Focused Crawling Using Context Graphs

M. Diligenti[†], F. M. Coetzee, S. Lawrence, C. L. Giles and M. Gori[*]

NEC Research Institute, 4 Independence Way, Princeton, NJ 08540-6634 USA
[†]Dipartimento di Ingegneria dell'Informazione, Università di Siena
Via Roma, 56 - 53100 Siena, Italy
{diligmic,gori}@dii.unisi.it, {coetzee,lawrence,giles}@research.nj.nec.com

Abstract

Maintaining currency of search engine indices by exhaustive crawling is rapidly becoming impossible due to the increasing size and dynamic content of the web. Focused crawlers aim to search only the subset of the web related to a specific category, and offer a potential solution to the currency problem. The major problem in focused crawling is performing appropriate credit assignment to different documents along a crawl path, such that short-term gains are not pursued at the expense of less-obvious crawl paths that ultimately yield larger sets of valuable pages. To address this problem we present a focused crawling algorithm that builds a model for the context within which topically relevant pages occur on the web. This context model can capture typical link hierarchies within which valuable pages occur, as well as model content on documents that frequently co-occur with relevant pages. Our algorithm further leverages the existing capability of large search engines to provide partial reverse crawling capabilities. Our algorithm shows significant performance improvements in crawling efficiency over standard focused crawling.

1 Introduction

The size of the publicly indexable world-wide-web has provably surpassed one billion (10^9) documents [1] and as yet growth shows no sign of leveling off. Dynamic content on the web is also growing as time-sensitive materials, such as news, financial data, entertainment and schedules become widely disseminated via the web. Search engines are therefore increasingly challenged when trying to maintain current indices using exhaustive crawling. Even using state of the art systems such as AltaVista's *Scooter*, which reportedly crawls ten million pages per day, an exhaustive crawl of the web can take weeks. Exhaustive crawls also consume vast storage and bandwidth resources, some of which are not under the control of the search engine.

Focused crawlers [2, 3] aim to search and retrieve only the subset of the world-wide web that pertains to a specific topic of relevance. The ideal focused crawler retrieves the maximal set of relevant pages while simultaneously traversing the minimal number of irrelevant documents on the web. Focused crawlers therefore offer a potential solution to the currency problem by allowing for standard exhaustive crawls to be supplemented by focused crawls for categories where content changes quickly. Focused crawlers are also well suited to efficiently generate indices for niche search engines maintained by portals and user groups [4], where limited bandwidth and storage space are the norm [5]. Finally, due to the limited resources used by a good focused crawler, users are already using personal PC based implementations [6]. Ultimately simple focused crawlers could become the method of choice for users to perform comprehensive searches of web-related materials.

While promising, the technology that supports focused crawling is still in its infancy. The major open problem in focused crawling is that of properly assigning credit to all pages along a crawl route that yields a highly relevant document. In the absence of a reliable credit assignment strategy, focused crawlers suffer from a limited ability to sacrifice short term document retrieval gains in the interest of better overall crawl performance. In particular, existing crawlers still fall short in learning strategies where topically relevant documents are found by following off-topic pages.

We demonstrate that credit assignment for focused crawlers can be significantly improved by equipping the crawler with the capability of modeling the context within which the topical materials is usually found on the web. Such a context model has to capture typical link hierarchies within which valuable pages occur, as well as describe off-topic content that co-occurs in documents that are fre-

**Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.**

quently closely associated with relevant pages. We present a general framework and a specific implementation of such a context model, which we call a Context Graph. Our algorithm further differs from existing focused crawlers in that it leverages the capability of existing exhaustive search engines to provide partial reverse crawling capabilities. As a result it has a rapid and efficient initialization phase, and is suitable for real-time services.

The outline of the paper is as follows: Section 2 provides a more detailed overview of focused crawling. Section 3 describes the architecture and implementation of our approach. Comparisons with existing focused crawling algorithms on some test crawls are shown in Section 4, and we conclude by discussing extensions and implications in Section 5.

2 Prior Work in Crawling

The first generation of crawlers [7] on which most of the web search engines are based rely heavily on traditional graph algorithms, such as breadth-first or depth-first traversal, to index the web. A core set of URLs are used as a seed set, and the algorithm recursively follows hyper links down to other documents. Document content is paid little heed, since the ultimate goal of the crawl is to cover the whole web.

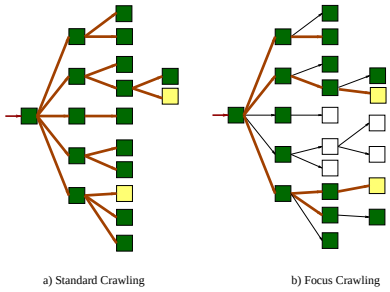

Figure 1: a) A standard crawler follows each link, typically applying a breadth first strategy. If the crawler starts from a document which is i steps from a target document, all the documents that are up to $i-1$ steps from the starting document must be downloaded before the crawler hits the target. b) A focused crawler tries to identify the most promising links, and ignores off-topic documents. If the crawler starts from a document which is i steps from a target document, it downloads a small subset of all the documents that are up to $i-1$ steps from the starting document. If the search strategy is optimal the crawler takes only i steps to discover the target.

A focused crawler efficiently seeks out documents about a specific topic and guides the search based on both the content and link structure of the web [2]. Figure 1 graphically illustrates the difference between an exhaustive breadth-first crawler and a typical focused crawler. A focused crawler implements a strategy that associates a score with each link in the pages it has downloaded [8, 9, 10]. The links are sorted according to the scores and inserted in a queue. A best first search is performed by popping the next page to analyze from the head of the queue. This strategy ensures that the crawler preferentially pursues promising crawl paths.

The simplest focused crawlers use a fixed model of the relevancy class, typically encoded as a classifier, to evaluate the documents that the current document links to (henceforth referred to as children) for topical relevancy [2, 3, 11]. The classifier is either provided by the user in the form of query terms, or can be built from a set of seed documents. Each link is assigned the score of the document to which it leads. More advanced crawlers adapt the classifier based on the retrieved documents, and also model the within-page context of each hyperlink. In the most common adaptive crawlers decision directed feedback is used, where documents that are marked as relevant by the classifier are also used to update the classifier. However, ensuring flexibility in the classifier, without simultaneously corrupting the classifier, is difficult.

A major problem faced by the above focused crawlers is that it is frequently difficult to learn that some sets of off-topic documents often lead reliably to highly relevant documents. This deficiency causes problems in traversing the hierarchical page layouts that commonly occur on the web. Consider for example a researcher looking for papers on neural networks. A large number of these papers are found on the home pages of researchers at computer science departments at universities. When a crawler finds the home page of a university, a good strategy would be to follow the path to the computer science (CS) department, then to the researchers' pages, even though the university and CS department pages in general would have low relevancy scores. While an adaptive focused crawler described above could in principle learn this strategy, it is doubtful that the crawler would ever explore such a path in the first place, especially as the length of the path to be traversed increases.

To explicitly address this problem, Rennie and McCallum [12] used reinforcement learning to train a crawler on specified example web sites containing target documents. The web site or server on which the document appears is repeatedly crawled to learn how to construct optimized paths to the target documents. However, this approach places a burden on the user to specify representative web sites. Initialization can be slow since the search could result in the crawling of a substantial fraction of the host web site. Furthermore, this approach could face difficulty when a hierarchy is distributed across a number of sites.

An additional difficulty faced by existing crawlers is that links on the web are uni-directional, which effectively restricts searching to top-down traversal, a process that we call "forward crawling" (Obtaining pages that link to a particular document is referred to as "backward crawling"). Since web sites frequently have large components that are organized as trees, entering a web site at a leaf can result in a serious barrier to finding closely related pages. In our example, when a researcher's home page is entered, say

via a link from a list of papers at a conference site, a good strategy would be for the crawler to find the department member list, and then search the pages of other researchers in the department. However, unless an explicit link exists from the researcher's page to the CS department member list, existing focused crawlers cannot move up the hierarchy to the CS department home page.

Our focused crawler utilizes a compact context representation called a *Context Graph* to model and exploit hierarchies. The crawler also utilizes the limited backward crawling [13, 14] possible using general search engine indices to efficiently focus crawl the web. Unlike Rennie and McCallum's approach [12], our approach does not learn the context within which target documents are located from a small set of web sites, but in principle can back crawl a significant fraction of the whole web starting at each seed or on-topic document. Furthermore, the approach is more efficient in initialization, since the context is constructed by directly branching out from the good set of documents to model the parents, siblings and children of the seed set.

3 The Context Focused Crawler

Our focused crawler, which we call the *Context Focused Crawler* (CFC), uses the limited capability of search engines like AltaVista or Google to allow users to query for pages linking to a specified document. This data can be used to construct a representation of pages that occur within a certain link distance (defined as the minimum number of link traversals necessary to move from one page to another) of the target documents. This representation is used to train a set of classifiers, which are optimized to detect and assign documents to different categories based on the expected link distance from the document to the target document. During the crawling stage the classifiers are used to predict how many steps away from a target document the current retrieved document is likely to be. This information is then used to optimize the search.

There are two distinct stages to using the algorithm when performing a focused crawl session:

1. An initialization phase when a set of context graphs and associated classifiers are constructed for each of the seed documents

2. A crawling phase that uses the classifiers to guide the search, and performs online updating of the context graphs.

The complete system is shown in Figure 2. We now describe the core elements in detail.

3.1 Generating the Context Graphs

The first stage of a crawling session aims to extract the context within which target pages are typically found, and encodes this information in a context graph. A separate context graph is built for every seed element provided by the user. Every seed document forms the first node of its associated context graph. Using an engine such as Google,

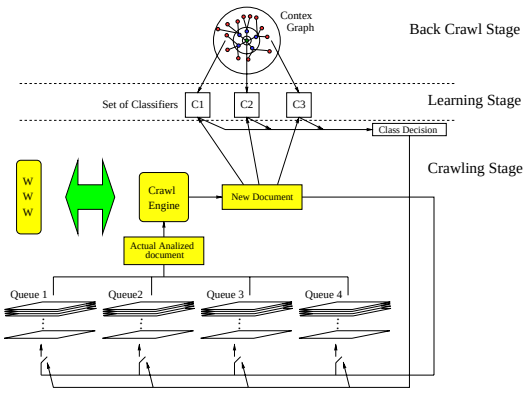

Figure 2: Graphical representation of the Context Focused Crawler. During the initialization stage a set of context graphs are constructed for the seed documents by back-crawling the web, and classifiers for different graph layers are trained. During the crawling stage the crawl engine and the classifiers manage a set of queues that select the next page on which to center the crawl.

a number of pages linking to the target are first retrieved (these pages are called the parents of the seed page). Each parent page is inserted into the graph as a node and an edge is established between the target document node and the parent node. The new nodes compose *layer* 1 of the context graph. The back-crawling procedure is repeated to search all the documents linking to documents of layer 1. These pages are incorporated as nodes in the graph and compose layer 2. The back-linking process is iterated, until a user-specified number of layers have been filled. In practice the number of elements in a given layer can increase suddenly when the number of layers grow beyond some limit. In such a case a statistical sampling of the parent nodes, up to some system dependent limit, is kept. To simplify the link structure, we also use the convention that if two documents in layer i can be accessed from a common parent, the parent document appears two times in the layer $i+1$. As a result, an equivalent induced graph can be created where each document in the layer $i+1$ is linked to one and only one document in the layer i.

We define the *depth* of a context graph to be the number of layers in the graph excluding the level 0 (the node storing the seed document). When N levels are in the context graph, path strategies of up to N steps can be modeled. A context graph of depth 2 is shown in Figure 3.

By constructing a context graph, the crawler gains knowledge about topics that are directly or indirectly related to the target topic, as well as a very simple model of the paths that relate these pages to the target documents. As expected, in practice, we find that the arrangement of the nodes in the layers reflect any hierarchical content structure. Highly related content typically appears near the center of the graph, while the outer layers contain more general pages. As a result, when the crawler discovers a page

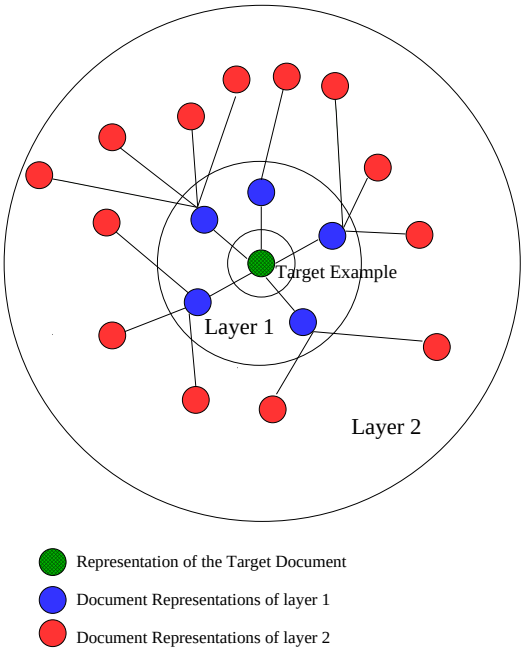

Figure 3: A context graph represents how a target document can be accessed from the web. In each node a web document representation is stored. The graph is organized into layers: each node of layer i is connected to one (and only one) node of the layer $i-1$ (except the single node in layer 0). There are no connections between nodes at the same level. The seed document is stored in layer 0. A document is in layer i if at least i steps (link followings) are needed to reach the target page starting from that document.

with content that occurs higher up in the hierarchy, it can use its knowledge of the graph structure to guide the search towards the target pages. Returning to our example of looking for neural network pages, the context graph will typically discover a hierarchy where levels correspond to researcher home pages, research group pages, and ultimately department and university home pages.

Once context graphs for all seed documents have been built, the corresponding layers from the different context graphs are combined, yielding a layered structure that we call the *Merged Context Graph*.

3.2 Learning Classifiers for the Context Graph Elements

The next stage in initialization builds a set of classifiers for assigning any document retrieved from the web to one of the layers of the merged context graph, and for quantifying the belief in such a classification assignment.

The classifiers require a feature representation of the documents on which to operate. Our present implementation uses keyword indexing of each document using a modification of TF-IDF (Term Frequency Inverse Document Frequency). TF-IDF representation [15] describes a document as a vector relative to a basis of phrases that define a vocabulary V. Each element in the vector represents the frequency of occurrence of the phrase in the document, weighted according to the discrimination implied by the presence of the phrase within a reference corpus. This discrimination is approximated by the phrase frequency in the reference corpus. If two phrases have the same number of occurrences in a document, the TF-IDF value of the less common phrase will be higher. The TF-IDF score $\nu(w)$ of a phrase w is computed using the following function:

$$\nu(w) = \frac{f^d(w)}{f^d{}_{max}} log \frac{N}{f(w)} \quad (1)$$

where $f^d(w)$ is the number of occurrences of w in a document d, $f^d{}_{max}$ is the maximum number of occurrences of a phrase in a document d, N is the number of documents in the reference corpus and $f(w)$ is the number of documents in the corpus where the phrase w occurs at least once.

We implement TF-IDF using the following steps. All the documents in the seed set, as well as optionally, the first layer, are concatenated into a single master document.

1. All *stop-words* such as "by", "and", or "at" are removed from the master document
2. Word stemming is performed to remove common word transformations, such as plurals or case changes [16];
3. TF-IDF computation is performed using a reference corpus derived from an exhaustive web crawl.

The resulting set of phrases form the vocabulary V. When a document is retrieved, the TF-IDF score for phrases in V that occur in the document are computed and placed in a vector. We then truncate the vector representation to include only the forty highest scoring components by zeroing out the remaining components. This processing, yielding a representation which we will refer to as reduced TF-IDF representation, ensures numerical stability and high speed of the classifiers.

Given a representation procedure, we next construct the classifiers. Our goal is to be able to assign any web document to a particular layer of the merged context graph. However, if the document is a poor fit for a given layer, we wish to discard the document, and label it as one of the category "other". The major difficulty in implementing such a strategy using a single classifier mapping a document to a set of $N+2$ classes corresponding to the layers $0, 1, \ldots N$ and a category "other", is the absence of a good model or training set for the category "other". To solve this problem we use a modification of the *Naive Bayes Classifier* for each layer. This classifier architecture provides reasonable performance, high speed, meets the requirement of our system that a likelihood estimate be provided for each classification, and is well studied [17, 18, 12].

Assume that we have a document d_i represented by the vector corresponding to the reduced TF-IDF representation

relative to the vocabulary V. Documents from class c_j, defined to correspond to layer j, are assumed to have a prior probability of being found on the web which we denote $P(c_j)$. The probability that a vector element w_t occurs in documents of class c_j is $P(w_t|c_j)$. To classify a page, we first wish to find the class c_j such that $P(c_j|d_i)$ is maximized. The solution is given by Bayes rule

$$P(c_j|d_i) \propto P(c_j)P(d_i|c_j) \propto P(c_j)P(w_{d_i,1} \dots w_{d_i,N}|c_j) \quad (2)$$

where $w_{d_i,k}$ is the k-th feature of the document d_i. The Naive Bayes assumption ignores joint statistics and formally assumes that given the document class the features occur independently of each other, yielding the final solution

$$P(c_j|d_i) \propto P(c_j)P(d_i|c_j) \propto P(c_j) \prod_{k=1}^{N_{d_i}} P(w_{d_i,k}|c_j) \quad (3)$$

where N_{d_i} indicates the number of features in the document d_i. If N denotes the maximum depth of the context graphs, then $N+1$ discriminant functions $P(c_j|d_i)$ are built corresponding the layers $j = 0, 1, \dots N$.

The discriminant functions allow for a given page d_i to first be assigned to one of the layers of the merged context graph, by finding the layer j^* for which the discriminant function $P(c_j|d_i)$ is maximized. Subsequently, by computing the likelihood function $P(c_{j^*}|d_i)$ for the winning layer j^*, and comparing this likelihood to a threshold, it is possible to discard weakly matching pages. These pages are effectively marked as "other", which avoids the need for construction of an explicit model of all documents not in the context graph. In effect, we build a set of parallel two-class Naive Bayes classifiers, one for each layer, and select the winning layer by maximizing the *a-posteriori* likelihood of the layer based on the context graph.

In training the classifiers, the documents that occur in layer j of all of the seed document context graphs are combined to serve as a training data set D_j. The phrase probabilities $P(w_t|c_j)$ are computed on the sets D_j by counting the occurrences of the feature w_t and then normalizing for all the words in the documents of class c_j:

$$P(w_t|c_j) = \frac{1 + \sum_{d_i \in D_j} N(w_t, d_i) P(c_j|d_i)}{|V| + \sum_{d_i \in D_j} \sum_{s=1}^{|V|} N(w_s, d_i) P(c_j|d_i)} \quad (4)$$

where $N(w_t, d_i)$ is the number of occurrences of w_t in the document d_i and $|V|$ is the number of phrases in the vocabulary V [17, 18, 12].

The parameters $P(c_j)$ can be calculated by estimating the number of elements in each of the layers of the merged context graph. While useful when the layers do not contain excessive numbers of nodes, as previously stated practical limitations sometimes prevent the storage of all documents in the outermost layers. In these cases the class probabilities $P(c_j)$ are set to a fixed constant value $1/C$, where C is the number of layers. This corresponds to maximum likelihood selection of the winning layer. In our experience, performance is not severely impacted by this simplification.

The classifier of layer 0 is used as the ultimate arbiter of whether a document is topically relevant. The discriminant and likelihood functions for the other layers are used to predict for any page how many steps must be taken before a target is found by crawling the links that appear on a page.

3.3 Crawling

The crawler utilizes the classifiers trained during the context graph generation stage to organize pages into a sequence of $M = N+2$ queues, where N is the maximum depth of the context graphs. The i-th class (layer) is associated to the i-th queue $i = 0, 1, \dots N$. Queue number $N+1$ is not associated with any class, but reflects assignments to "other". The 0-th queue will ultimately store all the retrieved topically relevant documents. The system is shown graphically in Figure 2.

Initially all the queues are empty except for the dummy queue $N+1$, which is initialized with the starting URL of the crawl. The crawler retrieves the page pointed to by the URL, computes the reduced vector representation and extracts all the hyperlinks. The crawler then downloads all the children of the current page. All downloaded pages are classified individually and assigned to the queue corresponding to the winning layer, or the class "other". Each queue is maintained in a sorted state according to the likelihood score associated with its constituent documents.

When the crawler needs the next document to move to, it pops from the first non-empty queue. The documents that are expected to rapidly lead to targets are therefore followed before documents that will in probability require more steps to yield relevant pages. However, depending on the relative queue thresholds, frequently high-confidence pages from queues representing longer download paths are retrieved.

The setting of the classifier thresholds that determine whether a document gets assigned to the class denoted "other" determines the retrieval strategy. In our default implementation the likelihood function for each layer is applied to all the patterns in the training set for that layer. The confidence threshold is then set equal to the minimum likelihood obtained on the training set for the corresponding layer.

During the crawling phase, new context graphs can periodically be built for every topically relevant element found in queue 0. However, our focused crawler can also be configured to ask for the immediate parents of every document as it appears in queue 0, and simply insert these into the appropriate queue without re-computing the merged context graph and classifiers. In this way it is possible to continually exploit back-crawling at a reasonable computational cost.

4 Experimental Results

The core improvement of our focused crawler derives from the introduction of the context graph. We therefore com-

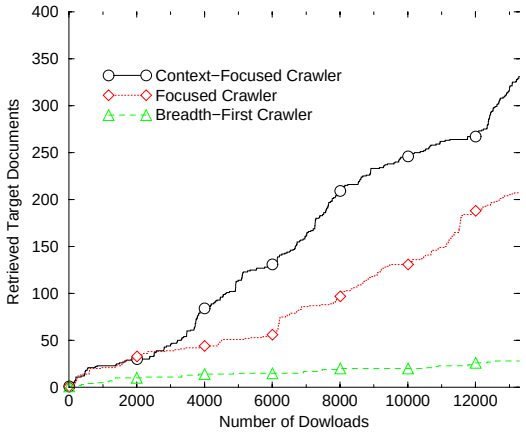

Figure 4: Both the Context Focused Crawler and the standard focused crawler are orders of magnitude more efficient than a traditional breadth-first crawler when retrieving "Call for Papers" documents. The Context Focused Crawler outperforms the standard crawler, retrieving 50 – 60% more "on-topic" documents in a given period.

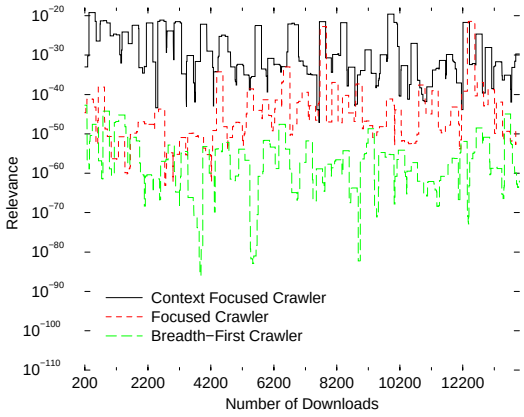

Figure 5: The average relevance of the downloaded documents computed using a sliding window of 200 downloads, illustrating the improved capability of the Context Focused Crawler to remain on-topic.

pare the efficiency of our Context Focused Crawler to two crawlers:

- a breadth-first crawler, using the classifier constructed by the Context Focused Crawler on the seed set;
- a traditional focused crawler which does not use the context to search targets. This crawler evaluates all children of the current document using the same classifier used by the Context Focused Crawler for the seed set, and schedules crawls based on the document with the highest score.

As a test set we performed a focused crawl for conference announcements, and, in particular, for the "Call for Papers" section. We used ten examples as a seed set and constructed ten context graphs of depth four. We limited the number of documents in a single layer of the context graph to 300 . The resulting data was used to learn the 5 Naive Bayes classifiers associated with the five queues.

To evaluate our algorithm we used the accepted metric of measuring the fraction of pages that are on-topic as a function of the number of download requests. The results are shown in the Figure 4. Both of the focused crawlers significantly outperform the standard breadth-first crawler. However the Context Focused Crawler has found on average 50-60% more on-topic documents than the standard focused crawler on the "Conference" task.

The ability of the crawlers to remain focused on "on-topic" documents can also fruitfully be measured by computing the average relevance of the downloaded documents. The relevance of a document is equated to the likelihood that the document has been generated by the Naive Bayes model of the seed set. In Figure 5 we show the average relevance using a sliding window of 200 downloads. In each case the Context Focused Crawler maintains a significantly higher level of relevance than either of the other two crawlers, reflecting the ability of the crawler to use off-topic paths to find new sets of promising documents.

Our experiments showed that the overhead due to the initialization stage is negligible, especially when the overall higher efficiency of the crawler is taken into account.

The improvement that results from querying for the parents of every on-topic document when it is discovered can be seen in Figure 6. This approach shows bursts of rapid retrieval when back-crawling from a target document yields a hub site. We note that for general use we ration the number of back-link requests to avoid over-taxing the search engines.

Figure 7 shows a different category, "Biking", for which our focused crawler showed the least average performance improvement over standard focused crawling (although it still significantly outperforms the standard crawler over most of the trial). We found that such difficult categories are those where target content is not reliably co-located with pages from a different category, and where common hierarchies do not exist or are not implemented uniformly across different web-sites. It is therefore to be expected that the context graph provides less guidance in such cases. However, due to our architecture design, and as illustrated by Figure 4 and Figure 7, the performance will at worst approach that of the standard focused crawling approach.

5 Discussion

We presented a focused crawler that models the links and content of documents that are closely linked to target pages to improve the efficiency with which content related to a desired category can be found. Our experiments show that the Context Focused Crawler improves the efficiency of traditional focused crawling significantly (on average about

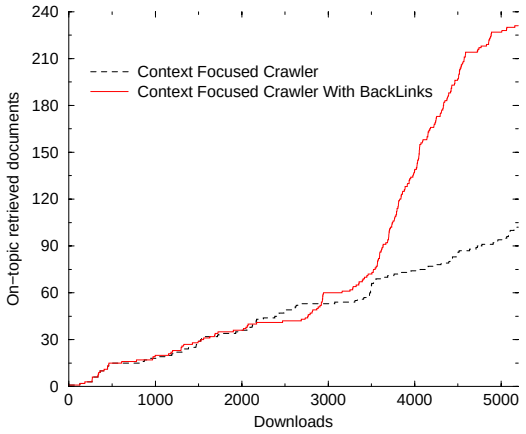

Figure 6: Performance of Context Focused Crawler as compared to Context Focused Crawler with BackLinks, where parents of each on-topic document are obtained from the search engines immediately after the document is discovered. The task is retrieval of "Call for Papers" documents on the web.

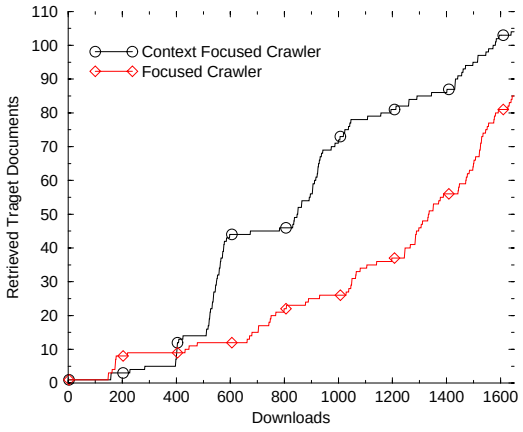

Figure 7: Performance of focused crawlers on the category "Biking".

50-60%), and standard breadth-first crawling by orders of magnitude.

The major limitation of our approach is the requirement for reverse links to exist at a known search engine for a reasonable fraction of the seed set documents. In practice, this does not appear to be problem. However, even when no seed documents have yet been indexed by search engines, the approach can be bootstrapped. In this case a content model of the seed set is extracted and other high-confidence target data can be found using query modifications on a search engine. The indexed target data pages returned by the search engine can then be used to build context graphs. The system can also start with a breadth-first crawl and a set of example documents for training the level 0 classifier, with context graphs being built once a minimum number of target documents have been recovered.

The architecture we presented already yields very good performance, but can be fruitfully improved. Major improvements should result from more sophisticated approaches for setting the confidence thresholds of the classifiers. We are currently investigating other machine learning algorithms for setting these thresholds. A promising approach maintains a graph representation of the current crawl, which a separate on-line process uses to quantify the effect of different queue thresholds.

Other areas of interest are the extension of the feature space to include page analysis of HTML structure, using the statistics gained during the search to develop ranking procedures for presenting results to the user, and performing online classifier adaptation. At present adaptation in our system is restricted to periodically incorporating the context graphs of newly discovered target documents that are found in queue 0, and re-building the classifier (the tests in this paper did not use this feature, to avoid introducing another parameter). Online parameter updating of the classifiers using the EM approach [19] should result in more efficient continuous optimization of the classifier performance.

Not only can our approach be used for background gathering of web material, the computational and bandwidth requirements of the crawler are sufficiently modest for the crawler to be used in an interactive session over a DSL or cable modem connection on a home PC. The focused crawler can therefore be used as a valuable supplement to, and in some cases a replacement for, standard search engine database queries. We have no doubt that further improvement of focused crawling will soon make crawling not only the privilege of large companies that can afford expensive infrastructures, but a personal tool that is widely available for retrieving information on the world wide web.

References

[1] "Web surpasses one billion documents: Inktomi/NEC press release." available at http://www.inktomi.com, Jan 18 2000.

[2] S. Chakrabarti, M. van der Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," in *Proc. of the 8th International World-Wide Web Conference (WWW8)*, 1999.

[3] J. Cho, H. Garcia-Molina, and L. Page, "Efficient crawling through URL ordering," in *Proceedings of the Seventh World-Wide Web Conference*, 1998.

[4] A. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Building domain-specic search engines with machine learning techniques," in *Proc. AAAI Spring Symposium on Intelligent Agents in Cyberspace*, 1999.

[5] A. K. McCallum, K. Nigam, J. Rennie, and K. Seymore, "Automating the construction of internet por-

tals with machine learning," *To appear in Information Retrieval*.

[6] M. Gori, M. Maggini, and F. Scarselli, "http://nautilus.dii.unisi.it."

[7] O. Heinonen, K. Hatonen, and K. Klemettinen, "WWW robots and search engines." Seminar on Mobile Code, Report TKO-C79, Helsinki University of Technology, Department of Computer Science, 1996.

[8] S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg, "Automatic resource compilation by analyzing hyperlink structure and associated text," in *Proc. 7th World Wide Web Conference, Brisbane, Australia*, 1998.

[9] K. Bharat and M. Henzinger, "Improved algorithms for topic distillation in hyperlinked environments," in *Proceedings 21st Int'l ACM SIGIR Conference.*, 1998.

[10] J. Kleinberg, "Authoritative sources in a hyperlinked environment." Report RJ 10076, IBM, May 1997, 1997.

[11] S. Chakrabarti, M. van den Berg, and B. Dom, "Distributed hypertext resource discovery through examples," in *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK* (M. P. Atkinson, M. E. Orlowska, P. Valduriez, S. B. Zdonik, and M. L. Brodie, eds.), pp. 375–386, Morgan Kaufmann, 1999.

[12] J. Rennie and A. McCallum, "Using reinforcement learning to spider the web efficiently," in *Proc. International Conference on Machine Learning (ICML)*, 1999.

[13] K. Bharat, A. Broder, M. Henzinger, P. Kumar, and S. Venkatasubramanian, "The connectivity server: Fast access to linkage information on the web," *WWW7/ Computer Networks 30(1-7)*, pp. 469–477, 1998.

[14] S. Chakrabarti, D. Gidson, and K. McCurley, "Surfing backwards on the web," in *Proc 8th World Wide Web Conference (WWW8)*, 1999.

[15] G. Salton and M. J. McGill, *An Introduction to Modern Information Retrieval*. McGraw-Hill, 1983.

[16] M. Porter, "An algorithm for suffix stripping," *Program*, vol. 14, no. 3, pp. 130–137, 1980.

[17] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[18] K. Nigam, A. McCallum, S. Thrun, and T. Mitchell, "Text classification from labeled and unlabelled documents using EM." To appear in Machine Learning, 1999.

[19] A. Dempster, N. Laird, and D. Rubin, "Maximum likelihood from incomplete data via the EM algorithm," *J. R. Statist. Soc. B*, vol. 39, pp. 185–197, 1977.