xCrawl: A High-Recall Crawling Method for Web Mining

Kostyantyn Shchekotykhin[1], Dietmar Jannach[2] and Gerhard Friedrich[1]

[1]University Klagenfurt, 9020 Klagenfurt, Austria
[2]Technische Universität Dortmund, 44221 Dortmund, Germany

Contact author: dietmar.jannach@tu-dortmund.de

Abstract. Web Mining Systems exploit the redundancy of data published on the Web to automatically extract information from existing web documents. The first step in the Information Extraction process is thus to locate as many web pages as possible that contain relevant information within a limited period of time, a task which is commonly accomplished by applying *focused crawling* techniques. The performance of such a crawler can be measured by its "recall", i.e. the percentage of documents found and identified as relevant compared to the total number of existing documents. A higher recall value implies that more redundant data is available, which in turn leads to better results in the subsequent fact extraction phase.

In this paper, we propose xCrawl, a new focused crawling method which outperforms state-of-the-art approaches with respect to the recall values achievable within a given period of time. This method is based on a new combination of ideas and techniques used to identify and exploit the navigational structures of websites, such as hierarchies, lists or maps. In addition, automatic query generation is applied to rapidly collect web sources containing target documents.

The proposed crawling technique was inspired by the requirements of a Web Mining System developed to extract product and service descriptions and was evaluated in different application scenarios. Comparisons with existing focused crawling techniques reveal that the new crawling method leads to a significant increase in recall whilst maintaining precision.

1 Introduction

The initial task of Web Mining Systems (WMS) like AllRight [1], SemTag [2], Deadliner [3] WEB→KB [4] and others that exploit the redundancy of data published on the Web for Information Extraction (IE) is to retrieve a suitable set of documents from the Web. Best results in the subsequent data extraction step of a WMS will of course be achieved if (a) the retrieved collection of documents only contains pages that are relevant

with respect to the extraction goal and (b) as many suitable documents as possible have been collected. If the goal of the WMS for instance is to automatically extract digital camera specifications from the Web, the document retrieval process should only consider pages that actually contain such specifications while at the same time it should try to find as many relevant pages as possible (e.g. in order to resolve inconsistencies or to measure the plausibility of the extracted information).

These two desirable properties are commonly measured in terms of the "precision" and "recall" of the set of collected documents. Today's WMSs, such as those mentioned above, are able to accurately validate the retrieved documents, i.e. they can efficiently determine whether a given document contains the desired information or not. However, evaluations of current approaches to Information Retrieval (IR) like *focused crawling* and *automatic query generation* (AQG) have shown rather low recall values for important mining domains such as digital consumer product specifications or their reviews. In these scenarios, documents containing target data tend to be located deep in the navigational structure of a website and force a crawler to download and validate a lot of irrelevant documents, which ultimately makes the WMS slower and less precise. The latter follows from the fact that the more documents that have to be validated by a WMS, the higher the probability that it will also accept irrelevant information.

In this work we propose a document retrieval method called xCrawl which combines new and existing techniques to quickly guide a focused web crawler toward the relevant web documents, thus improving the WMS's recall. The main idea of the approach is to exploit existing navigational structures like index pages, hierarchical category structures, menus, and site maps that are typically found in modern websites. The xCrawl algorithm aims to find pages that act as information *hubs*, i.e. web pages that contain a lot of links that directly lead the crawler to relevant documents, which we refer to as *authorities*. Consequently, the automated discrimination between hubs and authorities is the key problem in this context. The solution suggested in this paper is based on Kleinberg's HITS [5] algorithm that was successfully applied to a number of similar graph partitioning problems. Moreover, our evaluation shows that xCrawl can achieve even better recall values if HITS is used in combination with a topic-sensitive PageRank [6] as a proximity measure. Finally, the crawler follows a *random walk with restart* (RWR) strategy, which prevents it from getting stuck in irrelevant branches of a website's navigational structure.

In order to automate the process of finding a suitable initial set of websites, xCrawl also includes a simple AQG algorithm. This algorithm submits queries to public search engines using a set of sample documents from the target domain. The obtained results are used to (a) identify websites that contain target data to be extracted and (b) obtain an initial set of web pages for the focused crawling algorithm.

The paper is organized as follows. In Section 2, we give an overview of existing approaches and standard techniques for document retrieval in WMSs. An example in Section 3 illustrates the main ideas of the crawling technique, which are described in detail in Section 4. The results of an experimental evaluation in different application domains that show that xCrawl outperforms a state-of-the-art focused crawler with respect to the recall measure are presented in Section 5. Further implementation details as well as a short description of the ontology-based AllRight WMS in which xCrawl is embedded, are given in Section 6. The paper ends with a summary and an outlook on future work.

2 Background & Previous work

2.1 Background

Web crawling is current standard technique for the automated retrieval and refreshing of web documents in the context of WMSs and search engines [7]. Figure 1 depicts the general architecture and the main components of a (focused) crawling system. Crawlers, also called spiders, explore the Web by iteratively downloading individual pages and extracting hyperlinks that point to further pages.

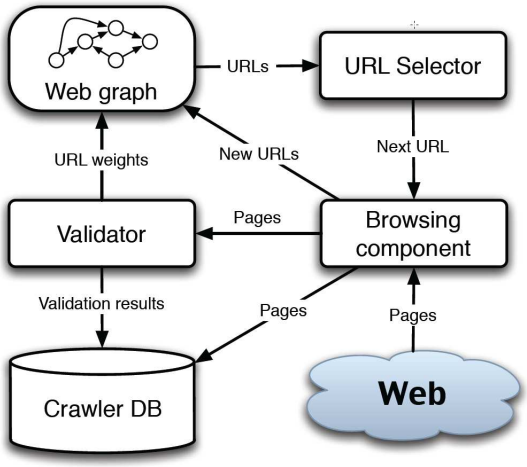

Fig. 1. A general focused crawler architecture.

In detail, the process typically starts with a number of seed URLs provided by the user which are stored in the crawler's internal presentation of the Web. This internal representation often has the form of an directed graph $G = (V, E)$, called *web graph*, in

which the vertices $v_i \in V$ correspond to web pages and edges $e_j \in E$ correspond to the hyperlinks found on each page. Since the crawler adds edges for all hyperlinks found on a web page, the set of nodes V includes both visited V_v and unvisited V_u pages. Moreover, in the context of focused crawling, the web graph is often a weighted graph, allowing a crawler's URL Selector component to compute different node proximity measures and thus to control the crawling process.

The *URL Selector* component chooses one of the unvisited nodes in V_u for further exploration and forwards the corresponding URL to the browsing component. The selection process can be based on various web graph traversal strategies. The simplest one is to start from a website's root page and to follow the links in a breadth-first order. More elaborate strategies might include techniques for detecting (and ignoring) URLs that are syntactically different but point to the same page, for example `http://news.google.com` and `http://google.com/news` [8]. The most recent URL selection strategies are based on proximity measures between the nodes of the web graph [9][1].

The *Browsing component* actually downloads a webpage with a given URL. Then it processes the retrieved page and extracts all the hyperlinks found on the page as well as other information required by the Validator component. Note that nowadays menus and other parts of web pages containing hyperlinks on the modern websites are often created dynamically. Thus, the analysis of static HTML code is no longer sufficient and more elaborate extraction techniques are required that are for instance also capable of interpreting JavaScript code in order to detect all links; see also Section 6.

In contrast to search engines that are designed to acquire all accessible pages with any content, WMSs are typically only interested in pages that contain information about a certain topic, e.g. digital camera specifications. Thus, the question of where to proceed in the web graph expansion process strongly depends on the goal of the document retrieval task. Therefore, a WMS crawler includes an additional *Validator component* that is capable of determining whether a given page is relevant with respect to the web mining goal. Consequently, not only the web pages themselves must be stored in the *Crawler DB*, but also the results of the validation process, i.e. whether a certain page is relevant or not. Different validation techniques may be chosen depending on the application domain and the mining goals. Existing systems for instance use Support Vector Machines [11], Bayesian Networks [4] or – as in the AllRight WMS, for which xCrawl was developed – special table recognition and fact extraction techniques [1].

Finally, the results of the validation process are also used to guide the crawler toward more relevant pages (focused crawling). For instance, one can influence the URL selec-

[1] For an in-depth discussion of further technical aspects of web crawling systems and of other common issues encountered in search engine crawler implementation, see e.g. [10].

tion process by modifying the weights associated with the web graph's nodes and edges according to the outcome of the validation process as shown in Figure 1.

2.2 Relation to previous work

The general architecture presented in the previous section is implemented by most existing focused crawlers, such as those described in [12–16]. Most of the components of these implementations differ only in terms of technical details and validation technologies. Indeed, the main differences tend to be located in the *URL Selector* component. This component implements the proximity measure, which is used to determine the similarity between any node of the web graph and nodes that are relevant to the mining goal. Note also, that the URL Selectors of some focused crawlers are designed only for use with particular validation technologies. On the one hand such crawlers benefit from better usage of specific validation feedback, however they are inapplicable to some domains where such validation technology performs poorly.

In general, proximity measures are context-dependant, graph-dependant or a combination of the two. The measures belonging to the first class, as used in [16, 14] analyze only the content of web pages to compute the proximity values for unvisited nodes. Graph-dependant measures, such as PageRank [17], only exploit the structure of a web graph. Finally, mixed approaches, such as the one presented in [15], try to improve their results by using a combination of both types of measures.

An early intelligent crawler described in [16] uses a context-based proximity measure maintained via reinforcement learning. The main task is to learn a function that, given a set of text tokens (taken from the neighborhood of a hyperlink as well as headers and title words), returns an estimate of how many interesting pages will be found when the crawler follows the hyperlink. Experiments show that the method presented in [16] significantly improves web crawling performance when compared to a baseline breadth-first crawler.

Chakrabarti et al. in [14] subsequently proposed another context-based measure that is also based on learning. The authors suggest using a Bayesian Network classifier, called *apprentice*, trained to predict the relevance of an unvisited page given a hyperlink and its context. The context of a link is defined by the content of specially preselected Document Object Model (DOM) elements surrounding the hyperlink on a page. The crawler follows the hyperlink with the best score and downloads the corresponding document. Then the baseline classifier analyzes the document and determines its relevance. Classification feedback together with the context of the hyperlink is used to continuously train the apprentice, thus improving its classification accuracy.

However, the described methods only work well on websites, which embed their navigational structure within the page's text blocks. In such settings, each link is surrounded

by data that can be efficiently used to train the URL selector's classifier. However content-based proximity measures perform poorly when applied to some web site types, like review sites, as most of the data is published without accompanying textual descriptions. For instance, on review websites the navigational structure is typically presented as a series of product lists. Each list includes a large number of related product names and hyperlinks to documents describing the products. Consequently, no textual data may be utilized by the URL Selector to distinguish between links pointing to relevant and irrelevant products. In such cases a graph-based proximity measure, such as HITS, is often the best choice. Using this measure a crawler can identify URLs of target documents solely from the structure of the web graph.

"Context-focused crawling" (CFC) [15] is a learning-based approach that uses a mixture of both content and graph-dependant measures. The method applies a *reverse crawling* approach by querying public search engines to locate pages that contain links to a given document. Using this technique the crawler constructs so-called "Context Graphs" that describe special link hierarchies in which relevant documents appear. Based on this representation and a context-based TF-IDF measure, classifiers are trained to predict the minimum "link distance" (i.e. minimum number of links to be followed) from a given document to a target document. The predicted values are used to guide the crawling process.

The main drawback of CFC is that its results are completely dependent on the results returned by the search engines. As our experiments show these results can be unsatisfactory for the reverse crawling purposes (see Section 5). Moreover, the method generates a large number of queries to search services. This is particularly problematic in scenarios where search engine providers limit the number of queries from one client to reduce server load.

In addition to the proximity measure, focused crawlers have to solve another important problem, namely, how to detect additional websites suitable for crawling. Some crawlers assume that important websites that contain relevant documents are interlinked and that sooner or later the crawling process will automatically find them. Another approach is to (additionally) rely on publicly accessible directories such as `dmoz.org` which contain manually created lists of links to important websites. Such directories may, however, be incomplete or out-of-date since they are manually engineered. In [18], another approach based on a two-level architecture consisting of two web crawlers is proposed: an *external crawler* whose task is to locate interesting websites and an *internal crawler* that performs focused crawling on a given site.

Automated Query Generation (AQG) is another method of retrieving relevant documents by querying public search engines, which have already indexed large portions of the Web. The general workflow of an AQG-based system can be described as follows: a

number of seed tokens are extracted from relevant documents provided by the user and used to generate an appropriate search query. Then a predefined number of resulting documents are retrieved and analyzed, allowing more tokens to be extracted and added to the seed tokens. This procedure is repeated until some stop criterion is met. While such an approach can be very efficient as it re-uses existing document indexes, it requires an appropriate AQG method in order to ensure that the desired documents are returned by the search engine.

The *QXExtract* system [19] is an example of a query-based system designed for the fast retrieval of documents for information extraction from large text databases. The system first automatically extracts features from the text documents of the repository based on some user-specified seed tuples. Based on these examples, further queries to the repository are constructed with the aim of discovering further documents that are similar to the positive examples. For the query generation task, three different methods are combined: the term-weighting scheme from the *Okapi* system [20], the rule-based text classifier *Ripper* as well as another "rule" learning method based on Support Vector Machines. The experiments presented in [19] show that query-based document retrieval is particularly efficient for processing large document collections.

Compared to crawling-based approaches, however, query-based methods often face the problem of relatively low recall values, since these methods in many cases fail to learn the relevance functions used by the search engines. The main reasons for this are both the complexity of these relevance functions and the fact that many search engines limit the number of allowed queries for a client. In theory, better results could however be achieved by exploiting special features of an individual search engine such as the possibility to limit the search to a particular subarea. For a further discussion and comparison of searching and crawling approaches, see, e.g. [21].

As will be described in the following sections, the xCrawl system presented here picks up on some of the ideas of these existing methods and additionally introduces new techniques in order to achieve high recall values. During the discussion of an example and the technical details of the approach in the next sections, we will also describe the similarities and differences between xCrawl and the above-mentioned techniques.

3 Method overview and example

The principal idea of our method is based on the assumption that websites are organized in a way that allows data to be easily accessed by human users, i.e. almost all websites provide simple means of information access, both through *navigational structures* such as hierarchies, site maps or lists and through *search* functionalities. This idea of exploiting these navigational features of websites for IE is not new. "Hidden web"-oriented

approaches to IE for instance – like the one presented in [22] – aim to locate and use the search interfaces of websites in order to extract data from the underlying databases. "Query probing" is one of the central and critical tasks in such crawling approaches and in many cases the query generation process is based on the usage of a lexical database such as WordNet.

However, when the domain in which the knowledge is to be extracted is rather specific or technical (like the target domains of the AllRight project [1]), a specific vocabulary is required for each domain in order to achieve high recall values as the specific terms of such domains are generally hard to learn in an automated way. Therefore xCrawl follows an IE approach that exploits navigational structures of websites and does not rely on manually-engineered or learned domain vocabularies.

Figure 2 sketches the main steps and rationale of the xCrawl method. In the first phase, xCrawl uses a set of sample target documents provided by the user as input to derive a list of suitable keywords which are ordered according to their TF-IDF (Term Frequency - Inverse Document Frequency) weight[2]. The list of keywords is then used in the initial AQG phase which constructs search engine queries for the retrieval of an initial set of authorities, i.e. relevant documents (Step 1).

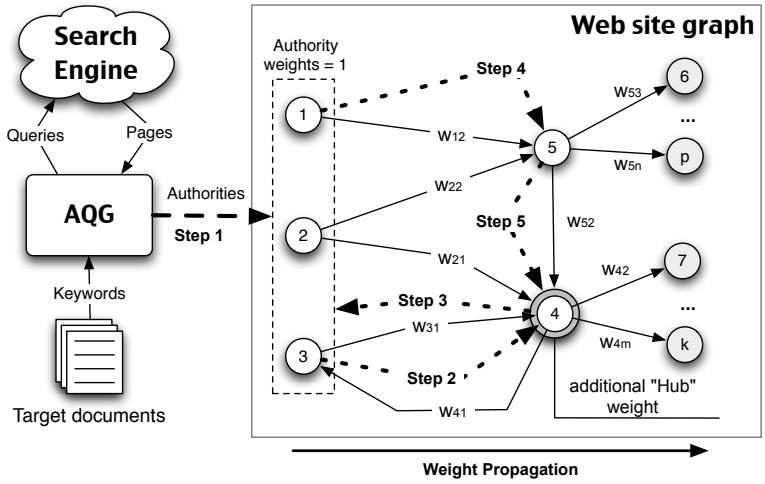

Fig. 2. Overview of the crawling method / Example.

The process of determining whether a page returned by the search engine is an authority or not is carried out by a validator component (omitted in Figure 2) which can

[2] Note that of course this list of keywords could also be directly provided by the user.

in principle be implemented by any of the above-mentioned validation techniques, i.e. xCrawl algorithm does not require specific, additional feedback from a certain validation technique.

After the AQG phase, the actual crawling process follows, which is guided by assigning weights to the outgoing links of the previously discovered pages in the web graph. Link weighting is based on a combination of "hub weights" and "authority weights" as produced by the HITS algorithm and the topic-sensitive PageRank metric. The desired effect of the weighting scheme is that the crawler is directed toward the hubs, which correspond to navigational structures such as index pages, and which directly lead to authorities.

The weights assigned to sets of hubs and authorities are computed using an iterative algorithm that approximates the optimal solution. Subsequent experiments showed that the computation time needed by such an algorithm is acceptable when applied to real-world websites (see Section 5 for details).

For the example, let us assume that Node 4 is such an index page (hub) that points to a set of relevant pages in the lower right corner of the figure. At the beginning of the process, initial weights are assigned to all authorities and their outgoing links. xCrawl starts by randomly selecting one authority (Node 3 in the example in Figure 2) and one of its outgoing links. Note that during the normal crawling process, the links weights are continuously updated based on PageRank and the crawler follows the link leading to the node with the highest weight. If there are several nodes with equal weights – which also happens here at the beginning of the crawling process – the crawler randomly selects one of them.

The crawler may for instance decide to follow link w_{31} in Step 2 of the example in Figure 2. It downloads the target page (Node 4) and forwards it to the validator component. Since Node 4 is not an authority in this example, it is rejected by the validator. Note that in order to prevent the system from getting lost deep within the web graph, xCrawl implements a "Random Restart" strategy with a given probability. In the example, this incidentally happens after Node 4 has been analyzed. After this restart (Step 3 in the example), the HITS algorithm is applied to the known web graph. In the example, Node 4 is identified as a content hub and assigned a corresponding *hub weight*. Based on this information, xCrawl updates the weights of the links connected to the hub within the web graph. After another restart, the algorithm again randomly selects an authority and follows one of the outgoing links. In this iteration (Step 4), Node 5 is selected and analyzed. Node 5 has three outgoing links which the crawler could now follow. However, it can be easily observed in the example that Node 4 is referenced by three nodes and also has an additional hub weight. After calculating the PageRank metric and after taking the weights produced by HITS into account, xCrawl will therefore decide to follow link

w_{52} to Node 4. Thus, the algorithm has reached the goal of guiding the crawler to the index page. When the crawler proceeds, it will choose one of the outgoing links of Node 4 and find the desired authorities (Node 7 to Node k).

Note that the way in which we use the HITS algorithm for finding hubs in a web graph is inspired by the work presented by Kleinberg et al. in [23], where the goal is to automatically identify "cyber-communities". Technically, the problem consists of finding a bipartite subgraph $G' = (V', E')$ of the web graph G, where the elements of V' can be split into two disjoint sets V'_1 (authorities) and V'_2 (hubs) such that the edges from E' only connect nodes from different sets. On the Web, however, such a clear distinction is not always possible, since authorities and hubs may also be interlinked. The popular customer review website `dpreview.com` for instance has a section that lists all digital camera models released in a certain year and therefore is a hub in the web graph. Since the page also contains links to the camera listings of previous years (which are also hubs), the web graph is no longer bipartite. The authority and the hub weights produced by the HITS method, however, help us to approximate the bipartite graph and detect the hub nodes.

4 Algorithm details

Figure 3 summarizes the xCrawl algorithm in pseudo-code. The input parameters that should be provided by the user include two parameters that control the crawling process (see Section 5 for details), a set of sample documents that describe instances of the WMS application domain and a validation component capable of distinguishing between relevant and irrelevant pages.

GetTokens: In the first step, a set of tokens is extracted from the set of target instance descriptions. These documents are tokenized and stop words like articles, prepositions and so forth are removed. Then TF-IDF weights are calculated for each token and GetTokens finally returns a token list ordered by weight. Note that the algorithm can also accept tokens already preselected by the user. In this case, this step may be fully omitted and instead xCrawl will automatically associate weights of 1 with each of the tokens. These weights are subsequently updated automatically later on.

GetAuthorities: This method implements the AQG aspect of xCrawl, which – in combination with the subsequent crawling step – leads to an important improvement over existing approaches as discussed in Section 3.

In order to optimize the results of the AQG step and to generate queries that will hopefully return only relevant pages, xCrawl first associates search weights with each of the extracted tokens individually. Each weight is defined as a normalized number of hits h, where $h \in [0, 1]$, returned by the search engine in response to a query that contains

```
function xCrawl returns a collection of retrieved documents
  inputs: Documents, a set of WMS target documents
      restartProbability, (number) a restart threshold of the crawler
      Validator, a component that determines if a page describes
          an instance of the WMS domain
       maxIterations, (number) a convergency threshold of the crawler
  local variables (initially empty):
      WebGraph, an object that contains the graph for the crawled part of the Web
      Hubs, a collection of pages that are linked with many authorities
      Authorities, a collection of WMS target documents
  Tokens := GetTokens( Documents );
  Authorities := GetAuthorities( Tokens, Validator, maxIterations )
  WebGraph := WebGraph ∪ Authorities
  Sites := GetWebSites( Authorities );
  for each ( site ∈ Sites ) {
    do {
      Page := SelectRandom( Authorities, site )
      repeat {
        Weights := CalculateWeights( WebGraph, Authorities, Hubs );
        Link := SelectLink( Page, WebGraph, Weights );
        Page := GetPage( Link );
        WebGraph := WebGraph ∪ Page;
        Authorities := Authorities ∪ Validator. AnalyzePage( Page );
      } until (Restart ( restartProbability ))
      Hubs := Hubs ∪ FindHubs( WebGraph, Authorities, Hubs )
    } while (noNewHubsFound( Hubs, maxIterations ))
  }
  for each ( hub ∈ Hubs ) {
    Authorities := Authorities ∪ DownloadAllAuthorities( Validator, hub )
  }
  return StoreDocuments( Authorities )
```

Fig. 3. Pseudo-code of the xCrawl algorithm

only this individual token. If a token is very general, it will be found on many web pages and thus the assigned weight will be close to 1. Correspondingly, very specific tokens obtain weights which are close to 0.

xCrawl then combines the search weights with the TF-IDF weights computed in the previous GetTokens step. The combined weights are calculated as a harmonic mean (H-Mean) of the TF-IDF and Search Weight (see Table 1). The harmonic mean was chosen because every weight value should contribute equally to the resulting value.

Token	resolution	review	sensor	specifications	zoom	lens	hot-shoe
Hits (10^3)	275000	956000	85500	142000	20100	84600	992
Search weight	0,288	1,000	0,089	0,149	0,021	0,088	0,001
TF-IDF	0,894	0,090	0,694	0,090	0,745	0,014	0,456
H-Mean	0,435	0,165	0,158	0,112	0,041	0,024	0,002

Table 1. Search weights combined with TF-IDF weights before the AQG step

Next, xCrawl's query generation algorithm starts from the middle of the list, which is ordered by H-Mean and proceeds in both directions simultaneously starting with the central element (in ascending order of distance). This ensures that very specific or popular terms are not used in the construction of queries. The AQG method tests all possible combinations of tokens starting with the center elements. For practical reasons, the maximum number of tokens in these generated queries can be set to a fixed number depending on the constraints of the given search engine. The AQG phase in xCrawl continues until no new authorities are found within a maximum number of iterations as defined by the *maxIterations* input parameter.

The generated queries are sent to a search engine and the first n URLs are retrieved. Assuming that the system is creating queries of length three within the example domain of digital cameras (see Table 1) the first query to be constructed would be "specifications sensor zoom", in our example scenario where "specifications" is the central element of the H-Means ordered list and "sensor" is the nearest neighbor followed by "zoom". At the time of the evaluation, the first $n = 10$ results returned by Google for this query actually included six pages with digital camera specifications, which were accepted by the AllRight WMS validator.

Subsequently, the authorities and their outgoing links are returned as the result of the AQG step and added to the web graph. Each website containing authorities is then analyzed separately. The list of website homepages (base URLs) is extracted from the authorities list using the GetWebSites method.

For each site $\in$ **Sites**: This block implements the main focused crawler functionality of xCrawl. Its main loop has two principal components, *website traversal* and *hub identification*. The first part is implemented as a "focused walk" over the web graph.

starting from a randomly selected authority of the analyzed website (SELECTRANDOM) and following links until a restart event occurs. This event is generated by the RESTART method which uses a random number generator and the associated *restartProbability*.

The web graph traversal is guided by the *edge weights*. The weight of a directed edge $weight(e(v_i, v_j))$ starting at the node v_i is computed as

$$weight(e(v_i, v_j)) = weight(v_i)/outdegree(v_i)$$

where $weight(v_i)$ is the weight of a node v_i and $outdegree(v_i)$ is the number of outgoing edges of v_i. The weight of a node can be updated by two algorithms: HITS and topic-sensitive PageRank.

The link weights are propagated from known authorities over the web graph based on the topic-sensitive PageRank [6] algorithm. During the initialization phase the highest weight of "1" is assigned to all nodes that correspond to known authorities, i.e. to the pages that were accepted by the validator. Weights of all other nodes are assigned according to the formula:

$$weight(v_i) = \begin{cases} weight(v_i), & \text{if } weight(v_i) > 0; \\ 1/n, & \text{if } weight(v_i) = 0. \end{cases} \quad (1)$$

where n is the number of pages in the web graph.

After this initialization phase, the weights are propagated over the web graph as specified by the PageRank algorithm. The weight of a node v_i with respect to a set of links L_{v_i} such that v_i is an end node of each edge $e_j \in L_{v_i}$ with a damping factor c is defined as follows:

$$weight(v_i) = c \sum_{e \in L_u} weight(e)$$

The CALCULATEWEIGHTS method for weight propagation is repeatedly applied until the weights stabilize within some predefined threshold or until the maximal number of iterations is reached. In the algorithm description in Figure 3, this behavior is implemented in the innermost loops, the RESTART and NONEWHUBSFOUND functions.

The edge weights for all nodes are calculated and are used to select – during the crawling process – the link that most probably points to the next most valuable page to be visited. This selection is done within the SELECTLINK method. The selected link is then followed by the crawler (GETPAGE method). Next, the new page is analyzed and the new node together with its corresponding outgoing edges are added to the web graph. In the final step of the traversal loop, the page is validated and is added to the list of authorities if the validator accepts it.

The second part of the crawler – hub identification – is executed whenever a restart event disrupts the traversal of the web graph. Within the FINDHUBS method, the HITS

algorithm [5] is used to compute approximations of bipartite cores. The main idea and assumptions of the algorithm are as follows: a "good" authority will be linked to "good" hubs and a "good" hub points to many authorities thus mutually reinforcing one another. Note that xCrawl – by applying the supplied validator – can differentiate between authoritative and other pages. Consequently, if a page is linked to pages that are accepted by the validator (authorities) it is assigned a higher weight because it is identified as an important part of a navigational structure.

The HITS algorithm differentiates between the *weights of authorities* v_i^a and the *weights of hubs* v_i^h. Initially, weights of all identified authorities are set to $v_i^a = 1$ and $v_i^h = 0$. For all other nodes $v_i^a = 0$ and v_i^h are set according to Formula 1. These weights are updated with each application of the HITS algorithm by applying two operators H and A, which are defined for nodes v_j and a set N of all nodes that are direct predecessors of v_j as follows:

$$H : v_j^h = \sum_{v_i \in N} v_i^a$$

$$A : v_j^a = \sum_{v_i \in N} v_i^h$$

After each iteration the obtained weights are normalized and the algorithm continues the reinforcement of node weights until a predefined number of iterations is reached or no significant weight changes occur. The hub weights are stored in the web graph as weights of all nodes except those accepted by the validator. The HITS authority weights are of no further relevance and are overwritten by the outcome of the validation component.

If no new hubs are identified for a website within the predefined number of iterations (*maxIterations*), the crawling algorithm terminates. Note that in each iteration the weights calculated by the HITS algorithm are used for further improvement of the future focused crawler navigation in the web graph corresponding to a reinforcement learning approach.

For each hub ∈ Hubs: Given all the hubs from the focused crawling step, the algorithm analyzes the DOM paths of all links pointing to known authorities and determines their DOM parent elements. xCrawl then downloads all the pages published on the same website and which are referenced from the identified sub-tree of each hub. All downloaded pages are then analyzed (DownloadAllAuthorities). If new authorities are found, i.e. a page was accepted by the WMS validator, they are stored in the corresponding list. This list is saved and returned as the result of xCrawl (method StoreDocuments).

5 Experimental evaluation

The main objectives of our evaluation were (a) to measure the recall values that the new crawling method achieves in a given time frame, (b) to compare these results with the

recall values of a baseline crawler and (c) to make suggestions for the xCrawl input parameters "restart probability" and "number of iterations".

5.1 Experiment setup

Two different web mining experiments were selected that reflect common scenarios faced by modern WMSs. In the first one, the goal is to automatically find and extract product data from highly-structured web sources, i.e. the task of the crawler is to find as many web pages as possible that contain usable product specifications published in tabular form. This scenario is also the driving scenario of the AllRight project [1], in the context of which xCrawl has been evaluated. The test application domains for the baseline crawler were *digital cameras* and *laptop computers*.

In the second scenario, the goal is to find *text reviews* for certain products written in natural language. The application domains were *MP3 players* and *cell phones*. This second scenario was specifically selected in order to demonstrate that xCrawl is not limited to the specific web mining problems for which it was originally developed.

Correspondingly, different types of validators were used in the two scenarios. In the first case the standard validator of the AllRight system was utilized which is able to recognize if a page contains a *tabular presentation* of a target instance [1]. In the second scenario we used a Bayesian Network (BN) classifier implemented in the WEKA framework [24], which was trained to recognize *text reviews* of the products.

Also for the sake of making the different approaches comparable, the standard AllRight workflow – in which a knowledge acquisition component automatically supplies the crawler with keywords defined in a user ontology – was not followed during the experiments. Instead, twenty valid sample documents for each domain were manually downloaded and used as seed data for the crawler. In particular, these seed documents were required for training the BN classifier of the baseline approach before the crawling process. The obtained relevance score distributions were strongly bimodal and similar to those reported in [14]. Note also that the evaluation within the experiments was carried out for very specific topics further simplifying the classifier training phase.

The table recognition algorithm used for the first scenario does not need any training samples since it uses a number of heuristics to locate and extract a table from a page. Instead, the method relies on the correct rendering of the page by the crawler's web browser as the position of text-boxes plays a crucial role. Consequently a modern browser that can correctly render any web page was used in the experiments. A short discussion of the basic technology used follows in Section 6. Details of the table recognition algorithm are given in [25].

We considered two approaches to focused crawling as a baseline for our evaluation: the Context-Focused Crawler (CFC) from [15] and a crawler similar to one developed by

Chakrabarti et al. [14]. CFC is very similar to our approach since it takes the structure of a web site into account and thus seems to be a natural baseline for xCrawl. However, in our experiments CFC failed to construct the needed "context graphs", because the search engines used (Google and Yahoo!) did not return a sufficient number of pages that were linked to the initial authorities found by the AQG method. Google's search engine, for instance, did not return any of the at least three pages published on `www.dpreview.com` that contained a link to the Canon A650is page that were identified by the AQG method as authorities. Therefore we only compare our results to the results obtained using the combination of a main validator and an apprentice similar to the method described in [5].

In each experiment, the crawlers were limited to the list of six top websites identified by the xCrawl AQG strategy. This limitation was required to subsequently measure the achieved *recall* values of each crawler. In addition, in contrast to the xCrawl method, the baseline BN crawler has no built-in capability to identify such target websites effectively. If, for instance, an important website is not listed in a directory such as `dmoz.org`, it will not be considered by the BN-based crawler. The website `imaging-resource.com` for example contains more than 790 specifications for digital cameras but is not listed in the corresponding `dmoz.org` category[3]. Still, this site is highly ranked by Google, which means that it will most probably be found by xCrawl's AQG component. Thus, in order to provide equal opportunities for both tested crawling approaches, the baseline focused crawler was configured to start from a pool of pre-defined websites which were selected by applying xCrawl's AQG phase once.

Note that in our implementation the apprentice was trained only on one of a number of training sources suggested in [14], i.e. on Document Object Model features. These features are represented as pairs $\langle d, t \rangle$, where t is a token and $|d| \leq 5$ is the distance from the DOM element that contains that token to the DOM element that represents the link (see [14] for more details). The restriction for the apprentice described above is based on the requirement of designing a crawler which is independent of the WMS's validation technique. Therefore in our implementation the crawler's URL selection method cannot rely on any special feedback from the validation component, such as a vector of class probabilities computed by the BN validator. The restriction was introduced with the aim of developing and comparing crawling methods for general web mining problems and thus treats the (potentially domain and problem-specific) validator of the WMS as a black-box that is only able to return "accepted" or "not accepted".

[3] Home/Consumer_Information/Electronics/Photography/Digital_ Cameras

5.2 Measurement method and results

In order to calculate recall, we manually analyzed how many target instances were actually published on the selected websites. In cases where this information was not explicitly available, the approximate number of instances was counted by browsing the corresponding sections of the websites manually. The website `steves-digicams.com` for instance at the moment of our analysis contained descriptions of 686 cameras (see Figure 4). We started the evaluation with tests of xCrawl, which algorithm converged after approximately six hours for the first domain of digital cameras. Next, the baseline crawler was also executed for the same period of time. Finally, the number of instances found was compared with the actual number. The detailed numbers for the individual websites are shown in Figures 4 to 7.

Website	Found by xCrawl	Actually existing	Found by baseline
reviews.cnet.com	643	670	6
imaging-resource.com	794	894	72
dcresource.com	372	432	175
parkcameras.com	179	191	181
steves-digicams.com	634	686	479
dpreview.com	825	845	625

Fig. 4. Digital camera domain (Tabular sources)

Website	Found by xCrawl	Actually existing	Found by baseline
digitaltrends.com	217	255	46
www.mp3.com	134	150	82
www.pixmania.co.uk	195	227	112
www.pcmag.com	186	214	149
www.reviewcentre.com	183	196	195
reviews.cnet.com	648	713	350

Fig. 5. MP3 player domain (Text sources)

The results of the evaluation, i.e. the recall values achieved using each method, are shown in Figure 8. xCrawl performed significantly better in every domain that was

Website	Found by xCrawl	Actually existing	Found by baseline
dabs.com	289	302	57
newegg.com	493	507	58
datavis.com	96	108	84
europc.co.uk	86	89	88
ww2.inoax.com	612	626	142
tabletpc.alege.net	793	851	148

Fig. 6. Laptop computer domain (Tabular sources)

Website	Found by xCrawl	Actually existing	Found by baseline
laptopmag.com	82	86	46
www.mobiledia.com	132	132	75
mobile-phones-uk.org.uk	74	88	80
pcmag.com	365	389	153
reviews.cnet.com	503	554	158
www.mobile-review.com	524	524	345

Fig. 7. Cell phone domain (Text sources)

tested and major average recall improvements, of approximately 42%, were achieved with respect to the baseline crawler.

In order to test the impact of the different proximity measures used for crawling, we also evaluated the algorithm using (a) only the weights obtained from the HITS algorithm and skipping the PageRank calculation phase and (b) HITS in combination with "standard" PageRank instead of topic-sensitive PageRank.

The results for these experiments are also shown in Figure 8. In all domains, xCrawl using HITS in combination with topic-sensitive PageRank worked best. For the domains with tabular data (digital cameras and laptop computers), xCrawl using only HITS weights in the CalculateWeights method also outperformed the baseline crawler. This can be explained by the fact that in these domains fewer pages exist which contain text that can be used by the baseline method to train its apprentice and, thus, to learn its proximity measure. In the domains with textual data, both crawlers, i.e. the baseline crawler and xCrawl using only HITS, performed with comparable recall. A version of xCrawl with a proximity measure calculated as a harmonic mean of HITS and standard PageRank values performed on average 31% better than the baseline method.

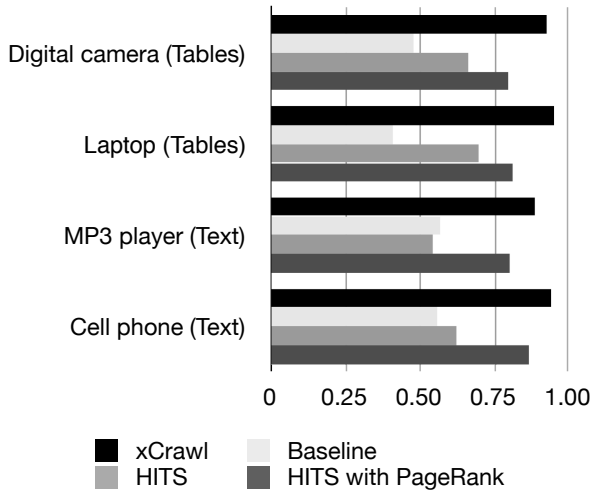

Fig. 8. Comparison of average recall

When analyzing the individual website results in more detail, we can observe that the recall of the baseline method was considerably lower when applied to large websites with a number of different subsections. The reason for this limited performance can be found in the fact that these sections usually include many structurally identical HTML pages, which however present specifications of different product types. Consider for instance the fact that websites containing digital camera specifications in many cases also have a section on photo printers and that many printer manufacturers are also producers of cameras. Subsequently, with the baseline approach, the apprentice assigns higher probability scores to tokens that correspond to manufacturer names than to all other tokens as review sites often arrange lists of products in such a way that links to product specifications are surrounded by the names of other products of the same manufacturer within the same DOM sub-tree. In such cases, the apprentice, which was trained using several successful retrievals at the beginning of the crawling process, often misleads the focused crawler which then becomes stuck in the wrong section of the website. Therefore, the results obtained for the digital camera domain for review websites like `reviews.cnet.com` or `digitalcameras.alege.net` are unsatisfactory. xCrawl, on the other hand, was able to return to the correct section of the website by utilizing its restart strategy and the bottom-up crawling process.

When analyzing the xCrawl running times in detail, we could furthermore observe that the actual computation time required for applying HITS and managing the web graph only took a relatively small fraction of the overall running time. The most time-

consuming activities were the browsing-related tasks such downloading pages or rendering pages internally, which on average consumed nearly three of the six hours of running time. The details of this analysis are shown in Figure 9.

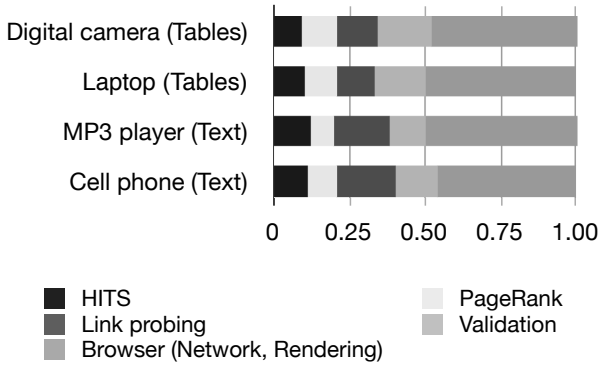

Fig. 9. Percentage of time consumed by main xCrawl tasks

5.3 xCrawl parametrization

We evaluated the xCrawl method with different input parameters in order to give some recommendations on their selection. In the experiments, the *number of iterations* was first set to a relatively high value to allow the crawler to be executed with different values for the restart probability. The Harvest rate, defined as $HR = |C|/|P|$, where C is a set of pages accepted by the validator and P is a set of all pages retrieved by the crawler, was used as the evaluation measure.

The selection of the *restart probability* value was based on the observation that the relevance of two web pages decreases considerably when the distance between them on the web graph increases [12]. We started with a restart probability of 0.05 and iteratively increased it up to 0.3 until substantial decrease in the harvest rate was observed (see Figure 10). The best performance in all evaluation domains was observed with a restart probability of 0.15, see Figure 10.

The *number of iterations* parameter values were then evaluated with this best-known probability value of 0.15. The experiments showed that the number of iterations could be reduced to a value of around 50 without significantly affecting harvest rates.

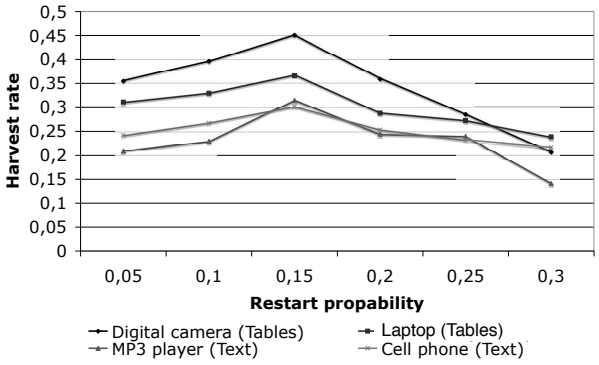

Fig. 10. Harvest rate dependency on the restart probability

6 Browsing component details and AllRight WMS overview

In this section, we will give some technical details on the browsing component used in our xCrawl implementation and sketch the overall workflow and structure of the AllRight Web Mining System, in the context of which xCrawl has been developed.

6.1 Browsing modern web pages

As mentioned above, crawling contemporary websites is not a trivial task since most of them are complex applications built with modern scripting technologies. The wide adoption of navigation menus, sidebars, pictures that act as buttons and so forth reduces the efficiency of many web crawlers because these elements "hide" the real structure of the web graph. Moreover, links are often constructed on the fly and thus cannot be extracted just by scanning the source files. Hence, the pages that can be reached by a human just in one or two clicks using the menus may be unreachable by solely considering normal HTML anchors or by analyzing the Document Object Model. Therefore, additional techniques have to be used to simulate the activities of a website user. Moreover, the adopted browsing technology should support all modern standards in order to be able to render a web page correctly. This requirement is essential, since some validation techniques – like table recognition methods [25] – require that the document is rendered to the crawling system in the same way that it is presented to a user.

Our xCrawl implementation thus includes a component capable of handling these modern techniques using *XULRunner*[4], which is the runtime part of the Firefox[5] browser,

[4] http://developer.mozilla.org/en/docs/XULRunner
[5] http://en.www.mozilla.com/en/firefox/

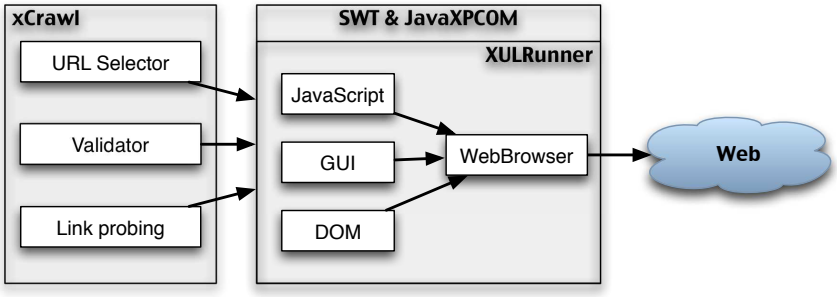

Fig. 11. General architecture of xCrawl's web browser implementation

as a main component (see Figure 11). XULRunner can be used on the same platforms as Firefox itself. In addition, SWT[6] and JavaXPCOM[7] were used to access the web browser's functionality from the Java programming language, in which the xCrawl system is built. In order to ensure a correct behavior of the crawler, a number of services of the Graphical User Interface (GUI) – such as alert-box notification or printing – were replaced to prevent the system from getting stuck on pages that require user interaction and which are not relevant to the crawling task. Moreover, a link probing technique was added to correctly find all clickable objects on the page and simulate the user interaction with them. Thus, by accessing the *JavaScript* and *DOM* functionality of XULRunner, xCrawl is able to open menus or other dynamically created DOM elements and probe actions such as "click" or "mouseover". The system captures all the navigation events and web page states that are generated by the *WebBrowser* component in response to simulated user interaction.

In addition, xCrawl's web browser can effectively prevent the redirection of the browser to other pages potentially caused by probing actions, hence performing a depth-first link search over all possible states of a dynamic HTML page. Finally, all URLs identified by the browsing component are added to the web graph, thus creating a better approximation of the actual website graph. The different components of xCrawl's web browsing component and its integration with XULRunner are shown in Figure 11.

The only major limitation of this technique is that it cannot simulate user input into text fields. Thus it can fail to extract all links within some modern Ajax-based web applications. Note that in general, the problem of crawling and indexing Ajax-based websites is still largely unsolved. For a recent approach toward effective crawling of such web documents, see for example the work of Mesbah et al. [26], who propose inferring a

[6] http://www.eclipse.org/swt/
[7] http://developer.mozilla.org/en/docs/JavaXPCOM

"state-flow-graph" from an AJAX application and reconstructing the set of linked static pages.

6.2 The AllRight Web Mining System

The proposed XCRAWL method is part of the larger ALLRIGHT web mining and ontology instantiation system as shown in Figure 12. The development of this system was inspired by the need for an automatically acquired database of detailed item descriptions for use in the knowledge-based recommender system ADVISOR SUITE [27]. The overall process flow shown in Figure 12 can be summarized as follows.

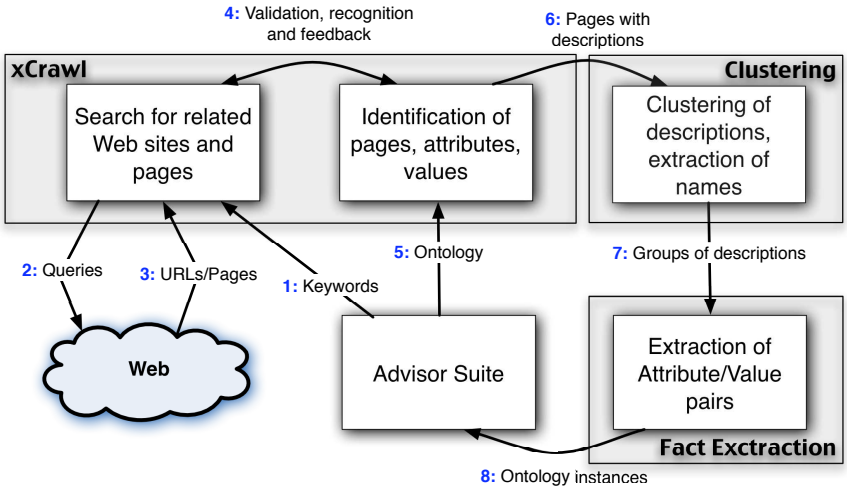

Fig. 12. Overview of the ALLRIGHT WMS [1]

The seed knowledge for the information retrieval process consists of a *domain ontology*, which is modeled using the graphical knowledge acquisition user interface of the ADVISOR SUITE system. This ontology describes the target items of the web mining process. In our digital camera example, this ontology would also contain a list of names of camera characteristics such as "resolution" or "sensor size" and a description of the possible values for these camera features. From this ontology, a set of keywords is derived and used as an input to the AGQ phase of XCRAWL, see Step 1 in Figure 12. The result of the crawling and validation phases (Steps 2 to 6) described in this paper is a set of validated pages containing target item descriptions. In order to exploit the redundancy of the information on the web, the documents are then clustered in a way such that each group only contains descriptions of the same (camera) instance. The subsequent

fact extraction process is based on a novel instance name recognition and query-based fact validation algorithm, see also [28]. The outcome of this process (Step 7) is a set of target document, i.e. detailed item descriptions that are fed back into the recommender system. An evaluation in different domains showed an overall web mining accuracy of over 80% [1], given only a limited amount of seed knowledge.

7 Conclusion

In this paper we have presented a new crawling technique whose goal is to increase the "recall" of a WMS's information retrieval component, which is of particular importance for WMSs that take advantage of the redundancy of data published on the Web. This new method is based on a combination of *search* and *focused crawling* and the exploitation of navigational patterns in web graphs. A detailed evaluation of our approach in four popular domains indicates that it outperforms state-of-the-art focused crawling techniques and finds a higher of number relevant documents from a domain within a fixed period of time.

Our future work in the context of xCrawl will in particular focus on the development and the evaluation of additional and novel proximity measures that can benefit from the information obtained from the HITS algorithm. Current approaches commonly utilize learning algorithms such as Bayesian Networks or Support Vector Machines to approximate a proximity measure, and predict whether a link points to an authority or not. In our future research, we will investigate whether the given hub weights in combination with classification methods can help to further improve the performance of xCrawl.

Furthermore, the evaluation of xCrawl suggests that – at least in some application domains – the consideration of text fragments surrounding the links in the proximity measure can positively influence the effectiveness of the crawling method. Therefore, we plan to develop a method that can automatically adapt to the different data presentation styles used on modern websites.

Acknowledgments

We would like to thank David Wieser, who implemented parts of the xCrawl algorithm and the anonymous reviewers for their valuable comments. The research project is partially funded by grants from the Austrian Research Promotion Agency (Program Line FIT-IT Semantic Systems Project AllRight, Contract 809261) and by grants from the Austrian Science Fund (Project V-Know, contract 19996).

References

1. K. Shchekotykhin, D. Jannach, G. Friedrich, and O. Kozeruk, "AllRight: Automatic ontology instantiation from tabular web documents," in *Proceedings of 6th International Semantic Web Conference*, Busan, Korea, 2007, pp. 466–479.
2. S. Dill, J. Tomlin, J. Zien, N. Eiron, D. Gibson, D. Gruhl, R. Guha, A. Jhingran, T. Kanungo, S. Rajagopalan, A. Tomkins, J. Tomlin, and J. Y. Zien, "SemTag and Seeker: bootstrapping the Semantic Web via automated semantic annotation," in *Proceedings of the 12th International World Wide Web Conference WWW2003*, Budapest, Hungary, 2003, pp. 178–186.
3. A. Kruger, C. L. Giles, F. Coetzee, E. Glover, G. Flake, S. Lawrence, and C. Omlin, "DEADLINER: Building a new niche search engine," in *Proceedings of 9th International Conference on Information and Knowledge Management, CIKM 2000*, Washington, DC, November 2000, pp. 272–281.
4. M. Craven, D. DiPasquo, D. Freitag, A. McCallum, T. Mitchell, K. Nigam, and S. Slattery, "Learning to construct knowledge bases from the World Wide Web," *Artificial Intelligence*, vol. 118, no. 1, pp. 69–113, 2000.
5. J. Kleinberg, "Authoritative sources in a hyperlinked environment," *Journal of the ACM (JACM)*, vol. 46, no. 5, pp. 604–632, 1999.
6. T. H. Haveliwala, "Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search," *IEEE Transactions on Knowledge and Data Engineering*, vol. 15, no. 4, pp. 784–796, 2003.
7. A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins, "The discoverability of the web," in *Proceedings of the 16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007, pp. 421–430.
8. U. Schonfeld, Z. Bar-Yossef, and I. Keidar, "Do not crawl in the DUST: different URLs with similar text," in *Proceedings of the 15th International World Wide Web Conference*, New York, NY, USA, 2006, pp. 1015–1016.
9. J. Cho, H. Garcia-Molina, and L. Page, "Efficient Crawling Through URL Ordering," *Computer Networks and ISDN Systems*, vol. 30, no. 1-7, pp. 161–172, 1998.
10. S. Chakrabarti, *Mining the Web: Discovering Knowledge from HyperText Data.* Science & Technology Books, 2002.
11. H. Yu, J. Han, and K. C.-C. Chang, "PEBL: positive example based learning for web page classification using SVM," in *KDD '02: Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Edmonton, Alberta, Canada, 2002, pp. 239–248.
12. S. Chakrabarti, M. van den Berg, and B. Dom, "Focused crawling: a new approach to topic-specific web resource discovery," *Computer Networks*, vol. 31, no. 11-16, pp. 1623–1640, 1999.
13. C. C. Aggarwal, F. Al-Garawi, and P. S. Yu, "Intelligent crawling on the world wide web with arbitrary predicates," in *Proceedings of the 10th International World Wide Web Conference*, New York, NY, USA, 2001, pp. 96–105.

14. S. Chakrabarti, K. Punera, and M. Subramanyam, "Accelerated focused crawling through online relevance feedback," in *Proceedings of the 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, 2002, pp. 148–159.
15. M. Diligenti, F. Coetzee, S. Lawrence, C. L. Giles, and M. Gori, "Focused crawling using context graphs," in *Proceedings of 26th International Conference on Very Large Data Bases*, Cairo, Egypt, 2000, pp. 527–534.
16. J. Rennie and A. McCallum, "Using reinforcement learning to spider the web efficiently," in *Proceedings of the Sixteenth International Conference on Machine Learning*, San Francisco, CA, USA, 1999, pp. 335–343.
17. F. Menczer, G. Pant, P. Srinivasan, and M. E. Ruiz, "Evaluating topic-driven web crawlers," in *SIGIR '01: Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*, New Orleans, Louisiana, United States, 2001, pp. 241–249.
18. M. Ester, H.-P. Kriegel, and M. Schubert, "Accurate and efficient crawling for relevant websites," in *VLDB '04: Proceedings of the Thirtieth International Conference on Very Large Data Bases*, Toronto, Canada, 2004, pp. 396–407.
19. E. Agichtein and L. Gravano, "Querying text databases for efficient information extraction," in *Proceedings of the 19th IEEE International Conference on Data Engineering*, Bangalore, India, 2003, pp. 113–124.
20. S. E. Robertson, "On term selection for query expansion," *Journal of Documentation*, vol. 46, no. 4, pp. 359–364, 1990.
21. P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano, "To search or to crawl? Towards a query optimizer for text-centric tasks," in *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, Chicago, IL, USA, 2006, pp. 265–276.
22. A. Bergholz and B. Chidlovskii, "Crawling for domain-specific hidden web resources," in *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, Washington, DC, USA, 2003, pp. 125–133.
23. J. M. Kleinberg, R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins, "The web as a graph: Measurements, models, and methods," in *Computing and Combinatorics, Springer LNCS 1627*, 1999, pp. 1–17.
24. I. Witten and E. Frank, *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000.
25. W. Gatterbauer, P. Bohunsky, M. Herzog, B. Krüpl, and B. Pollak, "Towards domain-independent information extraction from web tables," in *Proceedings of the 16th International World Wide Web Conference*, Banff, Alberta, Canada, 2007, pp. 71–80.
26. A. Mesbah, E. Bozdag, and A. van Deursen, "Crawling AJAX by inferring user interface state changes," in *Proceedings of the 8th International Conference on Web Engineering*, Yorktown Heights, NY, USA, July 2008, pp. 122–134.
27. A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "An integrated environment for the development of knowledge-based recommender applications," *International Journal of Electronic Commerce*, vol. 11, no. 2, pp. 11–34, Winter 2006-7 2007.

28. K. Shchekotykhin, D. Jannach, and G. Friedrich, "Clustering web documents with tables for information extraction," in *The Fourth International Conference on Knowledge Capture (K-CAP)*, Banff, Canada, October 27-31 2007.