xCrawl: A High-Recall Crawling Method for Web Mining

Kostyantyn Shchekotykhin
University Klagenfurt
9020 Klagenfurt, Austria
kostya@ifit.uni-klu.ac.at

Dietmar Jannach
Technische Universität Dortmund
44227 Dortmund, Germany
dietmar.jannach@tu-dortmund.de

Gerhard Friedrich
University Klagenfurt
9020 Klagenfurt, Austria
gerhard@ifit.uni-klu.ac.at

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 within a limited period of time as many web pages as possible that contain relevant information, 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 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 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 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 & Background

The initial task of Web Mining Systems (WMS) like AllRight [17], WEB$\rightarrow$KB [7] 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. The 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 desired properties are commonly measured in terms of the "precision" and "recall" of the set of collected documents. Today's WMSs, like those mentioned above, are able to validate given documents with a very high precision, i.e. they are good at determining whether a given document contains the desired information or not. In this paper, however, we focus on increasing the "recall" value, i.e. finding more suitable documents on the Web in a shorter period of time using Information Retrieval (IR) methods such as focused crawling and automatic query generation. Before discussing the proposed XCRAWL method we summarize the shortcomings of existing IE data collection approaches.

Crawling. Today, *Web Crawling* is the standard method for retrieving and refreshing document collections [8] within WMSs (as opposed to *searching*, see [12]). In many cases, simple crawlers follow a breadth-first search strategy, starting from the root of a website (homepage) and traversing all URLs in the order in which they were found. This procedure is then applied to all URLs extracted from newly downloaded pages. More sophisticated crawlers, see for instance [16], try to minimize the effort of crawling, for example by identifying and ignoring different URLs that point to the same page.

However, in many web mining scenarios the desired documents are located deep in the navigational hierarchy, requiring a breadth-first crawler to download and subsequently process a large set of documents using a *WMS validator*. Unfortunately, the validation of documents is a time consuming task due to the complexity of modern validation algorithms. Hence the speed of the retrieval process is dramatically reduced when lots of irrelevant documents have to be checked. In addition, the increased number of documents increases the probability that the validation system will accept irrelevant documents, thus reducing the precision of the WMS's output.

Focused crawlers, in contrast to breadth-first crawlers used by search engines, typically use an informed-search strategy and try to retrieve only those parts of the Web relevant to some given topic [1, 5, 9, 15]. The documents retrieved by the web browsers of focused crawlers are validated before they are stored in a repository or database. The links contained in the retrieved pages are added to a *web graph* and specific URL selection algorithms, which exploit the results of previous actions, are used to select the next URL that – with the highest probability – will be accepted by the validator.

Depending on the domain and the mining goals, the validation and information extraction task can be accomplished by different algorithms such as Support Vector Machines [19], Bayesian Networks [7], or table detection techniques [17]. In order to ensure the broad applicability of a focused crawling technique in different scenarios, these algorithms should not depend on a specific implementation or outcome of the validation process.

Searching. Beside crawling, *Automatic Query Generation* (AQG) is another document retrieval approach, which is however not so commonly used in WMSs. AQG works by sending queries to large-scale search engines, thus enabling quick access to documents that are relevant to the current web mining task.

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 should be retrieved and analyzed, allowing more tokens to be extracted and added to the seed tokens. This procedure should be repeated until some stop criterion is met.

The main disadvantage of AQG-based methods is that achieving high recall values is not an easy task because the relevance function, which is used by the search engine for ranking, is typically not known. Hence, the results of AQG-based search will be mainly limited to top ranked documents. Slight improvements with respect to recall can be theoretically achieved by exploiting special features of modern search engines such as the ability to search in particular areas of target websites. Complex queries, however, create a huge load on the search engines and thus search service providers limit the number of queries that can be performed at once, in turn hampering the implementation of such approaches in practice.

1.1 Novel aspects

The goal of our work and the proposed XCRAWL method is to improve the document retrieval process by retrieving more relevant documents in shorter period of time than currently possible with existing techniques. This new method focuses in particular on downloading only a minimal number of documents to be validated as downloading and validation are time consuming tasks.

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. The 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 [3] – 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 [17]), 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.

In order to find all relevant pages (*authorities*) on the website, index pages (*hubs*), which contain many links to relevant pages, are identified. In principle, the crawler tries to identify the set of hubs as completely as possible in order to obtain an exhaustive set of authorities. The main problem within this context is discriminating between hubs and authorities in the web graph. Technically, the central task of the XCRAWL method is thus to find a subgraph $G' = (V', E')$ of the web graph $G = (V, E)$, which is bipartite, i.e. in which the nodes of the set V' can be partitioned into two disjoint sets V'_1 of authorities and V'_2 hubs in a way that edges $e \in E'$ only connect nodes from V'_1 with nodes of V'_2 and vice versa.

Figure 1 shows an example of a website graph that includes a set of authorities, a set of hubs as well as some other pages that do not belong to the sub-graph being located. The approach presented here attempts to partition the graph as described above with the particular goal of detecting the hubs. Subsequently, this knowledge is used to compute weights for the edges in the web graph which guide the crawling process in the right direction, i.e. towards the hubs (which most probably contain links to further authorities).

Unfortunately partitioning in many websites is often impossible because the authorities or hubs are interlinked. On the popular digital camera review site `www.dpreview.com`, for instance, reviews are grouped according to the camera announcement date under the “Timeline” section. Each page of this section corresponds to a year and contains links pointing to camera specifications announced during this year, i.e. functions as a hub in our problem. If a focused crawler that searches for product specifications is able to identify such a “Timeline” page, it will easily find all published specifications thus lifting the recall almost to 1. However each such page also contains links to pages with announcements from previous years (which are also hubs) violating the properties of bipartite graphs. Therefore, a partitioning algorithm has to be used that can detect such links between the nodes of the same set and remove or ignore them. Within the example of Figure 1 the nodes and edges which should be removed or ignored are depicted using dashed lines.

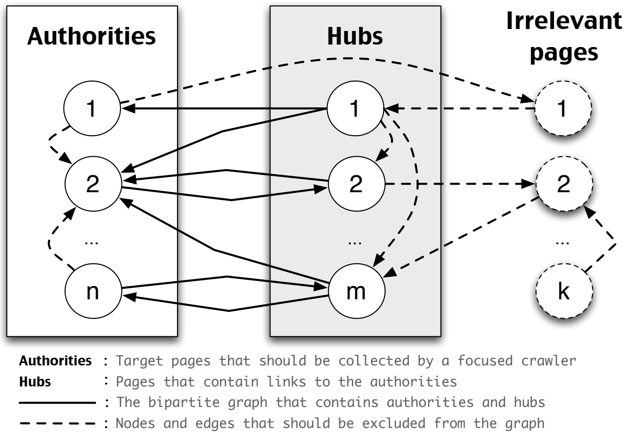
Authorities
Hubs
Irrelevant pages
1
2
...
n
1
2
...
m
1
2
...
k
Authorities : Target pages that should be collected by a focused crawler
Hubs : Pages that contain links to the authorities
: The bipartite graph that contains authorities and hubs
: Nodes and edges that should be excluded from the graph

Figure 1. Example of the web graph partition.

The method presented here uses the HITS algorithm [13] to effectively identify the set of hubs on a web graph given the set of authorities. Moreover, the weights corresponding to these sets can be 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 3.1 for details).

XCRAWL also implements the automatic identification of an initial set of websites that are likely to contain pages with target data, providing an effective start point. Rather than requiring the manual provision of a set of start sites, XCRAWL re-uses existing information which can for instance be retrieved from public search engines or from manually engineered directories like `dmoz.org`.

Furthermore, the retrieval of relevant websites is based on *Automatic Query Generation* [12], i.e. relevant keywords from the domain are used to create queries for public search engines to locate interesting websites.

Overall, the evaluation of this combination of new and existing techniques, which is discussed in Section 3, shows that a significant increase in recall values can be achieved with the new XCRAWL method.

1.2 Previous systems

Existing systems for focused crawling such as those described in [1, 5, 9, 15] usually implement the following steps: locate the relevant resources, learn about the website’s environment that contains links to the desired resources and repeatedly apply the learned environmental patterns to new websites. The method presented in [5] for instance uses two classifiers for learning: a primary classifier that decides on the relevance of documents and an apprentice classifier that predicts the relevance of unseen pages. In their approach, the apprentice acts as a simple reinforcement learner that is trained using features derived from the Document Object Model (DOM) of the retrieved documents and the output of the main classifier. The main classifier’s output is used as feedback to the current choice of apprentice.

Similar to the previous approach, the crawler presented in [15] relies on reinforcement learning. In particular, this approach exploits text fragments in the neighborhood of links or within title and header tags with the goal of incrementally learning whether or not a link is a promising one and should be followed by the crawler.

Our approach overcomes two limitations of these approaches. First, it does not depend on the existence or feedback of a specific validation technique, i.e. different validation algorithms can be used depending on the goal of the WMS. Second, in modern web applications, links and their environments are in many cases created through scripts and used in menus or sidebars. Thus, exploiting only the Document Object Model as well as the surrounding text may be insufficient or even not possible for effective website crawling. XCRAWL can cope with such dynamically generated links by using a browsing client which also executes scripts (see Section 3.2 for details).

Diligenti et al. [9] propose a *Context Focused Crawler* (CFC) that discovers typical link hierarchies (Context Graphs) in which relevant documents appear. CFC uses existing search engines to implement reverse crawling in order to calculate the expected link distance from any document to the target document. In their approach, a set of classifiers is trained to assign different categories to the documents according to their links. Such training produces a massive load on search engines if the websites crawled are not homogeneous, i.e. if they include a variety of sections with different content. In contrast, the number of required search engine queries is significantly lower in our approach, thus it can be used in the cases when the number of queries allowed by search engines is limited. Moreover if a website for some reason is not indexed by a search engine, our approach allows the user to provide links to target pages manually since in this case they cannot be automatically detected by the AQG component.

QXTract [2] is an example of an AQG-based system which uses user defined tuples to extract the characteristics of relevant documents. The system then applies a hybrid technique that combines queries generated with the Okapi query expansion method and the two text classification methods *Ripper* and *SVM*. Despite the fact that this method enables fast access to the target data, it shares the same drawbacks as all query based approaches such as comparably low recall values [12].

Note that existing crawlers have no dedicated means of locating websites on which their targets are published. The main assumption of such crawlers is that pages of one relevant website will include links to other websites from the same domain or that directories (such as `dmoz.org`) exist that contain links to other target websites. The work described in [10], for instance, is based on the first assumption and is implemented as a combination of two focused crawlers: one to discover relevant websites and the other to crawl them. The combination of two crawlers appears to be efficient in terms of *coverage*. However, its running times are rather long in comparison with AQG-based approaches [12]. On the other hand, directory-based approaches rely solely on the information provided by its users. Consequently, some important websites could be missed by such a crawler. The XCRAWL method therefore implements a combination of search and crawling methods. Additionally, an AQG method is used to retrieve up-to-date data from search engines, which in turn leads to much better coverage whilst locating relevant websites in a shorter period of time.

The subsequent section gives a detailed presentation of the crawling technique. Next, we present the results of an evaluation of the new method within several application domains is presented, revealing that a significant increase in recall values can be achieved compared to existing approaches.

2 The xCrawl algorithm

Figure 2 summarizes the XCRAWL algorithm in pseudocode. The input parameters that should be provided by the user include two parameters that control the crawling process (see Section 3 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 given 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 can be fully omitted and XCRAWL associates weights of 1 with each of the tokens automatically. 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 1.

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 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 two other weights, TF-IDF and Search Weight (see Table 1). The harmonic mean was chosen because every weight value should contribute equally to the resulting value.

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 maximal 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 query is 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 were 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

```
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 )
```

Figure 2. Pseudo-code of the xCrawl algorithm

(base URLs) is extracted from the authorities list using the GetWebSites method.

For each site ∈ 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 *link weights*. The weight of a link l pointing to a page v is computed by the formula $linkWeight(l) = weight(v)/\ outdegree(v)$, where $weight(v)$ is a weight of a node v and $outdegree(v)$ is the number of outgoing links of v. The function $weight(v)$ actually combines three different weights: authority weight, hub weight, and PageRank. The overall node weight is defined as the average of all three weights, excluding those with a weight of 0. If, for instance, a node has an authority weight of 1 and all other weights are 0, the overall weight will be 1.

The link weights are propagated from known authorities over the web graph based on the PageRank [4] algorithm, in whose initialization phase the highest weight of "1" is assigned to all known authorities, i.e. to the pages that were accepted by the validator. All other non-hub pages (as identified by the FindHubs method) obtain weights of $1/n$, where n is the number of pages in the web graph. Hub nodes are assigned an average of $1/n$ and its original hub weight.

After this initialization phase, the weights are propagated over the web graph as specified by the PageRank algorithm. The weight of a page u with respect to a set of links L_u pointing to u and a damping factor c is defined as follows:

$$weight(u) = c \sum_{l \in L_u} linkWeight(l)$$

The CalculateWeights method for weight propagation is repeatedly applied until the weights stabilize within some predefined threshold or until the maximal number of

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

iterations is reached. In the algorithm description in Figure 2, this behavior is implemented in the innermost loops and the RESTART and NONEWHUBSFOUND functions.

The link weights for all nodes are calculated and are used to select – during the crawling process – the node that most probably points to a next valuable page to be explored. This selection is done within the SELECTLINK method. The new link is then followed by the crawler (GETPAGE method) and the new page is analyzed and added to the web graph together with all its outgoing links. 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 [13] is used as an efficient method for the identification of bipartite cores (e.g. for the extraction of cyber-communities [14]). The main idea and assumptions of the algorithm are the following: a "good" authority will be linked with "good" hubs and a "good" hub points to many authorities thus mutually reinforcing one another. Note that XCRAWL – by means of the provided validator – can differentiate between authoritative and other pages. Consequently, if a page is linked with pages that are accepted by the validator (authorities) it shall be 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* and the *weights of hubs*. Initially, the weights of each node u of the web graph are set to $1/n$, where n is the number of nodes in the graph. These weights are updated on each application of the HITS algorithm by applying the two operators H and A, which are defined for nodes u and a set of nodes L_u that contain links pointing on u as follows:

$$H : hubWeight(u) = \sum_{v \in L_u} authorityWeight(v)$$

$$A : authorityWeight(u) = \sum_{v \in L_u} hubWeight(v)$$

If during the XCRAWL validation phase a node is identified as an authority, the authority weight is ultimately set to 1 and the hub weight is set to 0. When a *hub node*, i.e. a node with a hub weight greater than zero, has been found in the web graph during previous HITS executions, its hub weight remains unchanged and the authority weight is set to 0. 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 weights are changed during the current iteration. The hub weights are stored in the web graph; the HITS authority weights are not relevant in that context and will be 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 $\in$ HUBS: Given all the hubs from the focused crawling step, the algorithm analyses the DOM paths to the links pointing to known authorities and detects the DOM parent element of all such links. Then XCRAWL downloads all the pages published on the same website and which are referenced from the identified subtree 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).

3 Evaluation

3.1 Experimental study

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".

Experiment setup. Two different web mining scenarios were selected that are common for 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

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

(a) Digital camera domain (Table)

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

(b) Notebook domain (Table)

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

(c) MP3 player domain (Text)

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

(d) Cell phone domain (Text)

Table 2. Number of identified authorities

form. This scenario is also the driving scenario of the ALLRIGHT project [17], 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*.

Correspondingly, different types of validators were used in the two scenarios. The first case utilized the standard validator of the ALLRIGHT system which is able to recognize if a page contains a *tabular presentation* of a target instance [17]. In the second scenario we used a Bayesian Network (BN) classifier implemented in the WEKA framework [18], which was trained to recognize *text reviews* of the products.

The second scenario, in which a different validator was used, was specifically selected in order to demonstrate that xCRAWL is not limited to the specific web mining problems for which it was originally developed.

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 are 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 [5]. Note also that the evaluation within the experiments was carried out for very specific topics thus 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 3.2. Details of the table recognition algorithm are given in [11].

We considered two approaches to focused crawling as a baseline for our evaluation: Context Focused Crawler (CFC) [9] and the crawler similar to one developed by Chakrabarti et al. [5]. 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 used search engines (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, which was identified by the AQG method as an authority. Therefore we compare our results only with the results obtained by only by the combination of a main validator and an apprentice similar to method described in [5].

In each experiment, the crawlers were limited to the list of top six 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[1].

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 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 [5], 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 a distance from the DOM element that contains that token to the DOM element that represents the link (see [5] 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" .

Measurement method and results. In order to determine the recall values, we manually analyzed how many target instances were actually published on those websites. When this information was not published explicitly, 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 Table 2(a)). We started the evaluation with tests of XCRAWL, which algorithm converged in approximately six hours for the first domain of digital cameras. Next, the baseline crawler was also executed for the same period of time. In all other tests we used the same run-time strategy. The number of found instances was compared with the number of actually existing instances. The detailed numbers for the individual websites are shown in Table 2.

The results of the evaluation are illustrated per experiment in Figure 3. XCRAWL performed significantly better in every tested domain and major average recall improvements were achieved when compared with the baseline crawler. The average improvement with respect to recall was around 42% in the experiments.

[1]Home/Consumer_Information/Electronics/Photography/Digital_Cameras

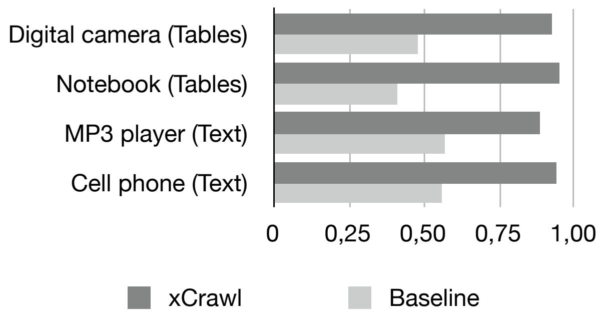

Figure 3. Comparison of average recall

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 those 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 on some 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 detailed XCRAWL running times, 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 4.

XCRAWL parametrization. We evaluated the XCRAWL

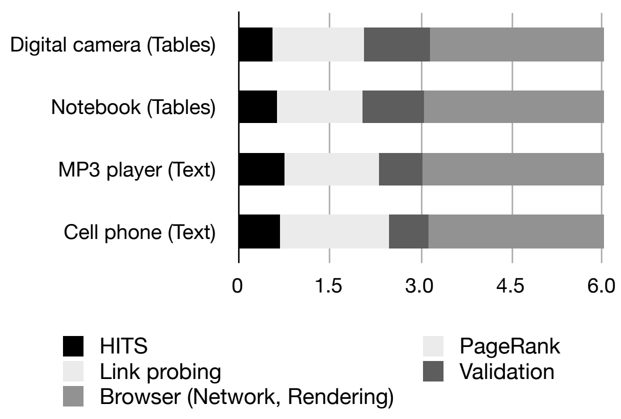

Figure 4. Percentage of time consumed by main XCRAWL tasks

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 was used as the evaluation measure for the probability values. Harvest is 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.

The selection of the *restart probability* value was based based on the observation that the relevance of two web pages decreases considerably when the distance between them on the web graph increases [6]. We started from the value of 0.05 and iteratively increased it up to 0.3 until substantial decrease in the harvest rate was observed (see Figure 5). The best performance in all evaluation domains was observed with a restart probability of 0.15, see Figure 5.

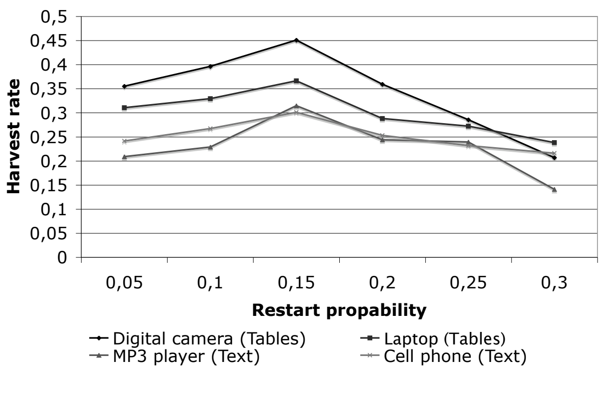

Figure 5. Harvest rate dependency on the restart probability

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 whilst achieving the same harvest rates as in the experiment for determining the restart probability.

3.2 Crawling dynamic HTML pages.

We conclude the discussion of the practical evaluation by giving a few details about the internal browsing and link-detection module of XCRAWL, which was implemented and evaluated in the context of the ALLRIGHT WMS. As already mentioned, 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 reduce the efficiency of the web crawler 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 of the validation techniques – like table recognition methods [11] – require that the document is rendered to the crawling system the same way it is presented to a user.

The XCRAWL system implements a component capable of handling these modern techniques using XULRunner[2], which is the runtime part of the Firefox[3] browser, as a starting point. XULRunner can be used on the same platforms as Firefox itself. In addition, SWT[4] and JavaXPCOM[5] 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 – 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, XCRAWL is able to open menus or other dynamically created DOM elements and probe actions such as “click” or “move over”. The system captures all the navigation events and web page states that are generated by the browser in response to simulated user interaction. XCRAWL can effec-

[2]http://developer.mozilla.org/en/docs/XULRunner
[3]http://en.www.mozilla.com/en/firefox/
[4]http://www.eclipse.org/swt/
[5]http://developer.mozilla.org/en/docs/JavaXPCOM

tively 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. The only limitation of this technique is that it cannot simulate user input into text fields. Thus it can fail to extract all links with some modern AJAX-based web applications. All captured URLs are added to the web graph, thus creating a better approximation to the actual website graph.

4 Conclusions

In this paper we have presented a new crawling technique whose goal is to increase the "recall" measure which is of particular importance for IE approaches 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. The detailed evaluation of our approach in four popular domains indicates that it outperforms state-of-the-art focused crawling techniques and finds a higher number relevant documents from a domain within a in given period of time.

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 funded partly by grants from the Austrian Research Promotion Agency (Program Line FIT-IT Semantic Systems Project AllRight, Contract 809261) and by grants of the Austrian Science Fund (Project V-Know, contract 19996).

References

[1] 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*, pages 96–105, New York, NY, USA, 2001.

[2] E. Agichtein and L. Gravano. Querying text databases for efficient information extraction. In *Proceedings of the 19th IEEE International Conference on Data Engineering*, pages 113–124, Bangalore, India, 2003.

[3] A. Bergholz and B. Chidlovskii. Crawling for domain-specific hidden web resources. In *Proceedings of the Fourth International Conference on Web Information Systems Engineering*, pages 125–133, Washington, DC, USA, 2003.

[4] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. *Computer Networks and ISDN Systems*, 30(1-7):107–117, 1998.

[5] S. Chakrabarti, K. Punera, and M. Subramanyam. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th International World Wide Web Conference*, pages 148–159, Honolulu, Hawaii, USA, 2002.

[6] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16):1623–1640, 1999.

[7] 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*, 118(1):69–113, 2000.

[8] A. Dasgupta, A. Ghosh, R. Kumar, C. Olston, S. Pandey, and A. Tomkins. The discoverability of the web. In *Proceedings of the 16th international conference on World Wide Web*, pages 421–430, Banff, Alberta, Canada, 2007.

[9] 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*, pages 527–534, Cairo, Egypt, 2000.

[10] M. Ester, M. Grob, and H. Kriegel. Focused Web crawling: A generic framework for specifying the user interest and for adaptive crawling strategies. In *Proceedings of 27th International Conference on Very Large Data Bases*, pages 321–329, Roma, Italy, 2001.

[11] 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*, pages 71–80, Banff, Alberta, Canada, 2007.

[12] 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*, pages 265–276, New York, NY, USA, 2006.

[13] J. Kleinberg. Authoritative sources in a hyperlinked environment. *Journal of the ACM (JACM)*, 46(5):604–632, 1999.

[14] R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Trawling the Web for emerging cyber-communities. *Computer Networks*, 31(11-16):1481–1493, 1999.

[15] J. Rennie and A. McCallum. Using reinforcement learning to spider the web efficiently. In *Proceedings of the Sixteenth International Conference on Machine Learning*, pages 335–343, San Francisco, CA, USA, 1999.

[16] 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*, pages 1015–1016, New York, NY, USA, 2006.

[17] 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*, pages 466–479, Busan, Korea, 2007.

[18] I. Witten and E. Frank. *Data mining: Practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann, 2000.

[19] 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 eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 239–248, New York, NY, USA, 2002.