Search Engine-Crawler Symbiosis: Adapting to Community Interests

Gautam Pant, Shannon Bradshaw, and Filippo Menczer**

Department of Management Sciences
The University of Iowa, Iowa City IA 52242, USA
email: {gautam-pant,shannon-bradshaw,filippo-menczer}@uiowa.edu

Abstract. Web crawlers have been used for nearly a decade as a search engine component to create and update large collections of documents. Typically the crawler and the rest of the search engine are not closely integrated. If the purpose of a search engine is to have as large a collection as possible to serve the general Web community, a close integration may not be necessary. However, if the search engine caters to a specific community with shared focused interests, it can take advantage of such an integration. In this paper we investigate a tightly coupled system in which the crawler and the search engine engage in a symbiotic relationship. The crawler feeds the search engine and the search engine in turn helps the crawler to better its performance. We show that the symbiosis can help the system learn about a community's interests and serve such a community with better focus.

1 Introduction

General purpose search engines have typically used exhaustive crawlers to build and update large collections of documents. The cost of crawling and indexing the collections is amortized over millions of queries received by the search engines. However, the large size, the dynamic nature and the diversity of the Web warrant more focused solutions that allow for direct and indirect collaboration amongst Web searchers of similar interests. Such solutions may be more scalable and distributable while being effective and efficient. They may also lead to new ways of searching that are hard to imagine with the centralized exhaustive approach followed by the current search engines.

Web communities with focused interests can be found in many professional and casual settings. For example, a set of software engineers at their workplace may have very focused interests. It would not be surprising if most of their queries to search engines on a given day were around a narrow topic. In fact, a small collection of Web pages (say, 50,000 pages) may satisfy most of their needs for a short period of time. However, it is non-trivial to identify a small focused collection, out of a much larger Web, that is representative of a community's

** Current affiliation: School of Informatics and Computer Science Department, Indiana University. Email: fil@indiana.edu

interests. It is also important to tweak the collection with time so that it remains focused on the current interests of the community. Given that we have such a collection for a community, a search engine that indexes it may be able to better serve the community. For example, owing to the small size of the collection, the search engine can be kept extremely fresh by crawling on a daily or weekly basis. In addition, more sophisticated information retrieval, text mining and visualization tools may be applied that are not as efficient and/or effective for a much larger corpus. Also, such a system can provide collaboration opportunities between the users.

If the purpose of a search engine is to have as large a collection as possible to serve the general Web community, a close integration between the crawler and the other components of the engine may not be necessary. The main goal of the crawler is to keep the coverage and freshness of the search engine index high, and this task is not informed by user interactions. For this reason, the crawler and the rest of the search engine typically have little or no communication between them. Some search engines may use crawlers that focus their crawl using heuristics similar to those used by the search engine to rank the documents [1]. But they do not closely integrate the crawling algorithm with the search engine.

In this paper we discuss a particular Web searching model in which a *topical* or *focused* crawler and a search engine engage in a mutually beneficial relationship in order to cater to a particular community of users. Specifically, the goal is to incrementally refine a collection on a broad set of topics in order to focus the collection on the set of topics relevant to the community. The flexibility of the model allows for adaptation to drifting interests in the community.

2 Symbiosis between Search Engine and Crawler

The search engine-crawler[1] symbiotic system could live on a large server or a desktop machine. It could be serving a single user or a set of users. The system tightly couples a search engine and a crawler to learn from the user queries given to the search engine in the recent past. A learning process, involving search engine-crawler symbiosis, is used to prepare a focused collection that may be better suited for queries in the near future. The process is repeated on a daily or a weekly basis. The queries serve as an approximation for the interests of a set of users. We can use all or a sample of the recent queries as *representative* requests. These representative queries are used to capture a focused set of Web documents for answering the queries in the near future — each representative query is used as a learning opportunity.

A topical crawler picks one representative query at a time and queries the search engine using it. The search engine responds with the top `N_HITS` URLs that satisfy the query and also all the URLs that have a link to the top URLs. This gives the crawler a *seed set* of URLs. The crawler then crawls up to `MAX_PAGES` starting from the seed set, using the representative query to guide

[1] Note that for the rest of the paper, we use the term "search engine" to refer to all of the search engine components other than the crawler.

itself. After MAX_PAGES are crawled on a query, the search engine indexes the new pages retrieved. The crawler queries the search engine again, with the same representative query, and the steps mentioned above repeat. This search engine-crawler loop may continue up to MAX_ITER iterations or until some convergence level (based on a threshold THETA) is achieved in the seed sets from two consecutive iterations. The new pages from the last iteration are added to a new collection. The iterations are repeated for all of the representative queries. At the end, the current index is deleted and the new collection is used to create a new index. The new index is used by the search engine to answer the queries until the entire process repeats.

```
new_collection = ();
foreach query (representative_queries) {
    repeat (MAX_ITER or until intersect(old_seed_set, seed_set) > THETA) {
        old_seed_set = seed_set;
        seed_set = search_engine(query, N_HITS);
        new_pages = crawler(query, seed_set, MAX_PAGES);
        index(new_pages);
        new_collection = add(new_collection, new_pages);
    }
}
clear_index();
index(new_collection);
```

Fig. 1. Pseudo-code of symbiotic process between search engine and crawler

The idea is that through this symbiotic process, the crawler can exploit an existing search engine index in order to obtain a focused collection for the search engine that may better satisfy future user queries.

A pseudo-code illustrating this symbiotic process is shown in Figure 1. The parameters MAX_PAGES, MAX_ITER, THETA and the number of representative queries are fixed based on resource availability. For example, if the system runs on a desktop used by a single user, then we may index a few thousand pages. On the other hand, if the system runs on a large server with many users, it may be desirable to index millions of pages. Once we fix the maximum number of pages to be indexed (MAX_IN_INDEX), MAX_PAGES can be derived as follows:

$$\texttt{MAX\_PAGES} = \frac{\texttt{MAX\_IN\_INDEX}}{|\texttt{representative\_queries}|}$$

where |representative_queries| is the number of representative queries. Parameters such as MAX_ITER and THETA are influenced by available disk space since they lead to a temporary increase in size of the index before the new collection is ready. The number of queries that are used as representative queries may be based on the amount of time taken by iterations for a single query. One may like to restrict the total time taken by all of the representative queries to, say, 12 hours (off peak hours of a day). At the end of the process shown in Figure 1, the new collection to be indexed must be of a size less than or equal to MAX_IN_INDEX.

Hence, we maintain an upper bound on the size of the index used by the search engine to answer user queries.

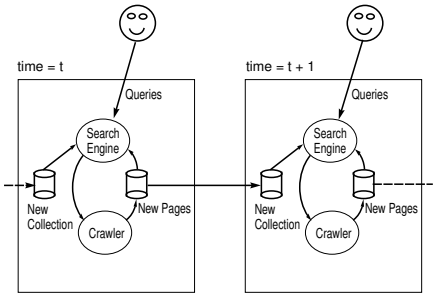

Fig. 2. Creation of a new index from an old one

Another way of looking at the process is through its temporal sequence. The pseudo-code shown in Figure 1 may be repeated at regular intervals to create new collections to cater to future queries. Figure 2 shows that the index at time t is used by the process described in Figure 1 to create a new index for time $t + 1$. The scale of time t could be a day, a week or a month based on the need for index freshness, the time available, and the maximum size of the collection (`MAX_IN_INDEX`).

3 Implementation

Currently, the search engine-crawler symbiosis is implemented using a search engine called Rosetta [5, 4] and a Naive Best-First crawler [14, 15]. Both the search engine and the crawler were not built specifically for this application. They have been used before for independent searching [5] and crawling [14, 15] tasks. We want to demonstrate the use of the symbiotic model by picking an off-the-shelf search engine and a generic topical crawler.

3.1 Rosetta

While the architecture we describe here is not wedded to any one type of indexing or retrieval system or for that matter any particular approach to crawling, the Rosetta search engine is particularly well-suited to this architecture. Rosetta is based on an indexing technique called Reference Directed Indexing (RDI) [5]. This technique is designed around the idea that for any topic, the community of Web users interested in that topic find and identify an ever-evolving body of useful information and do so in such a way that their findings can be leveraged to help the less well-informed find what they need. Rosetta uses hyperlinks and the contexts in which they are created as the basis for indexing and retrieving

documents. While many authors have introduced a variety of ways to use link information in search systems [6, 7], Rosetta's approach is novel in the way it uses the combined evidence of multiple references to a document to both determine the popularity of that document and to isolate the words that best identify why it is popular, and therefore, the queries for which it is most relevant. As an example we consider `www.mayura.com`, the Web site of Mayura software. Mayura is a drawing program for the Microsoft Windows platform that allows one to easily draw from scratch or import, modify and export images in a variety of formats. A comparative analysis of the text in the immediate vicinity of each link to `www.mayura.com` yields the following terms among those most frequently used in reference to this document: *Mayura*, *PageDraw*, *drawing program*, *Windows*, *export*, *PDF*, *EPS*. Many Web authors who have found this tool useful have referenced its homepage. In doing so they indicate not simply that Mayura Draw is useful in the abstract, but that it is useful for a very specific set of reasons, among them those identified by the terms listed above. Different referrers to a document tend to emphasize many of the same features of that document. As a result some of their terms naturally overlap. If many referrers use a particular term in reference to a document, it is quite likely that searchers will use that term in queries for the information the document contains. Rosetta indexes each document in its collection incrementally as it discovers references to these documents in pages gathered by the crawler. The RDI technique is something like a voting mechanism which uses terms found in the immediate vicinity of a hyperlink to a document. It treats each referring page as a voter permitted one vote for each index term used in reference to a document. Continuing with the Mayura example, the following lists the top 10 index terms extracted for `www.mayura.com` from the four referring pages. The number of votes indicates the number of pages that use each term in the immediate vicinity of a link to `www.mayura.com`.

TERM	VOTES
Mayura Draw	4
drawing program	4
format	4
illustrations	3
Windows	3
PDF	3
EPS	3
...	...

Rosetta uses the number of votes for a term as a measure of term frequency in a weighting metric similar to TFIDF [19]. In other work [5], we have found that the combined evidence from multiple contexts in which a document has been referred is an extremely accurate identifier of good index terms for a document, much better in fact, than measures of word use within the document itself [4]. In addition, inherent in the model is a measure of relative importance among all documents indexed by the same terms. The number of votes an index term receives indicates not only the relative value of that term in association with a particular document, but also the number of referrers that have chosen to direct their readers to that document versus another.

3.2 Naive Best-First Crawler

The Naive Best-First crawler uses the cosine similarity between a page and the query to score the URLs in the page. This similarity measure is based on simple term frequency, however common terms are conflated using a standard stemming algorithm [18]. The *crawl frontier* with unvisited URLs is kept as a priority queue based on the score. Every time the crawler needs to fetch a page, it picks the best one in the queue. In our previous evaluation studies, we have found the Naive Best-First crawler to be a strong competitor among other algorithms for short crawls of a few thousand pages, on general crawling tasks [14, 15].

The crawler can have a number of threads that share a single crawl frontier. Each thread picks the best URL to crawl from the frontier, fetches the corresponding page, scores the unvisited URLs within the page and adds the URLs to the frontier at appropriate positions. We allow the crawler to spawn as many as 75 threads and the maximum size of the frontier is set to 10,000. In order to avoid inundating any Web server with requests, the frontier enforces the constraint that every batch of D URLs fetched from it are from D different server host names ($D = 50$). Due to the multi-threaded nature of the crawler and the enforced ethics the crawler does not follow a strict best first order. Also, the multiple threads make the crawler behave like a Best-N-First crawler where N is related to the number of threads. Best-N-First crawler is a generalized version of Naive Best-First crawler that picks N best URLs to crawl at a time [17]. We have found Best-N-First (with $N = 256$) to perform well, especially when performance is measured through recall of relevant pages [15].

4 Evaluation

We want to investigate the use of search engine-crawler symbiosis in capturing community interests. The initial collection for the search engine may be generated by using the bookmark files of the users and doing a Breadth-First crawl until a certain number of pages (`MAX_IN_INDEX`) have been retrieved.

4.1 Community and Queries

We used a portion of the Open Directory Project[2] (ODP) as the basis for this simulation. In particular, we used the "Business/E-commerce" category to simulate a community. This category, being only two levels deep in the hierarchy of ODP topics, is sufficiently broad to enable the simulation of a community of people with many shared interests and many individual interests as well. From "Business/E-commerce" we used the root category and all sub-categories that have at least 10 external URLs. We assume that our fictitious community's interests lie within the categories selected. As seed URLs used to begin the simulation we selected as many as five URLs at random from each of the included categories.

[2] http://dmoz.org

In bootstrapping a search engine for a real community such URLs might be acquired from the Web browser bookmarks of members of that community or from a pool of URLs listed on a community resource page of some kind. Throughout the rest of the paper we will refer them as *bookmark* URLs.

Having selected the bookmark URLs, we then collected a pool of queries to represent the individual inquiries of our community during a period of one work week (five days). ODP editors include a brief summary of each URL listed within a given category. We used these summaries to derive phrases that simulate user queries. We first split the summaries into tokens using a set of stop words as delimiters, keeping only tokens with 2, 3, or 4 words for further processing. Next, we manually filtered the tokens to remove those that were incoherent groups of words representative of the type of errors made by such a shallow parsing technique. This process left us with nearly 1200 queries. The queries thus obtained form the *query pool* for the simulation to follow. Finally, we associated each query with the ODP category from which it was derived. This knowledge was not made available to the system, but was used to provide an understanding of the context of the query during the evaluation phase.

4.2 Simulation

We simulated 5 days of the search engine-crawler system at work. The initial collection was created for day 1 using a Breadth-First crawl that retrieved `MAX_IN_INDEX` $= 100,000$ pages from the Web starting from the bookmark URLs. To simulate a day of usage we randomly picked 100 queries from the query pool. At the end of each simulated day, we ran the symbiotic process described in Section 2 using all 100 queries to create a new index for the next day. For this simulation we set the `MAX_PAGES` (maximum pages per query iteration) parameter to 1000 pages, the maximum number of iterations is `MAX_ITER` $= 3$, and the number of top URLs used as seeds for each iteration, `N_HITS` $= 25$. Owing to the small `MAX_ITER`, we do not check for convergence in the current simulation.

In real-world use such a process may run during the off-peak hours (overnight) so that the new index is ready before the next morning. In fact, the implemented system using the Rosetta search engine and the Naive Best-First crawler takes around 11 hours to complete the process of shifting from the old index to a new one (Figure 2) using a 1Ghz pentium III IBM Thinkpad running the Windows 2000 operating system. Note that the current Rosetta implementation has foregone some parallelization and other optimizations for ease of implementation. Hence, we can speed up the process even further after some optimization.

4.3 Performance Metrics

The purpose of this technology is to incrementally refine a collection on a broad set of topics in order to focus the collection on the set of topics relevant to a particular community. To test the degree to which the system meets this objective we measured the relative performance as it progressed through the five days of simulation. We chose to treat the system as a black-box, and measure

the performance of the search engine in response to queries from each of the five days, because this is the way an end-user experiences the merits of the system.

To judge the search engine results for hundreds of queries over 5 days would be a time consuming task. Hence, we decided to use only a sample of the daily queries for our evaluation. Twelve test subjects evaluated five to ten queries randomly selected from the 100 (20 per day) sample queries used in the evaluation. The test subjects are graduate students in a class on information technology; they were awarded extra credit for participating in the study. We asked the subjects to determine the relevance of the top ten search results for each query based on the context in which the query originated. As we mentioned earlier, the context for each query was provided as the category in the ODP from which the query was drawn. The subjects were asked to browse the relevant category in order to acquire an understanding of the meaning of each query. This step is important because no real-world user submits a query to search engine unless a specific set of circumstances has motivated the need for information. Note that the search results for the same query on two different days in this evaluation were going against two different indexes. We maintained the indexes as they existed on each of the five days and submitted the queries sampled from that day against that day's index. To avoid manual bias in evaluation, we hid any information that would have indicated on which day the query originated.

Based on the binary relevance assessments obtained from the subjects, we computed the precision P among the top 10 hits returned by the search engine for each query. We average this performance measure over the sample queries for each day, and plot the mean performance over time.

In addition to the black-box approach mentioned above, we want to evaluate the quality of the collections created by the crawler. We would like to quantify a new collection's ability to satisfy the queries to follow. Note that precision P is not only affected by the collection quality but also by the quality of the indexing and the ranking mechanism. Here, we want to isolate the quality of the collection as it was created by the crawler. This also gives us an idea about the performance of the crawler. A way to measure a new collection's quality is by calculating the average cosine similarity between the pages in the collection and a representation (centroid) of the queries submitted on the corresponding day. To represent a day's queries, we simply concatenate them. Note that the collection is created before the queries are submitted to the system (see Figure 2). We measure the average cosine similarity between the collection and the queries as follows:

$$S(\mathcal{C}, q) = \frac{1}{\mid \mathcal{C} \mid} \sum_{p \in \mathcal{C}} \frac{\boldsymbol{v_q} \cdot \boldsymbol{v_p}}{\parallel \boldsymbol{v_q} \parallel \cdot \parallel \boldsymbol{v_p} \parallel}$$

where $\mathcal{C}$ is a collection of pages for a particular day, q is the concatenated query text for the same day, $\boldsymbol{v_q}$ and $\boldsymbol{v_p}$ are TFIDF [19] based vector representations of the concatenated queries and the page respectively, $\boldsymbol{v_q} \cdot \boldsymbol{v_p}$ is the inner product of the two vectors, and $\parallel \boldsymbol{v} \parallel$ is the Euclidean norm of the vector $\boldsymbol{v}$.

4.4 Results

We first present the results of the study in which we measure search engine performance across five days of simulation using the metric P defined in Section 4.3. Figure 3 (a) depicts the five values for P as judged by our test subjects, averaged over 20 queries per day. For the initial collection the system retrieved approximately 3.5 relevant documents among the top ten search results on average.

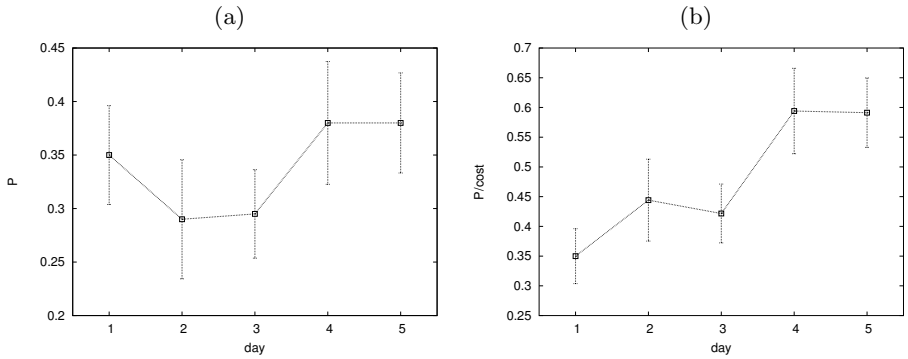

Fig. 3. (a) P over five days of simulation (b) The ratio of precision to cost of gathering and maintaining the collection for each day of the simulation. The error bars correspond to ± 1 standard error.

Performance dropped slightly on the second and third days of the simulation. As a reason for this we speculate that initially, the system may have overfit its collection to the set of queries from day one and possibly day two. We have seen this type of behavior before in other work on systems that learn the interests of their users [16]. Finally, this evaluation suggests that after the initial decline in performance, P improves to a level greater than that found with the initial collection. However, statistical significance of this preliminary result does not allow for a definitive confirmation; further experiments are required, both with more extensive evaluations (more queries) and over a longer period of time.

While not significant in terms of P, this result is perhaps stronger than it may first appear, because the system achieves this level of P with a collection that is over one-third smaller than the collection from the first day of the simulation. The size of the collection decreases from approximately 100,000 pages on the first day to approximately 65,000 pages on the fifth day. This is due to the crawler becoming more focused as a result of its interaction with the search engine. More specifically, as the crawler explores regions of the Web near the search results for a query, it finds many URLs encountered in crawls for other queries as well. Because we do not store the same URL in a collection more than once, any redundancy in the information gathered for a day results in a reduction in the size of the collection. While this may also negatively affect generalization performance during the initial phase of the symbiotic process (Figure 3 (a)), it affords

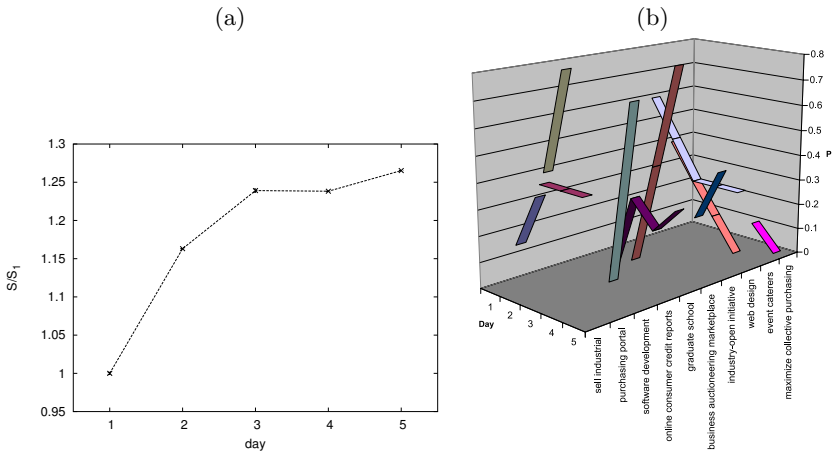

Fig. 4. (a) Relevance of the collection as a whole to the queries submitted on each of the five days (b) Performance on queries that occurred on more than one day.

significant efficiency gains in terms of saved disk space, crawling and indexing time, and bandwidth. To quantify such gain, we look at the ratio of average precision over the relative size of the collection or *cost*. The cost of a collection is equal to the size of the collection divided by `MAX_IN_INDEX` (100,000 pages). The plot in Figure 3 (b) indicates that from a perfomance/cost perspective, the symbiotic system can lead to substantial benefits.

Next, we measured the cosine similarity, $S(\mathcal{C}, q)$, between the collection for a given day and the queries for that day. Figure 4 (a) plots the ratio of $S(\mathcal{C}, q)$ to $S_1(\mathcal{C}, q)$ for day one. Over time, the crawler with the help of the search engine, is able to fetch pages that are lexically more similar to the queries of the day. The error bars on the plot are extremely small and hence not clearly visible. Hence, we find significant improvement in collection quality over the simulation period.

On further analysis, we found that 10 queries appeared more than once in the manually evaluated set of queries. We plotted the metric P for each of the repeating queries against the days on which they appeared (Figure 4 (b)). We noticed that for most of the queries the value of P improved with time. In particular, there are queries for which no result was found the first time they appeared, but on subsequent occurrence the system found several relevant results. On average the system found 4.5 more relevant results for repeating queries between the first and the last time they were submitted to the system.

5 Related Work

The large size and the dynamic nature of the Web has prompted many different efforts to build focused search engines [10, 12]. While we too attempt to build a focused search engine, our approach is adaptive to the changing interests of the

user(s). Furthermore, it tightly couples the search engine and the crawler, and is general enough that it can be applied to any topic.

Referential text has been used to find Web sites [9], categorize pages [2] and crawl pages [11]. Despite the active use of referential text for variety of information retrieval tasks, no one has yet demonstrated the effectiveness of this technique for general-purpose search. There is a large and growing body of work on topical or focused crawlers (e.g.,[11, 8, 13]). Such crawlers use variety of lexical and/or link-based cues within Web pages to guide their path.

Our work also relates to collaborative filtering [3] since the queries submitted by the users, help in preparing the collection for similar queries by other users in the future.

6 Conclusions

The purpose of the symbiotic system we describe here is to incrementally refine a broad collection in order to bring into focus the set of topics relevant to a particular community. While more experimentation is needed to make any strong claims about the benefits to end users, the work presented here demonstrates that in a short amount of time, the type of symbiotic system we have developed can eliminate much of the irrelevant information from an initial collection and thereby achieve the desired focus. It is important to note that the system, though it learns from the behavior of its users, does so implicitly, requiring no effort beyond the type of simple searches they are already doing.

One obvious extension to the work is to integrate the crawler and the engine further. For example, the crawler's best first strategy could be based on TFIDF similarity rather than just TF. A generic crawler does not have the luxury of an IDF index because the collection is not yet available. In our model however the collection indexed by the engine in the previous iteration can be used by the crawler in the current iteration to generate IDF weights that can improve its link scores. In addition, the crawler could tap into the more global information available to the search engine — such as good hubs and multiple contexts. On a different note, the proposed symbiotic model can be distributed on a larger Peer-to-Peer (P2P) search system which can bring in added opportunities for identifying and connecting communities of users.

Acknowledgments

We acknowledge the contributions of Padmini Srinivasan, Kristian Hammond and Rik Belew in related projects. Thanks go to the students who volunteered to help in the evaluation. This work is funded in part by NSF CAREER Grant No. IIS-0133124 to FM.

References

1. A. Arasu, J. Cho, H. Garcia-Molina, A. Paepcke, and S. Raghavan. Searching the Web. *ACM Transactions on Internet Technology*, 1(1), 2001.

2. G. Attardi, A. Gullí, and F. Sebastiani. Automatic Web page categorization by link and context analysis. In *Proceedings of THAI-99, 1st European Symposium on Telematics, Hypermedia and Artificial Intelligence*, 1999.
3. M Balabanović and Y Shoham. Content-based, collaborative recommendation. *Communications of the ACM*, 40(3):67–72, 1997.
4. S. Bradshaw and K. Hammond. Automatically indexing documents: Content vs. reference. In *Sixth International Conference on Intelligent User Interfaces*, San Francisco, CA, January 14–17 2002.
5. Shannon Bradshaw. *Reference Directed Indexing: Indexing Scientific Literature in the Context of its Use*. PhD thesis, Northwestern University, 2002.
6. S. Brin and L. Page. The anatomy of a large-scale hypertextual Web search engine. *Computer Networks*, 30(1–7):107–117, 1998.
7. S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks*, 30(1–7):65–74, 1998.
8. S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: A new approach to topic-specific Web resource discovery. *Computer Networks*, 31(11–16):1623–1640, 1999.
9. N. Craswell, D. Hawking, and S. Robertson. Effective site finding using link anchor information. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.
10. C.L. Giles, K.D. Bollacker, and S. Lawrence. Citeseer: an automatic citation indexing system. In *Proceedings of the third ACM conference on Digital libraries*, 1998.
11. M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalhaim, and S. Ur. The shark-search algorithm — An application: Tailored Web site mapping. In *WWW7*, 1998.
12. A.K. McCallum, K. Nigam, J. Rennie, and K. Seymore. Automating the construction of internet portals with machine learning. *Information Retrieval*, 3(2):127–163, 2000.
13. F. Menczer and R. K. Belew. Adaptive retrieval agents: Internalizing local context and scaling up to the Web. *Machine Learning*, 39(2–3):203–242, 2000.
14. F. Menczer, G. Pant, M. Ruiz, and P. Srinivasan. Evaluating topic-driven Web crawlers. In *Proc. 24th Annual Intl. ACM SIGIR Conf. on Research and Development in Information Retrieval*, 2001.
15. F. Menczer, G. Pant, and P. Srinivasan. Topical web crawlers: Evaluating adaptive algorithms. *To appear in ACM Trans. on Internet Technologies*, 2003. http://dollar.biz.uiowa.edu/~fil/Papers/TOIT.pdf.
16. F. Menczer, W.N. Street, N. Vishwakarma, A. Monge, and M. Jakobsson. IntelliShopper: A proactive, personal, private shopping assistant. In *Proc. 1st ACM Int. Joint Conf. on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2002.
17. G. Pant, P. Srinivasan, and F. Menczer. Exploration versus exploitation in topic driven crawlers. In *WWW02 Workshop on Web Dynamics*, 2002.
18. M. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.
19. G. Salton and M.J. McGill. *Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.