Distributed Ontology-Driven Focused Crawling

Rodrigo Campos, Oscar Rojas, Mauricio Marín
Computer Science Department
Universidad de Santiago
Santiago, Chile
{rodrigo.camposgu, oscar.rojas.d, mauricio.marin}@usach.cl

Marcelo Mendoza
Computer Science Department
UTFSM
Santiago, Chile
marcelo.mendoza@usm.cl

Abstract—Focused crawlers are programs designed to download web pages which are relevant to specific topics. Using information gathered at running time, focused crawlers explore the web following promissory hyperlinks and fetching only pages which appear to be relevant. These crawlers are receiving increasing attention because they favor the construction of vertical search engines, allowing users to focus on specific topics of information, providing higher accuracy and reducing computational costs involved in query processing. In this article, we introduce an efficient focused crawling strategy which considers a number of distributed focused crawlers which recover relevant pages to a given knowledge domain. We propose an ontology-based knowledge representation approach to drive the crawler to specific segments of the web. Experimental results with actual samples of the Web show the feasibility and efficiency of our strategy.

Keywords—Focused crawling; ontologies; vertical search

I. Introduction

The vast amount of information in the Web represents one of the most striking challenges for computer science in recent decades. On the one hand, the emergence of the Web has offered the opportunity to access more information than ever before. On the other hand, the wide range of topics covered and the diverse quality of these resources has pushed to the limit the development of new search technologies.

A significant number of people use Web search engines to formulate queries (a set of terms) and review a list of suggested answers. Search engines are built from practical implementations of information retrieval techniques devised to handle large-scale Web collections. These collections are constructed by using a crawler, which is a software that periodically traverses the Web by following the Web pages hyperlink structure, fetching relevant Web pages (documents) and other kind of potentially useful documents. Web collections contain millions of documents over a wide range of topics. Thus, the search for a relevant resource is like to find a needle in a haystack.

An increasing interest in the use of new specialized search engines has focused many efforts in the development of vertical search technologies. Vertical search engines are devised to serve on specific topics of information, providing higher accuracy than general purpose (horizontal) search engines and reducing computational costs involved in query processing. Vertical search engines employ focused crawlers to collect pages relevant to specific topics.

A focused crawler attempts to download (and thereby consume bandwidth for) only Web pages which appear to be relevant to a given topic. In practice, this task is very difficult. At running time, the crawler must decide whether a Web page is relevant before downloading it by just examining the hyperlink that points to its actual content. Once a page is fetched, the crawler uses the content of the page or a more specific portion of its text to decide whether to follow an outgoing link. This process continuously repeated by following the most promissory links, fetching each promising Web page, and so on. In practice, the Web is traversed by conducting a best first crawling process.

In this article we propose a distributed focused crawler framework which interacts with an ontol-
ogy devised to focus the crawling processes to the segments of the Web where relevant Web pages are located. Our crawler implementation allows us to evaluate distinct state-of-the-art Web traverse strategies, URL scoring functions, and source page text segmentation schemes. We conduct experiments to test the feasibility of our approach and the results show that the proposed crawler is effective and efficient.

The article is organized as follows. In Section 2 we review related work. The proposed crawler is described in Section 3. Experiments are presented in Section 4 and Section 5 presents concluding remarks.

II. RELATED WORK

Since the emergence of the Web, there has been many efforts oriented to collect and index specific portions of the Web, relevant to particular topics or specific information needs. Most of the early focused crawlers are essentially based on the use of a scoring component which evaluates the relevance of a given user query (a set of terms) to a fetched page. Based on the page score, the crawler decides whether or not to follow its outgoing links. Two representative focused crawlers which use this approach are Fish search Bra and Post (1994) and Shark search Hersovici et al. (1998).

A research line attempts to produce better information needs representations by using ontologies. Ontologies are systematic and structured body knowledge representations generated by experts. They include a set of concepts, described by using keywords, instances and relationships among these concepts. One of the first focused crawlers which considered the use of ontologies was introduced in Ehrig and Maedche (2003). This strategy uses the lexical part of the ontology to calculate scores for the fetched pages. Then, the relations of the ontology are used to assign more accurate scores, deciding which pages are more relevant to the existing ontology.

A more sophisticated approach was proposed by Zheng et al. (2008), where an ontology is used to train an Artificial Neural Network (ANN) which is applied to the classification of the crawled pages. In practice, the lexical part of the ontology is used as the input of the ANN. Chen et al. (2008) also uses the lexical part of the ontology to define a hierarchical distance notion for page scoring, useful for focused crawling.

Finally, the important question about how to evaluate focused crawlers is addressed by Srinivasan et al. (2005), where several measures are evaluated and compared regarding different performance aspects of crawling, such as efficiency (bandwidth use, downloads / targets), efficacy (precision, coverage, freshness), and topical specificity.

There are many good resources to conduct a deeper exploratory study about several aspects of focused crawling. Among these, one of the first extensive documents about focused crawlers Chakrabarti (2002), where a complete book chapter address the topic. A complete review of adaptive crawlers can be found in Micarelli and Gasparetti (2007). Recently, an overview of crawling was included in Menczer (2011), where some sections are specifically dedicated to focused crawling.

III. PARALLEL ONTOLOGY-DRIVEN FOCUSED CRAWLING

We propose a distributed focused crawler framework, which interacts with an ontology to focus the crawling processes to the segments of the Web where the most promising pages are expected to be hosted. The software architecture of the distributed crawler allows us to evaluate distinct state-of-the-art Web traverse strategies, URL scoring functions, and source page text segmentation schemes.

We start the crawling process by selecting a set of relevant pages to each of the concepts represented in the ontology. To do this, we use an external page ranking method where the keywords which compose each concept description in the ontology are used to retrieve the top-k results. Each page retrieved is considered as a seed for the crawling process. In our implementation we used the external ranking method known as Yahoo! Boss\(^1\).

For each concept of the ontology we create a crawler with its own set of seeds. Each crawling

\(^1\)http://developer.yahoo.com/search/boss/
process collects its own set of pages by following a best-first focused crawling strategy in accordance with an URL ranker which we have devised to calculate a relevance score for each of the outgoing links of a fetched page. The text of each fetched page is segmented according to its DOM structure, allowing to specify the portions of the page that are used to build a term-based vector representation of each outgoing link. The set of keywords which describes each concept of the ontology is also used to build a vector representation. This allows us to calculate the distance to each outgoing link by using inner dot product based-distance functions such as the Cosine distance. Then, these scores are used for URL prioritization. A duplicate detection module discard previously fetched pages by comparing URLs. Then, each new unseen URL is enqueued by following the order defined by its score. We illustrate this process in Figure 1.

Figure 1 shows how a single crawling process interacts with a central focused Web repository represented by the “crawl database” module. This repository stores the text of each fetched page for indexing, allowing to calculate frequency-based features at term level such as the inverse document frequency of each indexed term. This feature can be used by the page ranker module of each crawler. However, as our framework considers multiple crawlers, we introduce coordination among them to prevent from computing duplication. First, we consider a shared memory space where we allocate a global structure to avoid the fetching of previously indexed pages without requiring accesses to secondary memory. We explore also how to recommend a fetched page to be considered as a seed. This process requires coordination because a fetched page can be considered as a seed for any concept of the ontology.

In summary, our crawler implementation considers several particular processes and each one of these includes particular design criteria. Now we explain more details of each module of the crawling framework.

**Scheduler:** Our scheduler is based on an architecture proposed by Castillo et al. (2004). This architecture allows us to keep control over two dimensions of the scheduling problem in the focused crawling domain: 1) A long-term based scheduler, where best pages are visited at the beginning of the crawling process, 2) A short-term based scheduler, where the decision about how to traverse a given site is defined, according to a politeness policy and bandwidth use. Castillo et al. (2004) proposed this architecture for universal crawlers, showing that by modeling both dimensions of the problem it is possible to achieve a good balance between efficiency (bandwidth use and politeness) and effectiveness (how to fetch good pages first). Based on this evidence, we use for our focused crawling architecture two levels of prioritization, the first one related to the long-term scheduling dimension, where the ordering is conducted at site-level, and the second one related to the short-term scheduling dimension, where the Web pages of each site are sorted according to its score.

**DOM parser and information extraction:** For a given fetched page, we represent the content of each outgoing link by using the text of the source page. We explore several fragments of the page to do this, such as the anchor text, the closest header and title to the URL, and the URL in itself. We explore also the use of a window text which surrounds the URL, known as the “surrounding text”. The extraction of the surrounding text is far from being an easy problem. The content of a Web page is structured by using HTML encoding.
but the distribution of the text into the HTML is conducted to produce a visual deployment of the information in the Web browser. Thus in many cases it is very difficult to determine the text window which surrounds the URL. We use the DOM structure of the HTML encoding to perform this task. Figure 2 illustrates this process.

The DOM structure of a Web page corresponds to a HTML tag tree. In the example of Figure 2, we point to an outgoing link about "Don Giovanni" (see 2-a). We avoid the inclusion of the remaining anchor texts, which are related to other outgoing links hosted in this page (see 2-b), by traversing the DOM tag tree at the target page level and extracting only the text at this level (see 2-c).

**Page ranker:** Each outgoing link is prioritized by the page ranker module. This module is the core of the crawler architecture. To do this, our page ranker interacts with ontologies which allow us to represent a specific body of knowledge. An ontology is a formal specification of a shared conceptualization, which is composed by a set of concepts, properties, and relationships among concepts. Formally, a body of knowledge is represented by an ontology \( \mathcal{O} \), where a set of concepts \( C \) and properties \( \mathcal{P} \) among them, \( \mathcal{P} : \rightarrow C \times C \) is consolidated. Both sets are consolidated by using a structure \( \mathcal{H}^{C,P} \subset C \times C \), where each pair \((c_1, c_2) \in \mathcal{H}^{C,P}\), is labeled with a property of \( \mathcal{P} \). It is usual also to include a set of instances \( I \), where an instantiation function \( \text{In} : C \rightarrow I \) maps each concept to one or more concept instances. Thus, the ontology is the composition of concepts, properties, instances, and structures that relate them. Finally, each ontology entry (concept, property or instance) is augmented with lexical entries that favors the representation of each ontology entity by a set of alternative keywords.

Our page ranker takes advantage of the ontology knowledge representation to focus the crawling process to a specific domain. We follow a vector space-based approach for URL prioritization, where each concept of the ontology is represented by a term-vector. Each outgoing link is also represented by a term vector, constructed from the descriptive texts extracted by the DOM parser. Then we use vector pairs URL - concept to calculate inner products in the vector space. We use a standard TF-IDF term vector representation with a cosine distance function.
We explore several alternatives to build the descriptive text of the ontology concepts. In practice, the descriptive text of a given concept is a set of keywords, but an ontology offers several possibilities to conduct concept expansion operations. We have implemented and evaluated several of these expansion strategies in our crawler to study its performance.

Our parallel architecture: We propose a parallel architecture for our crawler which is based on a master-slave approach. The master coordinates each slave crawler by using a shared memory where local results recovered from each crawler are buffered avoiding page duplication.

Each slave crawler is focused on a specific concept extracted from the ontology, where a set of seeds is defined to start the process. Each crawler considers a scheduler which prioritize the pages to be fetched. Following the order proposed for the scheduler, the pages are fetched and indexed in a central database managed by the master.

Each crawler follows the instructions defined by the master, among them from what seed the process starts and which ranker will be used. Each slave considers a control thread that requests in the shared memory buffer for new assigned tasks. We illustrate the general architecture of the crawler in Figure 3.

As Figure 3 shows, the master considers a queue of pending jobs. The master coordinates the parallel crawlers by setting parameters such as the number of pages to be retrieved by each crawler, global page classification thresholds, the number of threads to be assigned to each topic defined from the ontology. All these data is published by the master in a shared buffer in the form of jobs assignments for each crawler. The communication between the different crawlers and the master is by means of concurrent accesses to this buffer. In addition, the master is responsible for generating URL seeds by getting them from the Yahoo! Boss API or loading them from a list of provided manually. Each crawler contains a scheduler that is in charge of distributing the work among the machine threads and they exchange URLs each other by means of a slave level shared data buffer. In both shared buffers, standard lock-based concurrency control strategies are used to prevent from read and write conflicts. Detailed flow processes for the master/slave components of our architecture are depicted in Figures 4 and 5.

IV. Experimental results

A. Crawling order

We started our experiments by evaluating the ranker and scheduler aspects of our proposal which are basically related to the order in which the pages are fetched from the Web. In an initial collection of 4,000 downloaded pages we measured the running time required to parse a given content fragment. This aspect is relevant as it can limit the downloaded page throughput achieved by the crawler. We considered 3 content fragments for each page: the header, the surrounding text recovered by using our DOM-based strategy, and the anchor text. We
considered also the full page for this evaluation. We show in Figure 6 the cumulative parsing time for batches composed of 500 downloaded pages. Notice that the inclusion of our DOM strategy to conduct the focused crawling process is not running time consuming.

In a second experiment we tested different ranker strategies. For evaluating the different strategies we calculated the similarity between the body of the fetched page and a given query considering these scores as a gold standard. Then the similarity between the backlinked pages and each query was calculated by considering a number of descriptive content fragments in the collection of pages which links to the fetched pages, allowing the evaluation of the ranker predictive ability. Each query was created by including terms of the ontology by starting from the leaf node which describes the specific topic and aggregating terms found along the path to the root in the ontology tree.

First we conducted this experiment by considering an ontology which describes science topics. In particular we tested our crawler in the specific topic “exact sciences”. We considered the ontology “TemaTrees”, available on http://vocabularyserver.com, which includes formal representations of several knowledge domains. TemaTrees is an online controlled vocabulary server where specific fragments of the vocabulary can be exported in formats as Dublin Core or Topic Maps. We show the results of the evaluation in Figure 7.

As Figure 7 shows, the best combination is achieved by the anchor + header ranker, where the most significant fraction of relevant pages is recovered. On the other hand, the worse combinations are the ones which consider the URL as it occurs in standard practice. In fact, when the surrounding text is considered by the ranker, previous combinations increase their performance, illustrating that the use of surrounding text or the header offers good predictive abilities.

In a third experiment we compared the performance of our scheduler against state of the art crawling strategies as breadth first and depth first.
Breadth-first tends to collect pages by traversing web sites level by level. Depth-first tends to collect all links derived from a given page first.

Our scheduler picks the best ranked link to conduct the focused crawling process, by considering the header + anchor + surrounding text ranker combination. We conducted this experiment on two additional topics, “Health” and “Java”, which were recovered from the TemaTrees ontology. We show these results in Figures 8 and 9, respectively.

Figure 8. Scheduler evaluation for the topic “Health”.

Figure 9. Scheduler evaluation for the topic “Java”.

As figures 8 and 9 show, our strategy achieves the best performance, which an additional fact that illustrates that our proposal is feasible for focused crawling and achieves better performance than alternative approaches.

B. Evaluation of parallelism

We evaluated the parallel performance of our proposal by implementing our strategy on a cluster with 8 processors. Each processor conducted a specific crawling process. A master process coordinated the tasks in the processors, according to the architecture discussed in Figure 3.

A maximum number of 1,000 crawled pages per processor was used in this evaluation. The timeout for the experiment is achieved when the last page (the 8,000-th fetched page) is downloaded. We measured the total amount of time required for fetching the entire collection by using 1, 2, 4 and 8 processors. We measured the fraction of the fetching time against the time required by a single processor, which allows us to calculate the speed up of our proposal per number of processors, which is shown in Figure 10.

Figure 10. Speed-up for number of processors

Figure 10 shows that the fraction of time required for fetching the entire collection decreases when we considers more processors, fact which is intuitive when a scalable strategy is being tested. In our case, we can observe that the relationship between time and number of processors is almost linear, namely it scales up efficiently.

In the following, we illustrate which part of the crawling process takes more running time to perform its task. We evaluate this aspect by measuring the average parsing, ranking, and fetching running times per number of processors. These results are shown in Figure 11.
Figure 11 shows that the amount of time required for fetching decreases when the number of processors increases. On the other hand, the parsing time increases when we consider more processors. The ranker time remains the same in all the cluster configurations, taking only a marginal fraction of the total time required to complete the task.

V. Conclusion

In this article, we have presented an efficient focused crawling strategy which considers a number of distributed crawlers which recover relevant pages to specific topics. We use an ontology-based knowledge representation to drive each crawler to specific fragments of the web.

The experimental results show the efficiency of our strategy. The results show that the use of specific content fragments (header + anchor + surrounding text) is more suitable for link ranking than the use of the URL text or the full page. Our scheduler outperforms well-known crawling strategies such as breadth first and depth first. In addition our proposal does not demand additional significant computational costs. Regarding parallelism, processing times and speed-ups show good performance figures when we increase the number of processors. In fact, the relationship between the number of processors and the time required to complete the crawling process is almost linear. Moreover, fetching times decrease when we consider include more processors, and the time required to calculate ranking scores takes a marginal fraction of the total crawling time.

All these performance measures show that the proposal of this paper is of practical use as (1) it is able to discover and retrieve Web pages on specific topics by using a knowledge ontology and (2) it achieves efficient running time performance and can be efficiently parallelized on distributed memory clusters of processors.

ACKNOWLEDGMENT

The authors were supported by FONDEF Grant DO8I1155 from CONICYT, Chile.

REFERENCES


