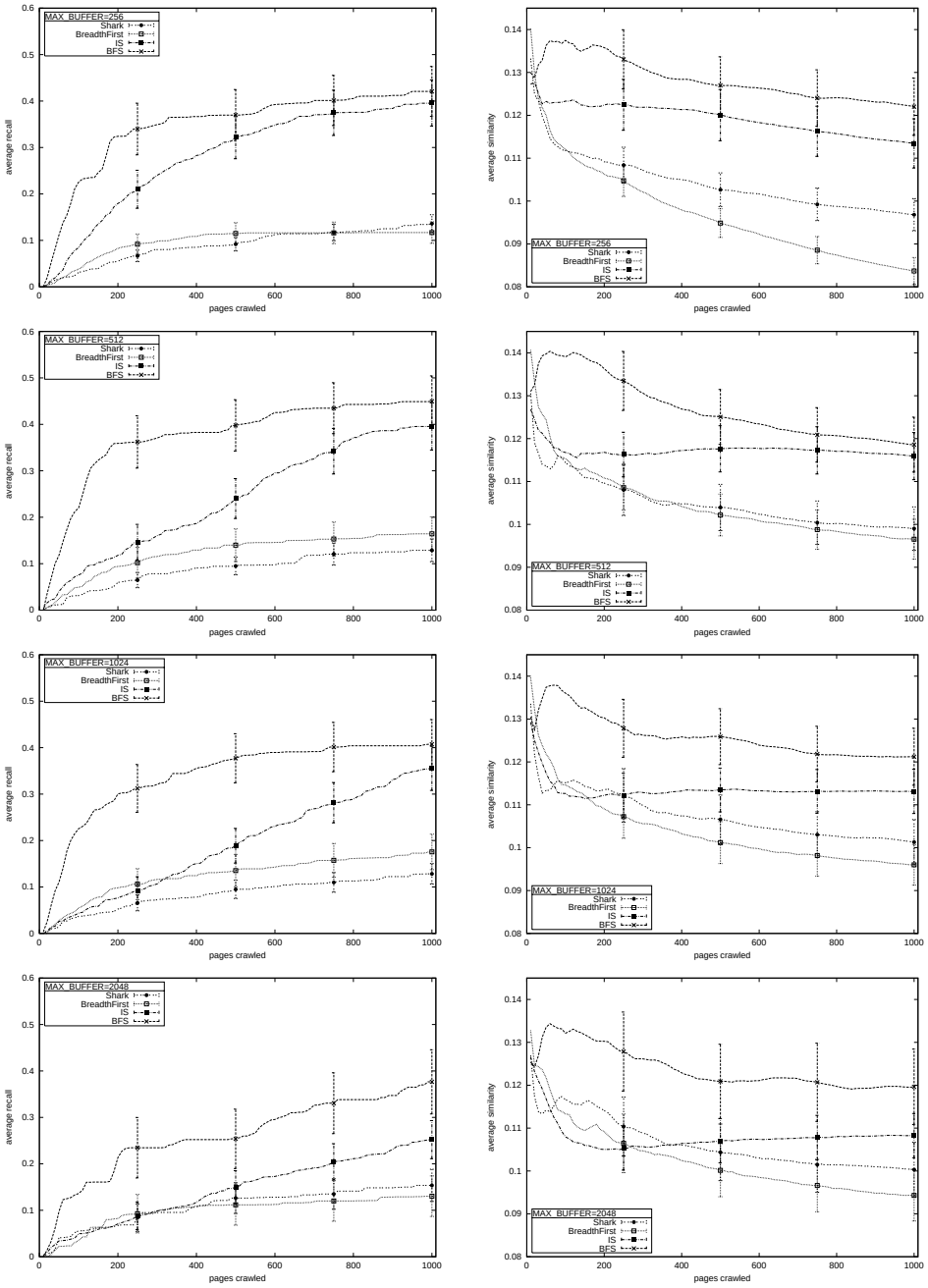

Fig. 12. Average target recall (left) and similarity to topic descriptions (right) by the Breadth-First, Best-First, Shark-Search and InfoSpiders crawlers on the $D = 3$ task. The plots in each row correspond to a different value of `MAX_BUFFER` between 256 and 2,048.

by the end of the crawls, except for the similarity metric in the MAX_BUFFER=2048 case, where InfoSpiders suffers a large initial loss from which it has not fully recovered after 1,000 pages.

While Best-First displays an advantage over InfoSpiders in the early stage of the crawls, by the time 1,000 pages are crawled the difference is no longer significant in most cases (with the exception of recall for MAX_BUFFER=2048). These plots seem to suggest that in the initial stages of the crawls BFS is helped by its greediness while InfoSpiders pays a penalty for its adaptability — the neural networks are not trained, the evolutionary bias has not yet had a chance to kick in, and the stochastic selector makes some locally suboptimal choices. However those features prove more useful in the later stages of the crawl. For example the similarity to topic descriptions seems to hold steady or improve slightly for InfoSpiders in 3 of the 4 graphs while all the other crawlers show a downward trend after 200 pages.

3.2.2 *Scalability.* Figure 12 shows that performance is affected by the value of the MAX_BUFFER parameter. Note that even when MAX_BUFFER > MAX_PAGES, performance can be affected by frontier size because the number of links encountered by a crawler is typically much larger than the number of pages visited. But while the limited buffer size forces any crawler to remove many links from its frontier, which links are eliminated depends on the crawler algorithm. To better assess how the various crawling algorithms scale with the space (memory) resources available to them, Figure 13 plots performance (target recall and similarity to topic descriptions at 1,000 pages) versus MAX_BUFFER.

Breadth-First, Shark-Search and Best-First do not appear to be affected by the frontier size, with one exception; similarity to topic descriptions increases significantly for Breadth-First when MAX_BUFFER goes from 256 to 512. This indicates that the limitation of space resources will hurt a crawling algorithm more if it cannot prioritize over its links, due to the higher risk of throwing away good links. For InfoSpiders, performance seems to decrease with increasing frontier size. We will discuss this effect in Section 7.

3.2.3 *Efficiency.* Let us now analyze the time complexity of the various crawling algorithms. We focus on CPU time, since all crawlers are equally subject to noisy network I/O delays. We also exclude the CPU time taken by common utility functions shared by all crawlers (cf. Section 2.1).

While we have made an effort to employ appropriate data structures and algorithms, naturally we can only evaluate the efficiency of our own implementations of the crawlers. Consider for example the data structure used to implement the frontier priority queue in the Best-First algorithm. The basic operations of inserting and removing links from the queue are most efficiently supported by a heap, yielding $O(\log(\texttt{MAX\_BUFFER}))$ complexity. However it is also necessary to check if a newly extracted URL is already in the frontier; this has linear complexity in a heap, and must be done for each link in a newly crawled page. Alternatively, using an associative array or hash table, this checking operation is cheap ($O(1)$) but inserting and removing links requires an expensive sorting operation ($O(\texttt{MAX\_BUFFER} \cdot \log(\texttt{MAX\_BUFFER}))$) once per page crawled. The asymptotic complexity of priority queue operations per crawled page is $O(L \cdot \texttt{MAX\_BUFFER})$ for the

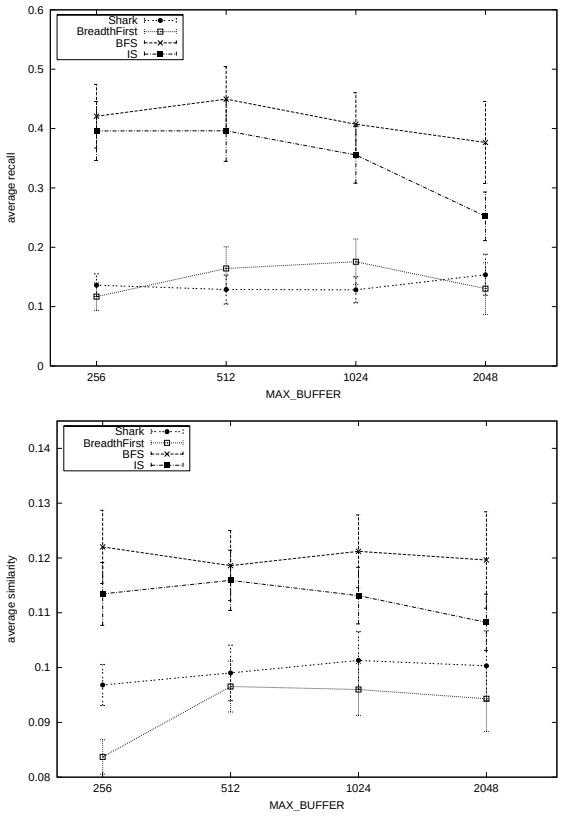

Fig. 13. Scalability of average target recall (top) and of similarity to topic descriptions (bottom) with MAX_BUFFER by the Breadth-First, Best-First, Shark-Search and InfoSpiders crawlers.

heap and $O(L + \texttt{MAX\_BUFFER} \cdot \log(\texttt{MAX\_BUFFER}))$ for the hash, where L is the average number of links per page. Figure 14 plots the actual costs for $L = 7$, which is an accurate estimate for the Web [Kumar et al. 2000]. The plot shows that while in the range of MAX_BUFFER used in our experiments the two data structures yield very similar efficiency, the hash is preferable for improved crawlers described in Section 4. For this reason we use an associative array to store the link frontier, not only in Best-First but in the other crawlers as well. Even so, further optimization may be possible for some of the crawlers and therefore we cannot claim that our efficiency analysis will necessarily apply to other implementations.

Another caveat is that the actual time taken by any algorithm is irrelevant since it depends largely on the load and hardware configuration of the machine(s) used to run the crawls — even the crawling experiments described in this paper were run on a large number of machines with widely different hardware configurations and changing loads. To obtain a measure of time complexity that can be useful to compare the efficiency of the crawlers, we introduce a *relative cost* c of crawler i as the ratio between the time taken by i to visit p pages and the mean cumulative time taken by a set of crawlers to visit MAX_PAGES pages:

$$c_p(i) = \frac{t_p(i) \cdot |C|}{\sum_{i' \in C} t_{\texttt{MAX\_PAGES}}(i')} \tag{8}$$

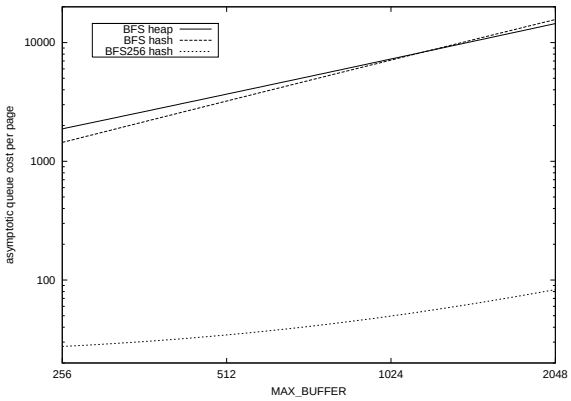

Fig. 14. Asymptotic complexity of priority queue operations in crawlers using heap and hash data structures. Higher L values ($L > 7$) and/or smaller `MAX_BUFFER` values make the comparison more favorable for the hash, and vice-versa. The BFS256 line corresponds to an algorithm described in Section 4, where the hash is sorted only once every 256 pages crawled.

where C is a set of crawlers and $i \in C$. The relative cost metric is robust with respect to crawling different topics from different machines, as long as all the crawlers run a single topic from the same machine.

Another useful metric would be one that combines the two sources of evidence for performance with efficiency. To this end let us define a *performance/cost* metric P as the ratio between performance and relative cost of a crawler i, where the former is given by the product of mean target recall and mean similarity to topic descriptions after p pages:

$$P_p(i) = \frac{\langle \rho_p(i) \rangle_Q \cdot \langle \sigma_p(i) \rangle_Q}{\langle c_p(i) \rangle_Q} \tag{9}$$

where Q is the set of topics.

Figure 15 plots both c_{1000} and P_{1000} versus `MAX_BUFFER` for the four crawlers. Shark-Search is the most expensive crawler and Breadth-First, as expected, is the most efficient. Breadth-First becomes more efficient for larger frontier size since it has to trim its frontier less frequently. Best-First has intermediate complexity. It becomes less efficient with increasing frontier size because there are more links to compare. What is most surprising, InfoSpiders is almost as efficient as Breadth-First, with the two time complexities converging for `MAX_BUFFER`=2,048. We discuss the decreasing complexity of InfoSpiders with increasing frontier size in Section 5.

The performance/cost ratio plot in Figure 15 yields InfoSpiders as the winner owing to its good performance and excellent efficiency. There is also a clear positive trend with increasing frontier size. Best-First pays a price for its complexity and is surpassed by Breadth-First for all but the smallest frontier size. Poor performance and inefficiency place Shark-Search in the last position. Clearly these results are strongly dependent upon the time complexity of the algorithms and their usefulness will depend on the particular application of the crawlers.

4. EXPLORATION VERSUS EXPLOITATION

In this section we raise the issue of exploration versus exploitation as a defining characteristic of a Web crawler. This issue is a universal one in machine learning and artificial intelligence, since it presents itself in any task where search is guided by quality estimations. Under some regularity assumption, a measure of quality at

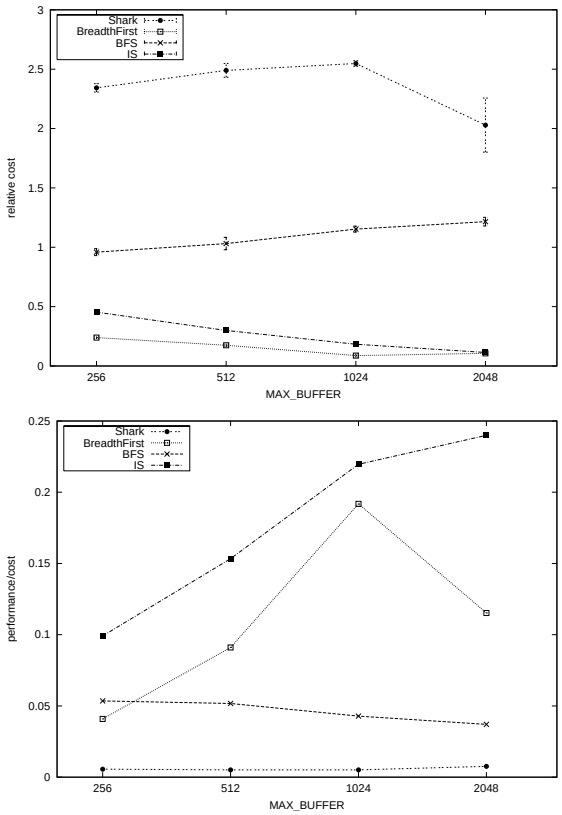

Fig. 15. Scalability of time complexity (top) and performance/cost ratio (bottom) with `MAX_BUFFER` by the Breadth-First, Best-First, Shark-Search and InfoSpiders crawlers.

one point in the search space provides some information on the quality of nearby points. A greedy algorithm can then exploit this information by concentrating the search in the vicinity of the most promising points. However, this strategy can lead to missing other equally good or even better points, for two reasons: first, the estimates may be noisy; and second, the search space may have local optima that trap the algorithm and keep it from locating global optima. In other words, it may be necessary to visit some "bad" points in order to arrive at the best ones. At the other extreme, algorithms that completely disregard quality estimates and continue to explore in a uniform or random fashion do not risk getting stuck at local optima, but they do not use the available information to bias the search and thus may spend most of their time exploring suboptimal areas. A balance between exploitation and exploration of clues is obviously called for in heuristic search algorithms, but the optimal compromise point is unknown unless the topology of the search space is well understood — which is typically not the case for interesting search spaces.

Topical crawlers fit into this picture very well if one views the Web as the search space, with pages as points and neighborhoods as defined by hyperlinks. A crawler must decide which pages to visit based on the cues provided by links from nearby pages. If one assumes that a relevant page has a higher probability to be near other relevant pages than to any random page, then quality estimate of pages provide cues that can be exploited to bias the search process. However, given the short

```
BFS_N (topic, starting_urls, N) {
   foreach link (starting_urls) {
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      links_to_crawl := dequeue_top_links(frontier, N);
      foreach link (randomize(links_to_crawl)) {
         doc := fetch(link);
         score :=  sim(topic, doc);
         merge(frontier, extract_links(doc), score);
         if (#frontier > MAX_BUFFER) {
            dequeue_bottom_links(frontier);
         }
      }
   }
}
```

Fig. 16. Pseudocode of BFSN crawlers. BFS1 ($N = 1$) corresponds to the algorithm in Figure 6.

range of relevance clues across the Web graph [Menczer 2004], a very relevant page might be only a few links behind an apparently irrelevant one. Balancing the exploitation of quality estimate information with exploration of suboptimal pages may be crucial for the performance of topical crawlers, something that we want to confirm empirically here.

4.1 Crawler Descriptions

In this section we describe generalized versions of the Best-First [Pant et al. 2002] and Shark-Search crawlers, designed to tune the greediness level of these crawlers and thus study the tradeoff between exploitation and exploration that is appropriate in a crawling application.

4.1.1 *Generalized Best-First.* Best-First was described in Section 3.1.3. BFSN is a generalization of BFS in that at each iteration a batch of top N links to crawl are selected. After completing the crawl of N pages the crawler decides on the next batch of N and so on. Figure 16 offers a simplified pseudocode of the BFSN class.

BFSN offers an ideal context for our study. The parameter N controls the greedy behavior of the crawler. Increasing N results in crawlers with greater emphasis on exploration and consequently a reduced emphasis on exploitation. Decreasing N reverses this; selecting a smaller set of links is more exploitative of the evidence available regarding the potential merits of the links. In our experiments we test five BFSN crawlers by setting N to 1, 16, 64, 128 and 256. Note that increasing N also results in increased efficiency because the frontier only needs to be sorted every N pages (cf. Figure 14).

4.1.2 *Generalized Shark-Search.* Shark-Search was described in Section 3.1.4. Figure 17 shows the pseudocode of our Shark$N$ class of crawlers, which generalizes the "standard" Shark algorithm in the same way that BFSN extends BFS. At each iteration a batch of top N links to crawl are selected. After completing the crawl of N pages the crawler decides on the next batch of N and so on. Thus Shark$N$ can be made more exploitative or explorative by varying N just as for Best-First.

```
Shark_N (topic, starting_urls, N) {
   foreach link (starting_urls) {
      set_depth(link, d);
      enqueue(frontier, link);
   }
   while (visited < MAX_PAGES) {
      links_to_crawl := dequeue_top_links(frontier, N);
      foreach link (randomize(links_to_crawl)) {
         doc := fetch(link);
         doc_score :=  sim(topic, doc);
         if (depth(link) > 0) {
            foreach outlink (extract_links(doc)) {
               score = (1-r) * neighborhood_score(outlink)
                      +  r   * inherited_score(outlink);
               if (doc_score > 0) {
                  set_depth(outlink, d);
               } else {
                  set_depth(outlink, depth(link) - 1);
               }
               enqueue(frontier, outlink, score);
            }
            if (#frontier > MAX_BUFFER) {
               dequeue_bottom_links(frontier);
            }
         }
      }
   }
}
```

Fig. 17. Pseudocode of Shark$N$ crawlers. Shark1 ($N = 1$) corresponds to the algorithm in Figure 8.

4.1.3 *Greediness in Other Crawlers.* Any crawler algorithm can be made more explorative or exploitative by regulating the amount of noise in its link estimates. In the case of PageRank, the frequency with which PageRank calculations are redone is key to its greediness level; the more frequent the calculations, the more exploitative the crawler.

In InfoSpiders we can regulate greediness by changing the value of β, the inverse-temperature parameter used in the stochastic selector (Equation 6). We increase β to get greater exploitation and decrease it for greater exploration, with the extreme case $\beta = 0$ generating a uniform probability distribution over the frontier.

4.2 Experimental Results

Here we focus on the BFSN and Shark$N$ classes of crawlers whose behavior is the most straightforward to interpret in terms of the exploration parameter N. The results in this section summarize and extend preliminary experiments [Pant et al. 2002] showing that for BFSN crawlers, $N = 256$ yields the best performance under a number of measures, for $2^0 \leq N \leq 2^8$.

4.2.1 *Dynamic Performance.* The dynamic results are summarized in Figure 18. The parameter `MAX_BUFFER` is set to 256 links. For readability, we are only plotting the performance of a selected subset of the BFSN crawlers ($N = 1$ and $N =$

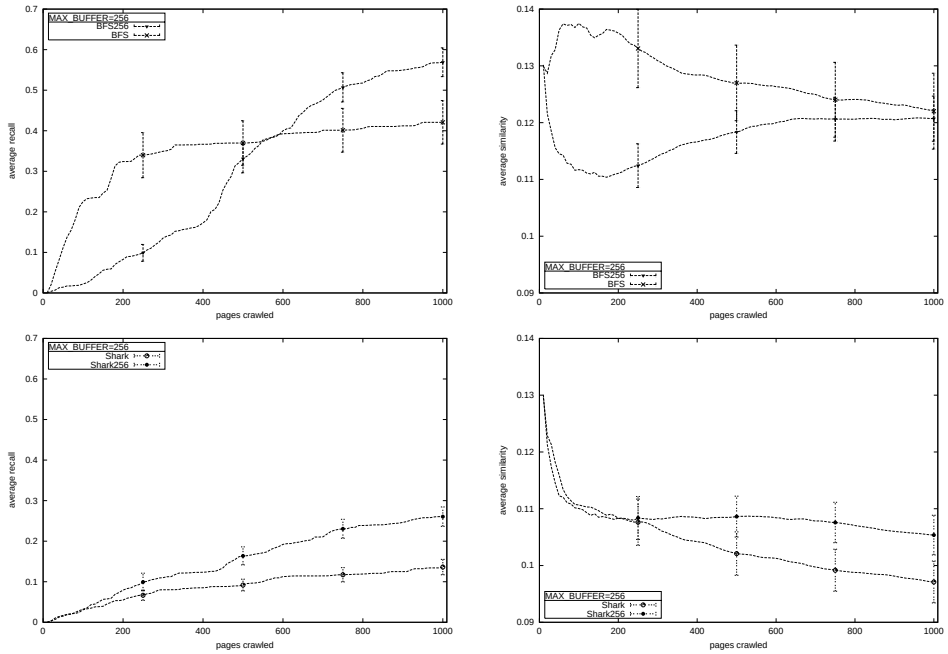

Fig. 18. Average target recall (left) and similarity to topic descriptions (right) by representatives of the BFSN (top) and Shark$N$ (bottom) crawling algorithms, for $N = 1$ and $N = 256$.

256). The behavior of the remaining crawlers (BFS16, BFS64 and BFS128) can be extrapolated between the curves corresponding to BFS1 and BFS256. The same holds for Shark$N$ crawlers.

In the case of Shark$N$, it is clear that exploration helps. However the performance of Shark$N$ crawlers in general is significantly worse than that of BFSN crawlers after the first 250 pages. Generalized Best-First also paints a more interesting picture. We observe that the more exploratory algorithm, BFS256, seems to bear an initial penalty for exploration but recovers in the long run; after about 500 pages BFS256 catches up, and in the case of recall BFS256 eventually surpasses BFS1. These results are consistent with the observations from the previous section — the greediness of BFS1 helps early on but hinders performance in the longer run. It is also noteworthy that pure (blind) exploration alone is not sufficient to achieve the best performance. If it were, Breadth-First would be the winner, however Figure 12 shows that Breadth-First is outperformed by BFS. We conclude that a successful crawling strategy must employ a combination of explorative and exploitative bias, as is generally the case in any interesting search problem with local optima.

Incidentally, inspection of Figures 12 and 18 reveals that the best overall performance after 1,000 pages crawled in the $D = 3$ task is achieved by BFS256 with `MAX_BUFFER`=256 links.

4.2.2 *Scalability.* Figure 19 plots the performance of BFS1 and BFS256 at 1,000 pages versus frontier size, for `MAX_BUFFER` between 256 and 1,024. No significant trends are observed; BFSN seems to be unaffected by the frontier size in this range.

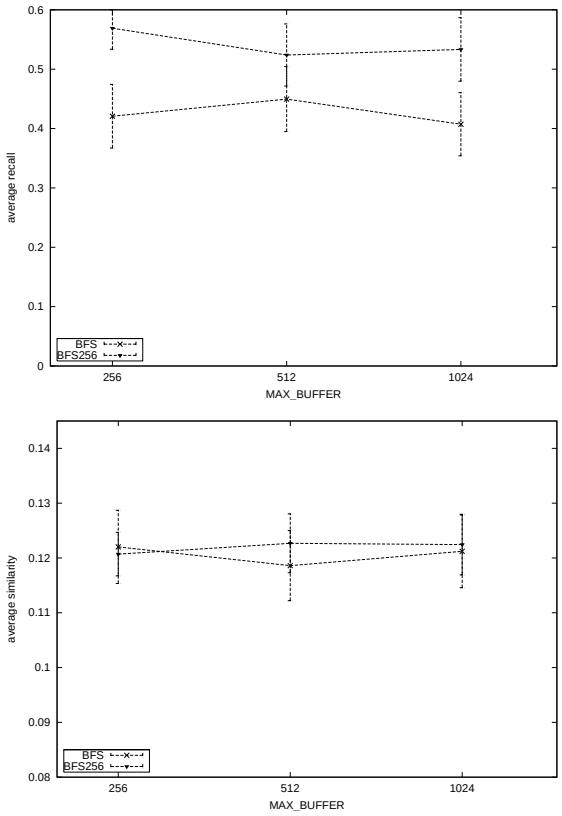

Fig. 19. Scalability of average target recall (top) and of similarity to topic descriptions (bottom) with `MAX_BUFFER` by representatives of the BFSN crawling algorithms, for $N = 1$ and $N = 256$.

5. ADAPTIVE CRAWLERS

The Web is a very heterogeneous space. The value of its link and lexical cues varies greatly depending on the purpose of pages. For example, a researcher's homepage might carefully link to pages of colleagues who do related work, while a commercial site might carefully avoid any pointers to the competition. In fact we have shown that pages in the `.edu` domain tend to link to lexically similar pages with a significantly higher likelihood than pages in the `.com` domain [Menczer 2004].

One question that arises from this observation, for topical crawlers, is whether crawling algorithms should take advantage of *adaptive* techniques to change their behavior based on their experiential context — if I am looking for "rock climbing" pages and the term "rock" has often misled me into following links about music rather than sports, it might be wise to give more importance to the term "climbing" than to "rock" in the future. In our previous research we have shown that this type of spatial and temporal context can be exploited by individual and population-based adaptive crawling algorithms [Menczer and Belew 2000], but those experiments were limited to relatively small and well-organized portions of the Web such as the Encyclopaedia Britannica Online.[7]

[7] `http://www.eb.com`

5.1 Crawler Description

In this section we study whether the noisy Web "at large" provides crawlers with signals that can be internalized to improve crawling performance in the course of a crawl. Such signals would have to be reliable enough to help, rather than hinder an agent's capability to focus its search. The InfoSpiders crawling algorithm described in Section 3.1.5 was designed principally to explore this question, therefore there are a number of adaptive mechanisms built into InfoSpiders.

At the individual level, an agent's neural network weights can be adjusted by reinforcement so that an agent may learn to predict the value of links. In the example above, the weight from the unit associated with the term "climbing" should be made larger relative to the weight associated with the term "rock." This would happen if an agent followed some links highly rated by the neural net because of their proximity with "rock," only to find that the pages pointed by those links had a small similarity with the topic "rock climbing." The algorithm uses the error in similarity prediction as a reinforcement signal to train the neural network into making more accurate link evaluations. To study the effect of individual learning on crawling performance, we consider variations of the InfoSpiders algorithm in which learning is disabled and all neural networks are frozen at their initial configurations, with identical weights.

At the population level, there are two adaptive mechanisms in InfoSpiders. Through selective reproduction, the search is biased to favor the portions of the Web that appear more promising. If an agent encounters several pages with high similarity to its topic keywords, that may be an indication that its frontier contains higher quality links than other agents. Agents can locally internalize this information by creating offspring when their energy surpasses the reproduction threshold. This in practice doubles the frequency with which the links of the parent agent are going to be jointly explored by the parent and offspring agents. Since for fairness the collective frontier of the population as a whole must be maintained within the `MAX_BUFFER` boundary, at reproduction the local frontier (and its size) is split between parent and offspring. This implies a cost for such an adaptive bias; as the crawling algorithm becomes more distributed, local link estimates will be based on smaller frontiers and thus more subject to local fluctuations. The `MAX_POP_SIZE` parameter determines the maximum degree of distributedness of the algorithm. We explore values of 8 and 1 here to study the tradeoff in performance between a distributed crawling strategy with evolutionary bias and a centralized one.

The second adaptive mechanism stemming from InfoSpiders' evolutionary bias is a *selective expansion* of the topic keywords. At reproduction, the offspring agent picks up a new term from the page where the parent is currently "situated." Recall that each agent maintains a biased random walk through the pages it visits, so that with high probability its "current location" is a page with high similarity to its topic keywords (cf. Figure 10 and Equation 7). The offspring thus can potentially improve on its crawling performance by internalizing a new lexical feature that appears positively correlated with relevance estimates. The risk, of course, is to add noise by paying attention to a term that in reality is not a good indicator of relevance. Selective expansion of topic keywords is disabled when the `MAX_POP_SIZE` parameter is set to 1.

5.2 Experimental Results

Here we compare four variations of InfoSpiders crawlers — with and without reinforcement learning enabled at the individual level, and with and without evolution enabled at the population level. Values between 256 and 2,048 are again explored for the MAX_BUFFER parameter.

5.2.1 *Dynamic Performance.* Figure 20 shows the dynamic performance plots of the four InfoSpiders variations. Few differences are significant but we can still observe some general trends. When the frontier is small (MAX_BUFFER=256) both evolution and learning seem to hinder target recall. The problem with evolution is understandable since with a small frontier, each agent has a very small set of links to work with and thus the risk to make poor decisions increases with the decentralization of the link selection process. Evolution does not seem to hinder (nor help) similarity to topic descriptions; there are probably more pages similar to the target descriptions than targets, so missing a target does not necessarily imply missing other possibly relevant pages.

The negative impact of learning on performance for small frontier is less clear and deserves further analysis. Figure 21 plots the errors made by an InfoSpiders agent (MAX_POP_SIZE=1) in estimating the similarity of the pages pointed by the links that are actually followed. The plot shows both the error of the neural net (used for training) and the error of the actual estimates used to pick the links, obtained by combining the neural net output with the similarity of the parent page (cf. Section 3.1.5). While the actual estimates do improve over time, the neural net's errors are subject to significant noise, with a sharp initial decrease followed by an increase after 100-300 pages, followed by a slower decrease. This data shows that the heterogeneity of the Web can lead an agent astray, especially after a few hundred pages. The negative effect of learning can thus be interpreted by considering the risk that spawned agents may not see enough pages to learn an effective representation.

When the frontier is large (cf. Figure 20, MAX_BUFFER=2,048) there is a reversal for similarity to topic descriptions; the dominant factor seems to be evolution while learning does not appear to have an effect. The fact that evolution helps here can be explain by considering that a large frontier allows each agent to retain an adequate number ($2,048/8 = 256$) of links to pick from. Thus decentralizing the search is less risky and the crawler can take advantage of the positive effects of evolution, namely biasing the search toward promising areas and selective keyword expansion.

Incidentally, inspection of Figures 12, 18 and 20 reveals that InfoSpiders (with no evolution and no learning) competes head-to-head with BFS256 for the best overall performance after 1,000 pages crawled in the $D = 3$ task with MAX_BUFFER=256 links. While we are pleased to have introduced the two best-performing crawlers, we find it somewhat disappointing that the adaptive capabilities of InfoSpiders appear to do more harm than good in these short crawls with very limited resources.

5.2.2 *Scalability.* The plots in Figure 22 show that performance in all of the InfoSpiders variants does not scale very well with the frontier size. Again this negative trend will be discussed in Section 7. However, few of the differences between performance of variants (at 1,000 pages) are significant. As was observed in Figure 20,

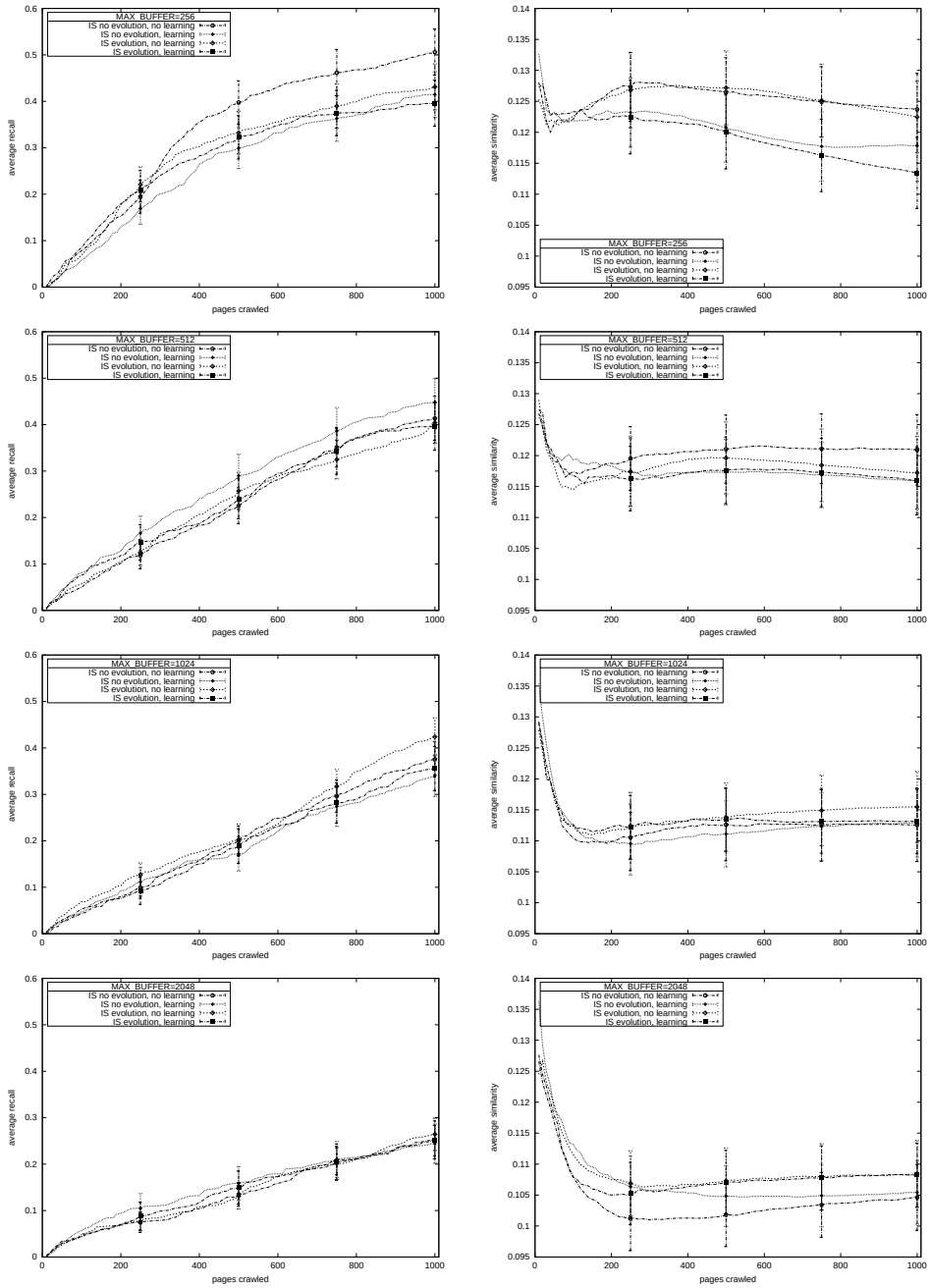

Fig. 20. Average target recall (left) and similarity to topic descriptions (right) by the four variants of InfoSpiders crawlers. The plots in each row correspond to a different value of `MAX_BUFFER`.

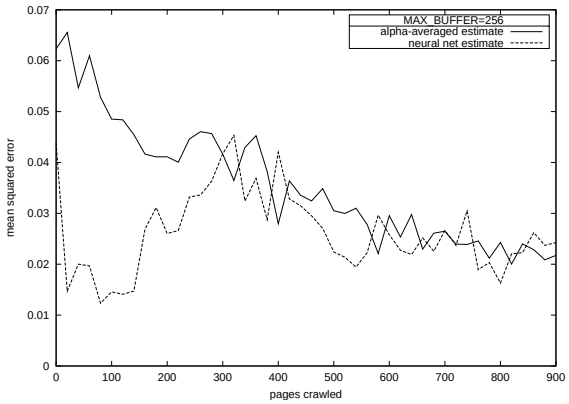

Fig. 21. Link estimate error of a single InfoSpiders agent, averaged across 8 topics.

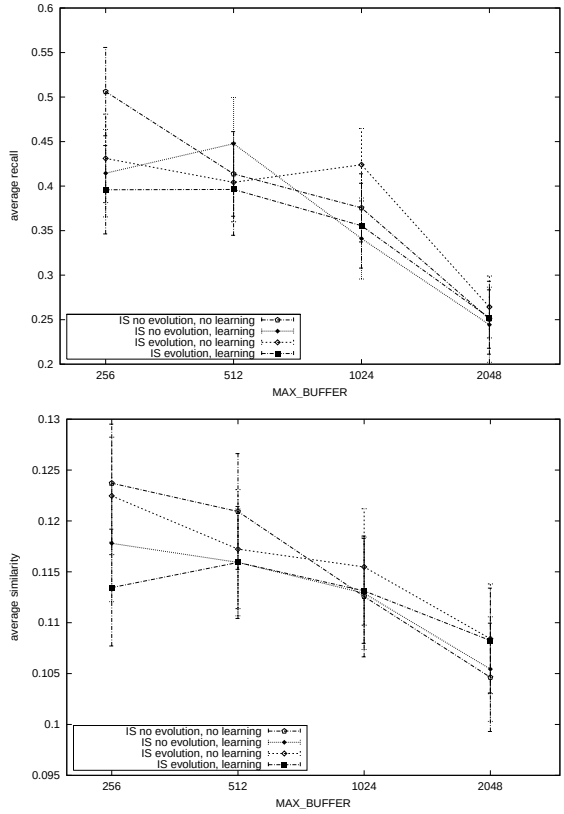

Fig. 22. Scalability of average target recall (top) and of similarity to topic descriptions (bottom) with `MAX_BUFFER` by the four variants of InfoSpiders crawlers.

the largest spread is observed for small frontier size (`MAX_BUFFER`=256), where both learning and evolution hinder target recall, and learning hinders similarity to topic descriptions as well.

5.2.3 *Efficiency.* The complexity plot in Figure 23 reveals some interesting results. As expected, learning increases the complexity of the crawler since it requires extra time to train the neural nets. But evolution decreases the complexity, and

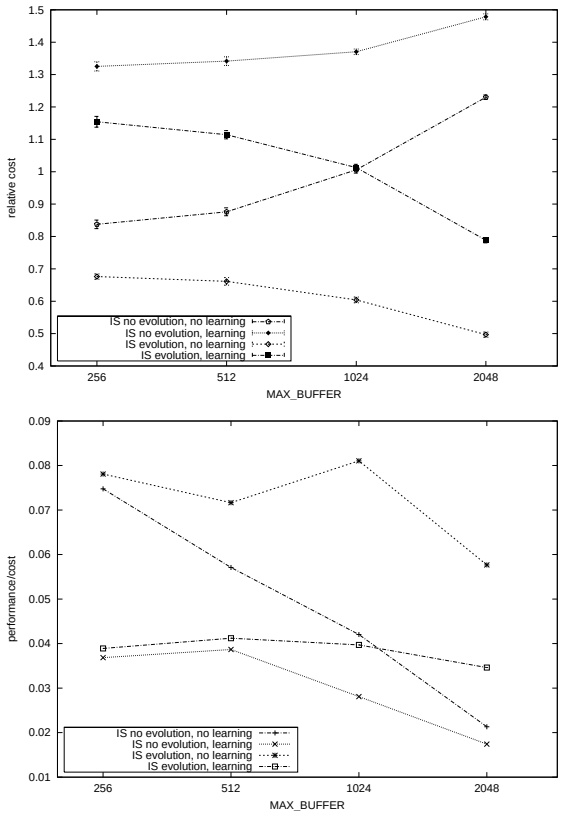

Fig. 23. Scalability of time complexity (top) and performance/cost ratio (bottom) with MAX_BUFFER by the four variants of InfoSpiders crawlers.

more so for larger frontier size as we have also seen in Figure 15. The increased efficiency of evolution stems from the fact that the distributed algorithm divides the problem into simpler ones, each tackled by an independent agent. The agent population cumulatively has a simpler job than a centralized agent. In the absence of evolution, complexity increases with MAX_BUFFER as there is more work to do in order to centrally manage a larger frontier of links. But in the presence of evolution the trend is reversed to the point that for MAX_BUFFER=1,024 the extra advantage of evolution counterbalances the extra cost of learning. We interpret the trend by imagining that the population size grows faster as a larger frontier must trigger earlier reproduction events.

The performance/cost plot in Figure 23 closely follows the relative cost of the algorithms since the differences in performance are not significant. The data from these short crawls suggest that InfoSpiders should use evolution but not learning.

6. LONG CRAWLS

All the crawls considered thus far are very short, visiting 1,000 pages. This is appropriate in certain circumstances, e.g. while the user is waiting [Pant and Menczer 2002]. However, the behavior of the crawling algorithms might turn out to be quite different upon crawling a much larger number of pages. To explore this question, in this section we briefly consider somewhat longer crawls of 50,000 pages.

6.1 Crawler Descriptions

We select a few representative crawlers among those described in this paper to see if their relative performance qualitatively changes as more pages are crawled. The naive BFS256 crawler is an obvious candidate because it displays one of the best performances. InfoSpiders without learning is also a strong performer in short crawls, however we are more interested in studying whether learning might turn out to be helpful in the longer run so we select the "normal" InfoSpiders with both evolution and learning. Finally, Breadth-First is selected as usual to provide a baseline. Note that in the theoretical limit for very long crawls (`MAX_PAGES` $\to \infty$) one might expect all crawlers to converge ($\sigma \to 0$ and hopefully $\rho \to 1$). The question is how fast this happens for each crawler. The best crawler is the one such that σ converges slowest and ρ converges to the highest level. Breadth-First is a helpful baseline because it provides a lower bound for σ.

6.2 Experimental Results

We ran 50,000-page crawls for BFS256, InfoSpiders and Breadth-First for 10 topics, with `MAX_BUFFER`=2,048 links. Note that the number of topics is limited by the time required to run these longer crawls.

Figure 24 shows the dynamic performance of the three crawlers in the longer crawls. The error bars can be quite large owing to the small number of topics. As expected, Breadth-First is significantly outperformed by both BFS256 and InfoSpiders for both target recall and similarity to topic descriptions.

For target recall BFS256 initially beats InfoSpiders (consistently with the results from shorter crawls), but eventually after 7,500 pages InfoSpiders surpasses BFS256, with the difference becoming significant after 20,000 pages. Note that all the three crawlers seem to have converged after about 20,000 pages, therefore we do not expect to learn much more from running even longer crawls. The final recall level of almost 70% of the targets achieved by InfoSpiders seems like a pretty good result considering the difficulty of the $D = 3$ task.

For similarity to topic descriptions the differences between BFS256 and InfoSpiders are not significant, however the trends paint a very similar picture. BFS256 performs better than InfoSpiders in the early phase of the crawl, while InfoSpiders eventually catches up by staying more focused. These results suggest that while the adaptive mechanisms of InfoSpiders may not be helpful (or may even be harmful) in very short crawls, they confer a winning advantage after the agents have had a chance to internalize important features about the environment. Such features allow the agents to discover relevant pages long after static crawlers have lost their focus.

7. CONCLUDING REMARKS

We believe that the crawling algorithms discussed in this paper are a good sample of the current state of the art. Of course there are many other algorithms that are not considered here due to timeliness, insufficient information available, or our own ignorance. Even for the most sophisticated crawlers described in the literature it is impossible to gauge their effectiveness in isolation, without fair comparative studies using well defined common tasks, performance measures, and equal resources. It is

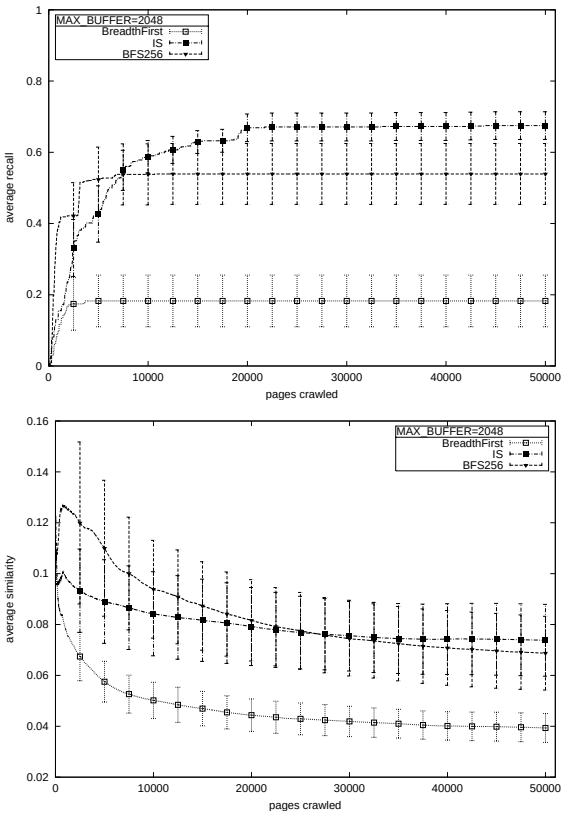

Fig. 24. Average target recall (top) and similarity to topic descriptions (bottom) by BFS256, InfoSpiders and Breadth-First over ten 50,000 page crawls.

our hope that by describing the state of the art in topical crawlers within a unified evaluation framework, one can foster further studies that will lead to improved crawling algorithms and ultimately better search tools for the public at large.

Let us conclude by summarizing the main contributions of this paper, discussing our results, and outlining some directions for future work.

7.1 Summary

The major contributions of the present work are summarized as follows:

— Two novel crawling algorithms, BFS256 and InfoSpiders, have been introduced by generalizing or improving on previous algorithms. Based on our evaluation metrics, and to the best of our knowledge, these crawlers are currently the best performers at difficult crawling tasks such as the one in which seed URLs are $D = 3$ links away from known relevant targets. Recall levels above 60% are achieved by InfoSpiders after around 10,000 pages.

— A number of crawlers from the literature have been reimplemented and evaluated under fair conditions. Their relative effectiveness has been shown to be consistent across different performance metrics and space (memory) resources.

— The naive Best-First algorithm has been shown to be an effective crawler, especially in the short run. For very short crawls it is best to be greedy; this is a

lesson that should be incorporated into algorithms for query time (online) crawlers such as MySpiders[8] [Pant and Menczer 2002].

— We have shown consistently that exploration leads to better coverage of highly relevant pages, in spite of a possible penalty during the early stage of the crawl. This suggests that exploration allows to tradeoff short term gains for longer term and potentially larger gains. However, we also found that a blind exploration leads to poor results, demonstrating that a mix of exploration and exploitation is necessary for good overall performance.

— Our study of adaptive crawlers has demonstrated that machine learning techniques making use of local information alone can confer a significant advantage in performance after sufficiently long crawls. However, while InfoSpiders is a very competitive crawler in general, we have shown that reinforcement learning does not provide for a sufficiently strong advantage in predicting the value of links to offset the increased cost and the risk of additional mistakes during the initial training phase.

— We have analyzed the time complexity of crawlers and found that distributed algorithms such as InfoSpiders can be extremely efficient and scalable, improving for large amounts of available resources to the point of being no more expensive that a blind algorithm such as Breadth-First. InfoSpiders thus achieves the best performance/cost ratio. Greedy algorithms such as Best-First, on the other hand, require a large amount of time to sort through candidate links and are less scalable.

7.2 Discussion

We find it somewhat disappointing that adaptation does not yield a greater advantage in performance, and that reinforcement learning in particular seems to hinder performance in short crawls. Clearly the Web is a very noisy environment and mistakes can be costly for tasks such as $D = 3$. It would be interesting to repeat the experiment with adaptive crawlers on simpler tasks such as $D = 0, 1, 2$. Another factor to consider is that there are several parameters in InfoSpiders, which have been tuned preliminarily but not extensively and not in the $D = 3$ task. Tuning some of those parameters, for example the learning rate for training the neural nets or the greediness parameter β, might prove beneficial.

We observed InfoSpiders becoming more efficient for large frontier sizes, but their performance parallelly worsening. This phenomenon deserves further study. Our current theory is that a larger frontier may lead to earlier reproduction, thus explaining the gain in efficiency. Such early reproduction might be seen as a sort of "premature bias," or suboptimal exploitation of early evidence that leads to missing other good pages later. In this sense a better understanding of optimal reproduction frequencies should be obtained by further tuning the reproduction threshold parameter (`THETA`). Of course one must also consider that the simple growth of the frontier implies an inherent increase in noise with which crawlers may have a hard time coping.

The present analysis is based on a small number of evaluation metrics selected for the particular task discussed in this paper. Many other measures could of course

[8] `http://myspiders.informatics.indiana.edu/`

be used. Furthermore, all the analyses in this paper are based on averages across topics. Much can be learned by taking a different approach, namely studying diverse characteristics of each topic and then observing whether certain topical features are associated with particularly good (or poor) performance by any given crawler. These are issues that we discuss in a companion paper focusing on the evaluation aspects of crawlers [Srinivasan et al. 2004].

In this study we have devoted significant resources to obtain statistically significant results when possible by analyzing data across many topics. This, together with the large number of experiments and crawlers, and given obvious limitations in hardware, bandwidth and time, severely constrained the length of the crawls that could be carried out. Longer crawls are of course always desirable, especially when considering applications such as in support of topical portals [Pant et al. 2003]. However it must be stressed that it takes many topics (or at least random replicates) to obtain reliable results and that such reliability is key to understanding the behavior of topical crawlers in such a complex environment as the Web.

ACKNOWLEDGMENTS

We are grateful to Soumen Chakrabarti, Steve Lawrence, Rik Belew, Nick Street and Dave Eichmann for helpful comments since the early stages of this project, to several anonymous reviewers for constructive criticism and suggestions, and to Alberto Segre and Dave Eichmann for computing support.

REFERENCES

Aggarwal, C., Al-Garawi, F., and Yu, P. 2001. Intelligent crawling on the World Wide Web with arbitrary predicates. In *Proc. 10th International World Wide Web Conference*. 96–105.

Ben-Shaul, I. et al. 1999. Adding support for dynamic and focused search with Fetuccino. *Computer Networks 31,* 11–16, 1653–1665.

Brewington, B. E. and Cybenko, G. 2000. How dynamic is the Web? In *Proc. 9th International World-Wide Web Conference*.

Brin, S. and Page, L. 1998. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks 30,* 1–7, 107–117.

Chakrabarti, S., van den Berg, M., and Dom, B. 1999. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks 31,* 11–16, 1623–1640.

Cho, J. and Garcia-Molina, H. 2000. The evolution of the Web and implications for an incremental crawler. In *Proceedings of the 26th International Conference on Very Large Databases (VLDB)*.

Cho, J., Garcia-Molina, H., and Page, L. 1998. Efficient crawling through URL ordering. *Computer Networks 30,* 1–7, 161–172.

Cyveillance. 2000. Sizing the internet. White paper. http://www.cyveillance.com/.

De Bra, P. and Post, R. 1994. Information retrieval in the World Wide Web: Making client-based searching feasible. In *Proc. 1st International World Wide Web Conference* (Geneva).

Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. 2000. Focused crawling using context graphs. In *Proc. 26th International Conference on Very Large Databases (VLDB 2000)*. Cairo, Egypt, 527–534.

Flake, G., Lawrence, S., Giles, C., and Coetzee, F. 2002. Self-organization of the Web and identification of communities. *IEEE Computer 35,* 3, 66–71.

Gibson, D., Kleinberg, J., and Raghavan, P. 1998. Inferring Web communities from link topology. In *Proc. 9th ACM Conference on Hypertext and Hypermedia*. 225–234.

Haveliwala, T. 1999. Efficient computation of pagerank. Tech. rep., Stanford Database Group.

Henzinger, M., Heydon, A., Mitzenmacher, M., and Najork, M. 1999. Measuring search engine quality using random walks on the Web. In *Proc. 8th International World Wide Web Conference* (Toronto). 213–225.

Hersovici, M., Jacovi, M., Maarek, Y. S., Pelleg, D., Shtalhaim, M., and Ur, S. 1998. The shark-search algorithm — An application: Tailored Web site mapping. In *Proc. 7th Intl. World-Wide Web Conference*.

Kleinberg, J. 1999. Authoritative sources in a hyperlinked environment. *Journal of the ACM 46,* 5, 604–632.

Kleinberg, J. and Lawrence, S. 2001. The structure of the Web. *Science 294,* 5548, 1849–1850.

Kumar, S., Raghavan, P., Rajagopalan, S., Sivakumar, D., Tomkins, A., and Upfal, E. 2000. Stochastic models for the Web graph. In *Proc. 41st Annual IEEE Symposium on Foundations of Computer Science*. IEEE Computer Society Press, Silver Spring, MD, 57–65.

Kumar, S., Raghavan, P., Rajagopalan, S., and Tomkins, A. 1999. Trawling the Web for emerging cyber-communities. *Computer Networks 31,* 11–16, 1481–1493.

Lawrence, S. and Giles, C. 1998. Searching the World Wide Web. *Science 280*, 98–100.

Lawrence, S. and Giles, C. 1999. Accessibility of information on the Web. *Nature 400*, 107–109.

McCallum, A., Nigam, K., Rennie, J., and Seymore, K. 1999. A machine learning approach to building domain-specific search engines. In *Proc. 16th International Joint Conf. on Artificial Intelligence*. Morgan Kaufmann, San Francisco, CA, 662–667.

Menczer, F. 1997. ARACHNID: Adaptive Retrieval Agents Choosing Heuristic Neighborhoods for Information Discovery. In *Proc. 14th International Conference on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 227–235.

Menczer, F. 2003. Complementing search engines with online Web mining agents. *Decision Support Systems 35,* 2, 195–212.

Menczer, F. 2004. Lexical and semantic clustering by Web links. *Journal of the American Society for Information Science and Technology*. Forthcoming.

Menczer, F. and Belew, R. 1998. Adaptive information agents in distributed textual environments. In *Proc. 2nd International Conference on Autonomous Agents*. Minneapolis, MN, 157–164.

Menczer, F. and Belew, R. 2000. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning 39,* 2–3, 203–242.

Menczer, F. and Monge, A. 1999. Scalable Web search by adaptive online agents: An InfoSpiders case study. In *Intelligent Information Agents: Agent-Based Information Discovery and Management on the Internet*, M. Klusch, Ed. Springer, Berlin, 323–347.

Menczer, F., Pant, G., Ruiz, M., and Srinivasan, P. 2001. Evaluating topic-driven Web crawlers. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, D. H. Kraft, W. B. Croft, D. J. Harper, and J. Zobel, Eds. ACM Press, New York, NY, 241–249.

Moukas, A. and Zacharia, G. 1997. Evolving a multi-agent information filtering solution in Amalthaea. In *Proc. 1st International Conference on Autonomous Agents*.

Najork, M. and Wiener, J. L. 2001. Breadth-first search crawling yields high-quality pages. In *Proc. 10th International World Wide Web Conference*.

Nick, Z. and Themis, P. 2001. Web search using a genetic algorithm. *IEEE Internet Computing 5,* 2, 18–26.

O'Meara, T. and Patel, A. 2001. A topic-specific Web robot model based on restless bandits. *IEEE Internet Computing 5,* 2, 27–35.

Pant, G., Bradshaw, S., and Menczer, F. 2003. Search engine - crawler symbiosis. In *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, T. Koch and I. Solvberg, Eds. Lecture Notes in Computer Science, Vol. 2769. Springer Verlag, Berlin.

Pant, G. and Menczer, F. 2002. MySpiders: Evolve your own intelligent Web crawlers. *Autonomous Agents and Multi-Agent Systems 5,* 2, 221–229.

Pant, G. and Menczer, F. 2003. Topical crawling for business intelligence. In *Proc. 7th European Conference on Research and Advanced Technology for Digital Libraries (ECDL)*, T. Koch and I. Solvberg, Eds. Lecture Notes in Computer Science, Vol. 2769. Berlin.

Pant, G., Srinivasan, P., and Menczer, F. 2002. Exploration versus exploitation in topic driven crawlers. In *Proc. WWW-02 Workshop on Web Dynamics*.

Pinkerton, B. 1994. Finding what people want: Experiences with the WebCrawler. In *Proc. 2nd International World Wide Web Conference* (Chicago).

Porter, M. 1980. An algorithm for suffix stripping. *Program 14,* 3, 130–137.

Rennie, J. and McCallum, A. 1999. Using reinforcement learning to spider the Web efficiently. In *Proc. 16th International Conf. on Machine Learning*. Morgan Kaufmann, San Francisco, CA, 335–343.

Rumelhart, D., Hinton, G., and Williams, R. 1986. Learning internal representations by error propagation. In *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, D. Rumelhart and J. McClelland, Eds. Vol. 1. Bradford Books (MIT Press), Cambridge, MA, Chapter 8, 318–362.

Srinivasan, P., Pant, G., and Menczer, F. 2004. A general evaluation framework for topical crawlers. *Information Retrieval*. Forthcoming.

Wills, C. and Mikhailov, M. 1999. Towards a better understanding of Web resources and server responses for improved caching. In *Proc. 8th International World Wide Web Conference* (Toronto).

Received May 2002; revised December 2002; accepted February 2003