Exploiting Interclass Rules for Focused Crawling

Ismail Sengör Altingövde and Özgür Ulusoy, *Bilkent University*

A focused crawler gathers relevant Web pages on a particular topic. This rule-based Web-crawling approach uses linkage statistics among topics to improve a baseline focused crawler's harvest rate and coverage.

Crawling the Web quickly and entirely is an expensive, unrealistic goal because of the required hardware and network resources. A focused crawler is an agent that targets a particular topic and visits and gathers only a relevant, narrow Web segment while trying not to waste resources on irrelevant material. An underlying paradigm for a focused crawler is implementing a best-first search strategy rather than a breadth-first search, which general-purpose crawlers typically employ.

Some focused crawlers, such as Ahoy homepage finder (www.cs.washington.edu/research/projects/WebWare1/www/ahoy), search for pages of a particular document type; others, such as a crawler for quantum physics, search for pages on a specific topic. Without losing generality, this article emphasizes the latter. Because a focused crawler searches only for pages relevant to its targeted topic, two fundamental questions are how to decide whether a downloaded page is on topic and how to choose the next page to visit.[1] Researchers have proposed several ideas to answer these two questions, which the "Related Work" sidebar discusses.

In our work, we started with a focused-crawling approach designed by Soumen Chakrabarti, Martin van den Berg, and Byron Dom, and we implemented the underlying philosophy of their approach to derive our baseline crawler.[2] This crawler employs a canonical topic taxonomy to train a naïve-Bayesian classifier, which then helps determine the relevancy of crawled pages. The crawler also relies on the assumption of topical locality to decide which URLs to visit next. Building on this crawler, we developed a rule-based crawler, which uses simple rules derived from interclass (topic) linkage patterns to decide its next move. This rule-based crawler also enhances the baseline crawler by supporting tunneling. Initial performance results show that the rule-based crawler improves the baseline focused crawler's harvest rate and coverage.

A typical Web crawler

With the Web's emergence in the early 1990s, crawlers (also known as robots, spiders, or bots) appeared on the market with the purpose of fetching all pages on the Web to allow other useful tasks, such as indexing. Typically, a crawler begins with a set of given Web pages, called seeds, and follows all the hyperlinks it encounters along the way, to eventually traverse the entire Web.[3] General-purpose crawlers insert the URLs into a queue and visit them in a breadth-first manner. Of course, the expectation of fetching all pages is not realistic, given the Web's growth and refresh rates. A typical crawler runs endlessly in cycles to revisit the modified pages and access unseen content.

Figure 1 illustrates the simplified crawler architecture we implemented based on the architecture outlined in Chakrabarti's book.[3] This figure also reveals various subtleties to consider in designing a crawler. These include caching and prefetching of DNS (Domain Name System) resolutions, multithreading, link extraction and normalization, conforming to robot exclusion protocols, eliminating seen URLs and content, and handling load balancing among servers (the politeness policy). We ignored some other issues, such as refresh rates, performance monitoring, and how to handle hidden Web pages, because they're not essential for our experimental setup.

Our crawler operates like a typical crawler. The URL queue is initially filled with several seed URLs. Each DNS thread removes a URL from the queue and tries to match the host name with an Internet Protocol address. At first, a DNS thread consults the

1541-1672/04/$20.00 © 2004 IEEE
Published by the IEEE Computer Society

Related Work

A focused crawler searches for the most relevant pages on a particular topic. Two key questions are how to decide whether a downloaded page is on topic and how to choose the next page to visit. Researchers have proposed several ideas to answer these two questions.

Early algorithms

The FishSearch system is one of the earliest approaches to ordering the crawl frontier (for example, through a priority queue of URLs).[1] The system is query driven. Starting from a set of seed pages, it considers only those pages that have content matching a given query (expressed as a keyword query or a regular expression) and their neighborhoods (pages pointed to by these matched pages).

The SharkSearch system is an improvement over FishSearch.[2] It uses a weighting method of term frequency (TF) and inverse document frequency (IDF) along with the cosine measure to determine page relevance. SharkSearch also somewhat smooths the depth cutoff method that its predecessor used. Meanwhile, Junghoo Cho, Hector Garcia-Molina, and Lawrence Page have proposed reordering the crawl frontier according to page importance,[3] which they can compute using various heuristics—page rank, number of pages pointing to a page (in-links), and so on. These three algorithms don't employ a classifier, but rather rely on techniques based on information retrieval (IR) to determine relevance.

Soft-focus crawling

Soumen Chakrabarti, Martin Van den Berg, and Byron Dom were the first to propose a soft-focus crawler,[4] which obtains a given page's relevance score (relevance to the target topic) from a classifier and assigns this score to every URL extracted from that page. We refer to this soft-focus crawler as the baseline focused crawler.

Focused crawling with tunneling

An essential weakness of the baseline focused crawler is its inability to model tunneling; that is, it can't tunnel toward the on-topic pages by following a path of off-topic pages.[5] Two remarkable projects, the context-graph-based crawler and Cora's focused crawler, achieve tunneling.

The context-graph-based crawler uses a best-search heuristic, but the classifiers learn the layers representing a set of pages that are at some distance to the pages in the target class (layer 0).[6] The crawler simply uses these classifier results and inserts URLs extracted from a layer-*i* page to the layer-*i* queue; that is, it keeps a dedicated queue for each layer. While deciding the next page to visit, the crawler prefers the pages nearest to the target class—that is, the URLs popped from the queue that correspond to the first nonempty layer with the smallest layer label. This approach clearly solves the problem of tunneling, but unfortunately requires constructing the context graph, which in turn requires finding pages with links to a particular page (back links). Our rule-based crawler, on the other hand, uses forward links while generating the rules and transitively combines these rules to effectively imitate tunneling behavior.

Cora is a domain-specific search engine on computer science research papers that relies heavily on machine-learning techniques.[7] In particular, Andrew McCallum and his colleagues have used reinforcement learning in Cora's focused crawler. Cora's crawler basically searches for the expected future reward by pursuing a path starting from a particular URL. Training classifiers involves learning the paths that could lead to on-topic pages in some number of steps. In contrast, our rule-based crawler doesn't need to see a path of links during training, but constructs the paths using the transitive combination and chaining of simple rules (of length 1).

Other approaches

The Web Topic Management System focused crawler fetches only pages that are close (parent, child, and sibling) to on-topic pages.[8] WTMS determines a page's relevancy using IR-based methods. Charu Aggarwal, Fatima Al-Garawi, and Philip Yu attempt to learn the Web's linkage structure to determine a page's likelihood of pointing to an on-topic page.[9] However, they don't consider interclass relationships in the way we do. Bingo! is a focused-crawling system for overcoming the limitations of initial training by periodically retraining the classifier with high-quality pages.[10] Recently, Filippo Menczer, Gautam Pant, and Padmini Srinivasan presented an evaluation framework for focused crawlers and introduced an evolutionary crawler.[11] A. Abdollahzadeh Barfourosh and his colleagues' survey provides a more comprehensive discussion.[5]

References

1. P. De Bra et al., "Information Retrieval in Distributed Hypertexts," *Proc. 4th Int'l Conf. Intelligent Multimedia Information Retrieval Systems and Management* (RIAO 94), Center of High Int'l Studies of Documentary Information Retrieval (CID), 1994, pp. 481–491.
2. M. Hersovici et al., "The SharkSearch Algorithm—An Application: Tailored Web Site Mapping," *Computer Networks and ISDN Systems*, vol. 30, nos. 1–7, pp. 317–326.
3. J. Cho, H. Garcia-Molina, and L. Page, "Efficient Crawling through URL Ordering," *Computer Networks and ISDN Systems*, vol. 30, nos. 1–7, pp. 161–172.
4. S. Chakrabarti, M.H. Van den Berg, and B.E. Dom, "Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery," *Computer Networks*, vol. 31, nos. 11–16, pp. 1623–1640.
5. A.A. Barfourosh et al., *Information Retrieval on the World Wide Web and Active Logic: A Survey and Problem Definition*, tech. report CS-TR-4291, Computer Science Dept., Univ. of Maryland, 2002.
6. M. Diligenti et al., "Focused Crawling Using Context Graphs," *Proc. 26th Int'l Conf. Very Large Data Bases* (VLDB 2000), Morgan Kaufmann, 2000, pp. 527–534.
7. A. McCallum et al., "Building Domain-Specific Search Engines with Machine Learning Techniques," *Proc. AAAI Spring Symp. Intelligent Agents in Cyberspace*, AAAI Press, 1999, pp. 28–39.
8. S. Mukherjea, "WTMS: A System for Collecting and Analyzing Topic-Specific Web Information," *Computer Networks*, vol. 33, nos. 1–6, pp. 457–471.
9. C.C. Aggarwal, F. Al-Garawi, and P. Yu, "Intelligent Crawling on the World Wide Web with Arbitrary Predicates," *Proc. 10th Int'l World Wide Web Conf.* (WWW 01), ACM Press, 2001, pp. 96–105.
10. S. Sizov, J. Graupmann, and M. Theobald, "From Focused Crawling to Expert Information: An Application Framework for Web Exploration and Portal Generation," *Proc. 29th Int'l Conf. Very Large Data Bases* (VLDB 2003), Morgan Kaufmann, 2003, pp. 1105–1108.
11. F. Menczer, G. Pant, and P. Srinivasan, "Topical Web Crawlers: Evaluating Adaptive Algorithms," to be published in *ACM Trans. Internet Technology*, vol. 4, no. 4.

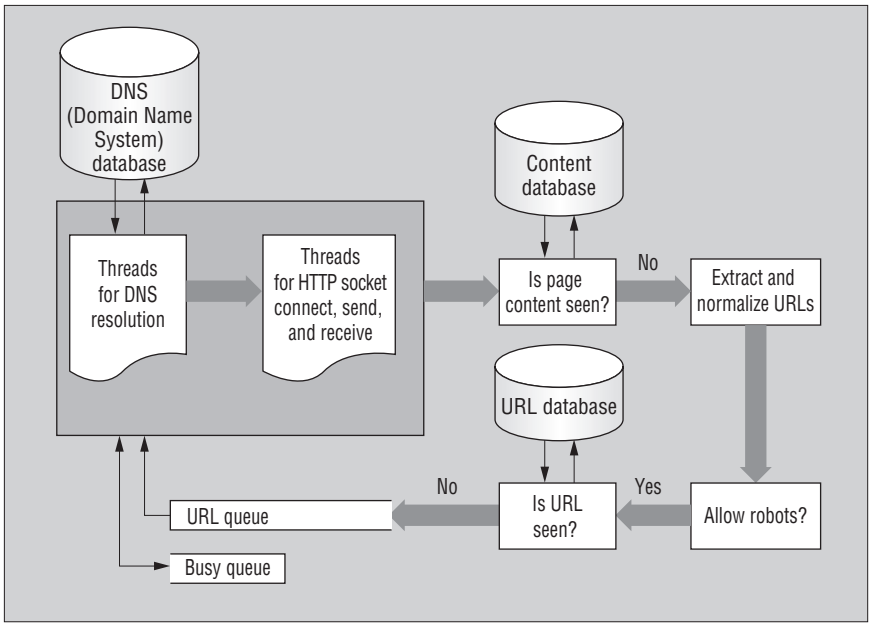


Figure 1. Our implementation of a typical crawler.

DNS database to see whether the host name has been resolved; if so, the thread retrieves the IP from the database.

Otherwise, DNS threads learn the IP from a DNS server. Next, a read thread receives the resolved IP address, tries to open an HTTP socket connection, and asks for the Web page. After downloading the page, the crawler checks the page content to avoid duplicates. Next, it extracts and normalizes the URLs in the fetched page, verifies whether robots can crawl those URLs, and checks whether the crawler has previously visited those extracted URLs (so that the crawler isn't trapped in a cycle). The crawler hashes the URLs with the MD5 message-digest algorithm (www.rsasecurity.com/rsalabs/node.asp?id=2253) and store the hash values in the URL database for this check. Finally, if this is the first time the crawler has encountered the URL, it inserts this URL into the URL queue—a first-in, first-out (FIFO) data structure in a general-purpose crawler. Of course, we don't want to annoy servers with thousands of simultaneous requests, so the first time the crawler accesses a server, it marks it as busy and stores it with a time stamp. The crawler accesses the other URLs from this server only after this time stamp is old enough (typically after 10 seconds, to be on the safe side). During this time, if a thread gets a URL referring to such a busy server, the crawler places this URL in the busy queue. The threads alternate between accessing the URL queue and the busy queue, to prevent starvation in one of the queues. Our current crawler doesn't use a dedicated queue per server, which could be a more sophisticated way to enforce a politeness policy. (For more details, see Chakrabarti's excellent book.[3])

Baseline focused crawler

Atop the basic crawler just described, we implemented Chakrabarti, Van den Berg, and Dom's focused crawler as our baseline crawler.[2] We use this crawler to present our rule-based crawling strategy and to evaluate its performance. The baseline crawler uses a best-first search heuristic during the crawling process. In particular, it uses both page content and link structure information while determining the promising URLs to visit.

The system includes a canonical topic (class) taxonomy, a hierarchy of topics along with a set of example documents. We can easily obtain such a taxonomy from the Open Directory Project (www.dmoz.org) or Yahoo (www.yahoo.com). Users determine the focus topics by browsing this taxonomy and marking one or more topics as the targets. Chakrabarti, Van den Berg, and Dom assume that the taxonomy induces a hierarchical partitioning of Web pages (each page belongs to only one topic), and we rely on this assumed partitioning for this project.

An essential component of the focused crawler is a document classifier. Chakrabarti, Van den Berg, and Dom use an extended naïve-Bayesian classifier called Rainbow to determine the crawled document's relevance to the target topic.[4] During the training phase, we trained this classifier with the topic taxonomy's example pages so that it learned to recognize the taxonomy.

Once the classifier constructs its internal model, it can determine a crawled page's topic—for example, as the topic in the taxonomy with the highest probability score. Given a page, the classifier returns a sorted list of all class names and the page's relevance score to each class. Thus, the classifier is responsible for determining the on-topic Web pages. Additionally, it determines which URLs to follow next, assuming a page's relevance can be an indicator of its neighbor's relevance—that is, the *radius-1 hypothesis*. (The radius-1 hypothesis contends that if page u is an on-topic example, and u links to v, then the probability that v is on topic is higher than the probability that a randomly chosen Web page is on topic.[3])

Clearly, this hypothesis is the basis of the baseline focused crawler and can guide crawling in differing strictness levels.[2] In a hard-focus crawling approach, if the classifier identifies a downloaded page as off topic, the crawler doesn't visit the URLs found at that page; in other words, it prunes the crawl at this page. For example, if the highest-scoring class returned by the classifier for a particular page doesn't fall within the target topic, or if the score is less than a threshold (say, 0.5), the crawler concludes that this page is off topic and stops following its links. This approach is rather restrictive compared to its alternative, soft-focus crawling. In that approach, the crawler obtains from the classifier the given page's *relevance score* (a score on the page's relevance to the target topic) and assigns this score to every URL extracted from this particular page. Then, the crawler adds these URLs to the priority queue on the basis of these relevance scores. Clearly, the soft-focus crawler doesn't eliminate any pages ahead of time. Another major component of the baseline crawler is the distiller, which exploits the link structure to further refine the URL frontier's ordering. In our research, we didn't include the distiller component in the baseline crawler implementation because we expect its effect to be the same for the baseline crawler and our rule-based crawler.

Thus, our baseline focused crawler includes a naïve-Bayesian classifier and decides on the next URL to fetch according to

the soft-focus crawling strategy. This means we simply add a step to classify the page and return its relevance to the target topic immediately before the URL extraction stage shown in Figure 1. We also replace the FIFO queues with priority queues.

Rule-based focused crawler

One problem with the baseline focused crawler is its inability to support tunneling. More specifically, the classifier doesn't learn that a path of off-topic pages can eventually lead to high-quality, on-topic pages. For example, if you're looking for neural network articles, you might find them by following links from a university's homepage to the computer science department's homepage and then to the researchers' pages, which might point to the actual articles (as Michelangelo Diligenti and his colleagues also discuss[5]). Our baseline focused crawler could possibly attach low relevance scores to university homepages (recall that naïve-Bayesian classifiers can be biased to return either too high or too low relevance scores for a particular class[6]) and thus might miss future on-topic pages. The chance of learning or exploring such paths would further decrease as the lengths of the paths to be traversed increased.

Chakrabarti, Van den Berg, and Dom report that they've identified situations in which pages of a certain class refer not only to other pages of its own class (as envisioned by the radius-1 hypothesis) but also to pages from various other classes. For example, they observed that pages for the topic "bicycle" also refer to "red-cross" and "first-aid" pages; and pages on "HIV/AIDS" usually refer to "hospital" pages more frequently than to other "HIV/AIDS" pages.[2]

To remedy these problems, we extract rules that statistically capture linkage relationships among the classes (topics) and guide our focused crawler by using these rules. We based our heuristic on determining relationships such as "pages in class A refer to pages in class B with probability *p*." During focused crawling, we ask the classifier to classify a particular page that has already been crawled. According to that page's class, we compute a score indicating the total probability of reaching the target topic from this particular page. Then, the crawler inserts the URLs extracted from this page into the priority queue with the computed score.

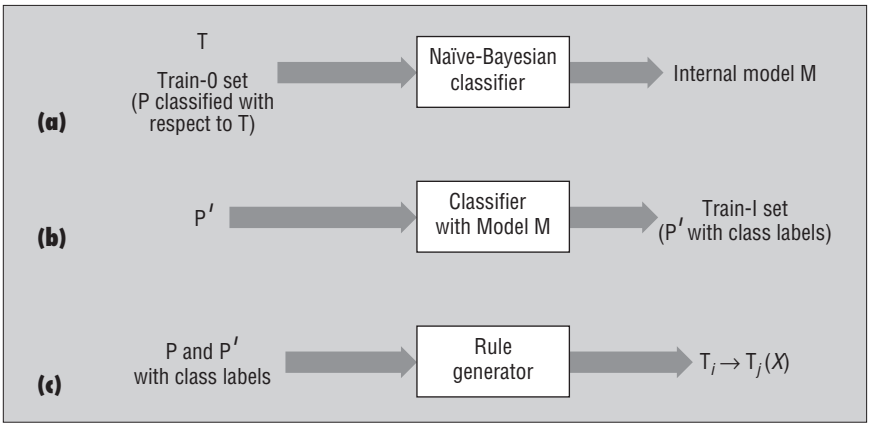


Figure 2. Stages of a rule generation process: (a) train the crawler's classifier with topic taxonomy T and the train-0 set to form internal model M, which learns T; (b) use page set P′, pointed to by P, to form the train-1 set; (c) generate rules of the form $T_i \rightarrow T_j (X)$, where *X* is the probability score.

To achieve our goal, we first trained the crawler's classifier component with a class taxonomy and a set of example documents for each class (which you can find in a Web directory such as DMOZ), and we call this the train-0 set (see Figure 2a). Next, for each class in the train-0 set, we gather all Web pages that the example pages in the corresponding class point to (through hyperlinks). Once again, we have a collection of class names (the train-1 set) and a set of fetched pages for each class, but this time the class name is the class of parent pages in the train-0 set that point to these fetched documents. We give the train-1 set to the classifier to find each page's actual class labels (see Figure 2b). Now, we know the class distribution of pages to which the documents in each train-0 set class point. So, for each class in the train-0 set, we count the number of referred classes in the corresponding train-1 page set and generate rules of the form $T_i \rightarrow T_j (X)$, meaning a page of class T_i can point to a page of class T_j with probability X (see Figure 2c). Probability score X is the ratio of train-1 pages in class T_j to all train-1 pages that the T_i pages in the train-0 set refer to. Subsequently, a focused crawler seeking Web pages of class T_j attaches priority score X to the pages of class T_i that it encounters during the crawling phase.

To demonstrate our approach, we present an example. Our taxonomy includes four classes:

- Department homepages (DH)
- Course homepages (CH)
- Personal homepages (PH)
- Sports pages (SP)

We can construct the train-0 set as just described (for example, from an existing Web directory). Next, for each class, we retrieve the pages that this class's example pages refer to. Assume that we fetch 10 such pages for each class in the train-0 set and that the class distribution among these newly fetched pages (that is, the train-1 set) is as listed in Table 1. Then, we can obtain the rules of Table 2 in a straightforward manner.

Now, we compare the behavior of the baseline and rule-based crawlers to see how the

Table 1. Class distribution of pages fetched into the train-1 set for each class in the train-0 set.

Department homepages (DH)	Course homepages (CH)	Personal homepages (PH)	Sports pages (SP)
8 URLs for CH	2 URLs for DH	3 URLs for DH	10 URLs for SP
1 URL for PH	4 URLs for CH	4 URLs for CH	None
1 URL for SP	4 URLs for PH	3 URLs for PH	None

Table 2. Interclass rules for the distribution in Table 1. (The number following each rule is the probability score.)

Department homepages (DH)	Course homepages (CH)	Personal homepages (PH)	Sports pages (SP)
DH → CH (0.8)	CH → DH (0.2)	PH → DH (0.3)	SP → SP (1.0)
DH → PH (0.1)	CH → CH (0.4)	PH → CH (0.4)	NA
DH → SP (0.1)	CH → PH (0.4)	PH → PH (0.3)	NA

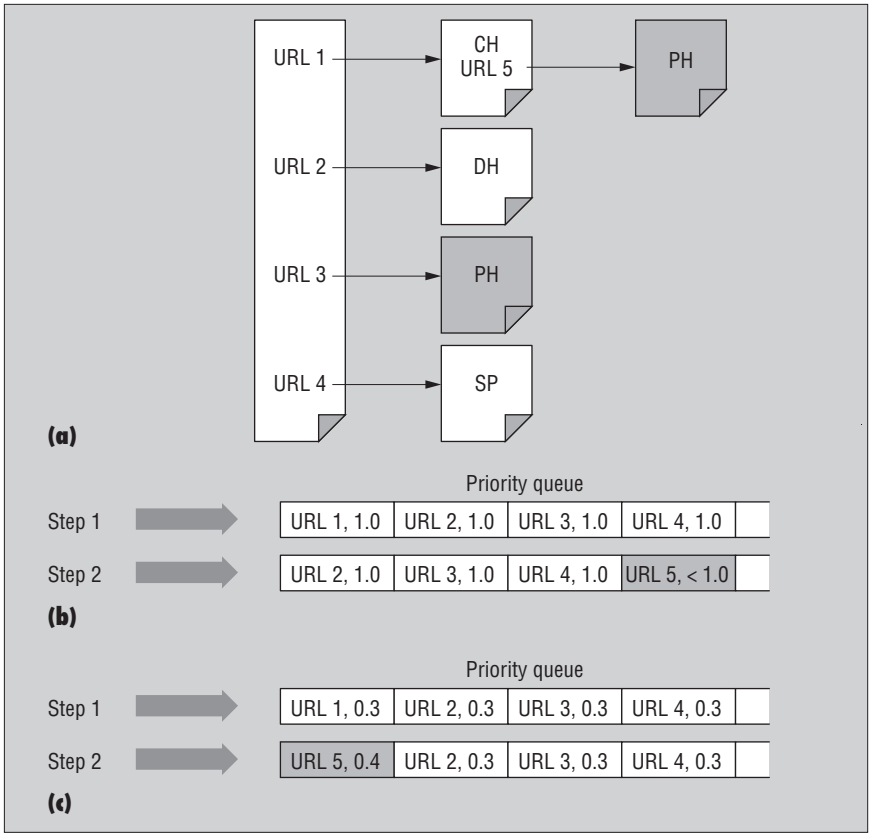


Figure 3. An example scenario: (a) seed page S of class PH; (b) steps of the baseline crawler; (c) steps of the rule-based crawler. (Shading in (a) denotes pages from the target class; shading in (b) and (c) highlights where the two crawlers differ in Step 2.)

rule-based crawler overcomes the aforementioned problems of the baseline crawler. Assume we have the situation given in Figure 3. In this scenario, the seed page is of class PH, which is also the target class; that is, our crawler is looking for personal homepages. The seed page has four hyperlinks, such that links URL 1 through URL 4 refer to pages of classes CH, DH, PH, and SP, respectively. Furthermore, the CH page itself includes another hyperlink (URL 5) to a PH page.

As Figure 3b shows, the baseline crawler begins by fetching the seed page, extracting all four hyperlinks, and inserting them into the priority queue with the seed page's relevance score, which is 1.0 by definition (Step 1). Next, the crawler fetches URL 1 from the queue, downloads the corresponding page, and sends it to the classifier. With the soft-focus strategy, the crawler uses the page's relevance score to the target topic according to the classifier. Clearly, the CH page's score for target class PH is less than 1, so the crawler adds URL 5, extracted from the CH page, to the end of the priority queue (Step 2). Thus, at best, after downloading all three pages with URLs 2 through 4, the crawler downloads the page pointed to by URL 5, which is indeed an on-topic page. If there are other intervening links or if the classifier score has been considerably low for URL 5, it might be buried so deep in the priority queue that it will never be recalled again.

In contrast, as Figure 3c shows, the rule-based crawler discovers that the seed page of class PH can point to another PH page with probability 0.3 (due to the rules in Table 2), so it inserts all four URLs to the priority queue with score 0.3 (Step 1). Next, the crawler downloads the page pointed to by URL 1 and discovers that it's a CH page. By firing rule CH → PH (0.4), it inserts URL 5 to the priority queue, which is now at the head of the queue and will be downloaded next (Step 2), leading to an immediate reward—an on-topic page. Figure 3 captures the overall scenario.

The rule-based crawler can also support tunneling for longer paths using a simple application of transitivity among the rules. For example, while evaluating URL 2 in the previous scenario, the crawler learns that the crawled page is of class DH. Then, the direct rule to use is DH → PH (0.1). Besides, the crawler can easily deduce that rules DH → CH (0.8) and CH → PH (0.4) exist and then combine them to obtain path DH → CH → PH with a score of $0.8 \times 0.4 = 0.32$ (assuming the independence of probabilities). In effect, the rule-based crawler becomes aware of path DH → CH → PH, even though the crawler is trained only with paths of length 1. Thus, the crawler assigns a score of, say, the sum of the individual rule scores (0.42, for this example) to the URLs extracted from DH and inserts these URLs into the priority queue accordingly.

Our rule-based scoring mechanism is independent of a page's similarity to the target page, but rather relies on the probability that a given page's class refers to the target class. In contrast, the baseline classifier would most probably score the similarity of a DH page to target topic PH significantly lower than 0.42 and might never reach a rewarding on-topic page.

Computing the rule-based scores

There can be cases with no rules (for example, the train-0 and train-1 sets might not cover all possible situations). For such cases, we decided to combine the soft-focus and rule-based crawling approaches. Figure 4 shows our final scoring function.

Because the rules can chain in a recursive manner, the MAX DEPTH value defines the maximum depth of allowed chaining. Typically,

we allow rules to have a depth of at most 2 or 3. Also, when there's more than one path from an initial class to the target class, the crawler must merge the paths accordingly. Two potential merging functions are *maximum* and *sum*; we prefer to employ *sum* for the experiments described in this article.

Finally, the crawler can compute all the rules for a particular set of target topics from the rule database before beginning the actual crawling. It's possible to represent the rule database as a graph, as Figure 5a shows. Then, for a given a target class, T, the crawler can efficiently find all cycle-free paths that lead to this class (except the loop T → T)—by modifying the breadth-first-search algorithm, for example.[7] Figure 5b demonstrates the rule-based score computation process for a page of class DH, where the target class is PH, and MAX DEPTH is 2.

```
if (∃ rules such that paths of C → … → T in at most MAX DEPTH steps exist)
    score(P, u) = rule-focus score (the sum of the product of individual rule probabilities in each such path)
else
    score(P, u) = soft-focus score (the score of P for the class T computed by the classifier)
```

Figure 4. Final scoring function for the rule-based crawler. In this formulation, T is the target class, P is the page at hand, C is the highest scoring class for P, and u is a URL extracted from P.

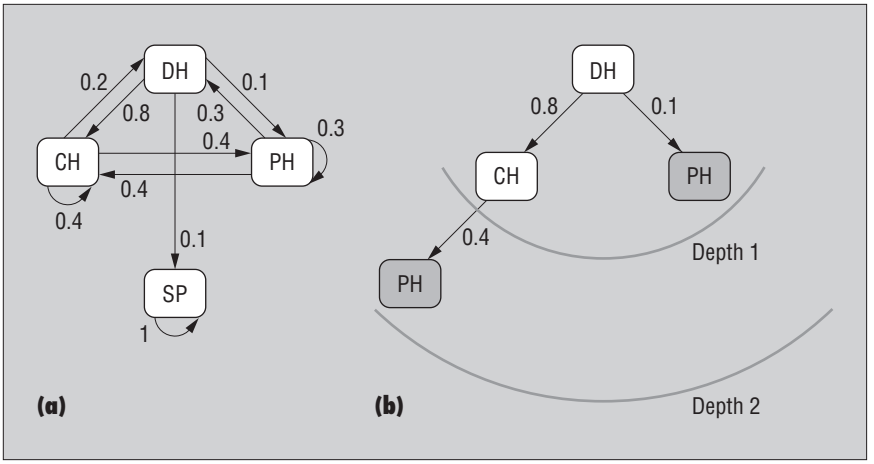


Figure 5. Rule-based score computation. (a) graph representation of the rule database, and (b) computation of rule paths and scores—for example, a DH page has the score (0.8 × 0.4) + 0.1 = 0.42. (Once again, shading denotes pages from the target class.)

Experimental setup

To evaluate our rule-based crawling strategy, we created the experimental setup described in earlier works.[2,6] We created the train-0 set using the DMOZ taxonomy and data. In this taxonomy, we moved the URLs found at a leaf node to its parent node only if the number of URLs was less than a predefined threshold (set to 150 for these experiments), and then we removed the leaf. Next, we used only the leaves for the canonical class taxonomy; we discarded the tree's upper levels. This process generated 1,282 classes with approximately 150 URLs for each class. When we attempted to download all these URLs, we successfully fetched 119,000 pages (including 675,000 words), which constituted our train-0 set.

Next, due to time and resource limitations, we downloaded a limited number of referred URLs (from the pages in 266 semantically interrelated classes on science, computers, and education) in the train-0 set. This amounted to almost 40,000 pages, and it forms our train-1 set. Because we also chose our target topics from these 266 classes for our evaluations, downloading the train-1 set sufficed to capture most of the important rules for these classes. Even if we missed a rule, its score would be negligibly low (for example, a rule from "Top.Arts.Music.Styles.Opera" to "Top.Computer.OpenSource" might not have a high score—if it even exists).

We employed the Bow library and the Rainbow text classifier as the default naïve-Bayesian classifier.[4] We trained the classifier and created the model statistics with the train-0 data set in almost 15 minutes. Next, we classified the train-1 data set using the constructed model (which took about half an hour). In the end, using the train sets, we obtained 4,992 rules.

We downloaded all the data sets using a modest crawler that we wrote in C. (Figure 1 shows the crawler's architecture.) We used Sleepycat Software's (www.sleepycat.com) Berkeley DB, an open-source developer database, to implement the underlying databases for storing DNS resolutions, URLs, seen-page contents, hosts, extracted rules, and the URL priority queues. During the data set downloading process, the crawler executed with 10 DNS and 50 read threads.

Performance of our rule-based crawler

Here, we provide initial performance results for three focused crawling tasks using the baseline crawler with the soft-focus crawling strategy and our rule-based crawling strategy. The target topics were Top.Science.Physics.QuantumMechanics, Top.Computer.History, and Top.Computer.OpenSource, for which there were 41, 148, and 212 rules, respectively, in our rule database. For each topic, we constructed two disjoint seed sets of 10 URLs each, from the pages listed at corresponding entries of the DMOZ and Yahoo directories.

We can evaluate a focused crawler's performance with the harvest ratio, a simple measure of the average relevance of all crawler-acquired pages to the target topic.[2,3] Clearly, the best way to solicit such relevance scores is to ask human experts. This is impractical for crawls ranging from thousands to millions of pages. So, following the approach of earlier works,[2] we again use our classifier to determine the pages' relevance scores. We compute the harvest ratio as follows:

$$HR = \frac{\sum_{i=1}^{N} Relevance(\mathrm{URL}_i, \mathrm{T})}{N}$$

where $Relevance(\mathrm{URL}_i, \mathrm{T})$ is the relevance of the page (with URL_i) to target topic T as returned by the classifier, and N is the total number of pages crawled.

Table 3 lists the harvest ratio of the baseline and rule-based crawlers for the first few thousands of pages for each target topic and seed

Table 3. Comparison of the baseline and rule-based crawlers.

		Seed set 1			Seed set 2		
Target topic	Evaluation metrics	Baseline	Rule based	Improvement (%)	Baseline	Rule based	Improvement (%)
Quantum mechanics	Harvest ratio	0.28	0.30	7.1	0.25	0.29	16
	URL overlap	10%	10%	NA	16%	16%	NA
	Exclusive HR	0.27	0.29	7.4	0.23	0.28	22
Computer history	Harvest ratio	0.29	0.40	38.0	0.36	0.37	3
	URL overlap	27%	27%	NA	22%	22%	NA
	Exclusive HR	0.26	0.39	50.0	0.35	0.37	6
Open source	Harvest ratio	0.52	0.56	9.0	0.48	0.61	27
	URL overlap	10%	10%	NA	8%	8%	NA
	Exclusive HR	0.51	0.54	6.0	0.47	0.61	30

set. The harvest ratios vary among the different topics and seed sets, possibly because of the linkage density of pages under a particular topic or the quality of seed sets. The results show that our rule-based crawler outperforms the baseline crawler by approximately 3 to 38 percent. Second, we provide the URL overlap ratio between the two crawlers. Interestingly, although both crawlers achieve comparable harvest ratios, the URLs they fetched differed significantly, implying that the coverage of these crawlers also differs. For each crawler, we extracted the pages exclusively crawled by it and computed the harvest ratio. The last row of Table 3 for each topic shows that the harvest ratio for pages that the rule-based crawler exclusively crawled is also higher than the harvest ratio for pages that the baseline crawler exclusively crawled.

In our second experiment, we wanted to observe the effects of seed set size on crawler performance. So, we searched Google with the keywords "open" and "source" and used the top 50 URLs to constitute a seed set. The harvest ratios were similar to the corresponding case for the first seed set in Table 3.

In this case, we also plotted the harvest rate, which we obtained by computing the harvest ratio as the number of downloaded URLs increased. The graph in Figure 6 reveals that both crawlers successfully keep retrieving relevant pages, but the rule-based crawler does better than the baseline crawler after the first few hundred pages.

In their recent article,[6] Soumen Chakrabarti, Kunal Punera, and Mallela Subramanyam enhanced Chakrabarti, Van den Berg, and Dom's baseline crawler with an *apprentice*, a secondary classifier that further refines URL ordering in the priority queue. In one experiment, they provided their secondary classifier with the class name of a page from which the URL was extracted, which resulted in a 1 to 2 percent increase in accuracy. They also report that because of a crawler's fluctuating behavior, it's difficult to measure the actual benefit of such approaches. We experienced the same problems, and we're conducting further experiments to provide more detailed measurements.

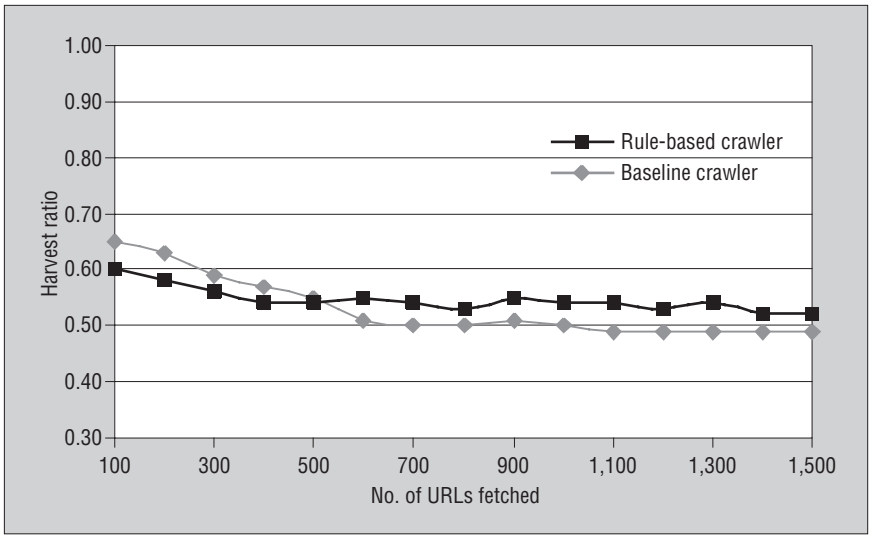


Figure 6. Harvest rates of baseline and rule-based crawlers for the Top.Computer.OpenSource, 50 seeds.

Chakrabarti and his colleagues also investigated the structure of broad topics on the Web.[8] One result of their research was a so-called topic-citation matrix, which closely resembles our interclass rules. However, the former uses sampling with random walk techniques to determine the source and target pages while filling the matrix, whereas we begin with a class taxonomy and simply follow the first-level links to determine the rules. Their work also states that the topic-citation matrix might enhance focused crawling. It would be interesting and useful to compare and perhaps combine their approach with ours.

We can enhance our rule-based framework in several ways. In particular, we plan to employ more sophisticated rule discovery techniques (such as the topic citation matrix), refine the rule database online, and consider the entire topic taxonomy instead of solely using the leafs. Our research has benefited from earlier studies,[2,3] but it has significant differences in both the rule generation and combination process as well as in the computation of final rule scores. Nevertheless, considering the diversity of the Web pages and topics, it's hard to imagine that a single technique would be the most appropriate for all focused-crawling tasks. ■

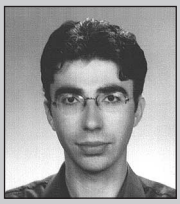
Acknowledgments

We thank Gültekin Özsoyoğlu and Z. Meral Özsoyoğlu for fruitful discussions, and Ö. Rauf Atay for the implementation.

References

1. M. Chau and H. Chen, "Personalized and Focused Web Spiders," *Web Intelligence*, N. Zhong, J. Liu, and Y. Yao, eds., Springer-Verlag, 2003, pp. 197–217.
2. S. Chakrabarti, M.H. Van den Berg, and B.E. Dom, "Focused Crawling: A New Approach to Topic-Specific Web Resource Discovery," *Computer Networks*, vol. 31, nos. 11–16, pp. 1623–1640.
3. S. Chakrabarti, *Mining the Web: Discovering Knowledge from Hypertext Data*, Morgan Kaufmann, 2003.
4. A. McCallum, "Bow: A Toolkit for Statistical Language Modeling, Text Retrieval, Classification and Clustering," 1996, www.cs.cmu.edu/~mccallum/bow.
5. M. Diligenti et al., "Focused Crawling Using Context Graphs," *Proc. 26th Int'l Conf. Very Large Data Bases* (VLDB 2000), Morgan Kaufmann, 2000, pp. 527–534.
6. S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated Focused Crawling through Online Relevance Feedback," *Proc. 11th Int'l World Wide Web Conf.* (WWW 02), ACM Press, 2002, pp. 148–159.
7. T.H. Cormen, C.E. Leiserson, and R.L. Rivest, *Introduction to Algorithms*, MIT Press, 1990.
8. S. Chakrabarti et al., "The Structure of Broad Topics on the Web," *Proc. 11th Int'l World Wide Web Conf.* (WWW 02), ACM Press, 2002, pp. 251–262.

The Authors



Ismail Sengör Altingövde is a PhD candidate in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His research interests include relational and object-oriented databases, Web mining and querying, and information retrieval. He received his MS in computer engineering from Bilkent University. Contact him at the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey; ismaila@cs.bilkent.edu.tr.

Özgür Ulusoy is a professor in the Computer Engineering Department of Bilkent University in Ankara, Turkey. His research interests include Web querying, multimedia database systems, data management for mobile systems, and real-time and active database systems. He received his PhD in computer science from the University of Illinois at Urbana-Champaign. He is a member of the IEEE and the IEEE Computer Society. Contact him at the Department of Computer Engineering, Bilkent University, Ankara 06800, Turkey; oulusoy@cs.bilkent.edu.tr.