A Conceptual Framework for Efficient Web Crawling in Virtual Integration Contexts

Inma Hernández, Hassan A. Sleiman, David Ruiz, and Rafael Corchuelo

University of Seville,
Seville, Spain
{inmahernandez,hassansleiman,druiz,corchu}@us.es
http://www.tdg-seville.info

Abstract. Virtual Integration systems require a crawling tool able to navigate and reach relevant pages in the Web in an efficient way. Existing proposals in the crawling area are aware of the efficiency problem, but still most of them need to download pages in order to classify them as relevant or not. In this paper, we present a conceptual framework for designing crawlers supported by a web page classifier that relies solely on URLs to determine page relevance. Such a crawler is able to choose in each step only the URLs that lead to relevant pages, and therefore reduces the number of unnecessary pages downloaded, optimising bandwidth and making it efficient and suitable for virtual integration systems. Our preliminary experiments show that such a classifier is able to distinguish between links leading to different kinds of pages, without previous intervention from the user.

Keywords: Crawlers, Web Navigation, Virtual Integration.

Virtual Integration aims at accessing web information in an automated manner. The virtual integration process starts with queries, in which users express their interests and information needs, and its goal is to obtain information relevant to those queries from the web (probably from different sites), and present it uniformly to the users. By relevant page, we mean a page that contains the information required to answer a user query.

Automated access to the web requires a crawler, that is, a tool able to navigate through web sites, looking for relevant pages, from which to extract the information that is returned to the user. Note that this process is online, which means that bandwidth and efficiency are important issues regarding virtual integration crawling [8], and downloading a minimum number of irrelevant pages is mandatory.

In the design of a virtual integration crawler we consider some requirements: first, the crawler must be able to fill in and submit forms, to access pages in

⋆ Supported by the European Commission (FEDER), the Spanish and the Andalusian R&D&I programmes (grants TIN2007-64119, P07-TIC-2602, P08-TIC-4100, TIN2008-04718-E, TIN2010-21744, TIN2010-09809-E, TIN2010-10811-E, and TIN2010-09988-E).

the Deep Web; then, as we said before, the crawler must be efficient, that is, it should minimise bandwidth usage and number of irrelevant pages downloaded. To accomplish this, features for classification must be located outside the page being classified; finally, creating large labelled training sets is burdensome for the user, so we focus instead on training the crawler using an unlabelled set obtained automatically.

Our goal in this paper is to present a conceptual framework that supports the design of crawlers supported by URL-based classifiers. These crawlers determine a page relevance from its URL without having to download it, which reduces the bandwidth and makes them efficient and suitable for virtual integration systems. Even though there are other crawling techniques and tools available that improve traditional crawlers efficiency, our proposal is different, since it is based on link classification to avoid downloading irrelevant pages. We focus on crawling web sites that are designed following a certain navigation pattern, which is the most common pattern in the Web [12]. This pattern consists on a form page that allows issuing queries, followed by a hub page that contains a list of responses to the queries, each of them containing links that finally lead to detail pages.

The rest of the article is structured as follows. Section 2 describes the related work; Section 3 presents the conceptual framework proposed to solve the aforementioned problem; finally, Section 4 lists some of the conclusions drawn from the research and concludes the article.

1 Related Work

Next, we describe the related work in the area of web crawling, enumerating the existing theoretical techniques, and analysing them according to the previous requirements.

1.1 Crawling Techniques

Crawlers are designed considering different requirements, according to their purpose. We distinguish between traditional crawlers, recorders, focused crawlers, and others.

Traditional crawlers [18] collect as many pages as possible from the Web, starting at a given entry point and following links until they meet some stopping conditions. These crawlers have many applications, being the most obvious to create a cache of all visited pages so that a search engine can index them later. Other typical crawling tasks are related to web site maintenance, like validating HTML or performing stress testing.

A recorder is a crawler in which each navigation step is defined by the user. Some examples of recorders are [3], [4], [6], [15], [21]. All of them rely on the user to define which links should be followed in every step, what forms to be filled, and which words to be used for that purpose, so that the crawler reaches exactly the pages targeted by the user. To help the user in the definition tasks, many proposals include a supporting graphical interface [3], [15]. However, in non-GUI proposals, the user has to know HTML code details, and define a step-by-step script, including the references to fields that must be filled in, the actions needed to submit the form (i.e., clicking on a button or a link), and the indication of what links to be followed. In both cases, the user has to provide the values for the different form fields.

Focused crawlers [1], [7], [5], [14], [16], [17] are crawlers which retrieve all pages belonging to a certain topic. They are used, for example, to create a corpora of documents for information retrieval purposes. Their behaviour is similar to that of a traditional crawler, but every retrieved page is analysed to check if it belongs to the topic, usually, with the help of a content-based web page classifier. If the page belongs to the topic, all its links become new seeds for the crawler. Otherwise, the page is judged not relevant and discarded.

Content-based classifiers use either: i) features in the page itself, often a bag of words; or ii) features that are in neighbour pages (pages that either link to or are linked by the target page), like words in the text surrounding the link, the link anchor, and the URL itself. Getting features from the linking page for classification avoids downloading the page beforehand. If the text surrounding the anchor or the anchor text itself contain descriptive words, it is possible to decide the page topic prior to downloading it.

Other crawlers consider different selection criteria for retrieved pages. For example, features like page structure or its location inside the web site directory. Furthermore, they are automated, requiring little intervention from the user, which distinguishes them from recorders. Some examples are [12], [13], [20].

1.2 Analysis

In Table 1, we present a comparison of existing crawling techniques, regarding the following requirements:

- Form filling: either user defined (UD) or applying intelligent techniques. As for the source of keywords for filling, it can be either values from a database (DB), provided by the user (UD), or extracted from the site itself (FA, TDF-IDF) (Column 1)
- Efficiency: Optimisations made to the crawler to reduce the number of irrelevant pages retrieved (Column 2)
- Features: whether they are obtained from the page itself (TP) or from the pages linking to it (LP) (Column 3)
- Training set: labelled (L) or unlabelled (U) (Column 4).

2 Conceptual Framework

We first present an overview of the framework, as shown in Figure 1. Then, we present the details of each module, including a definition of its responsibilities, an example of a typical use case, a list of the possible issues that should be considered in the design, and a discussion of the alternative solutions.

Table 1. Related work (UD = User Defined, ML=Machine Learning, FA=Frequency Analysis, LP=Linking Pages, TP=Target Pages, L=Labelled, U=Unlabelled)

CRAWLING TECHNIQUE	PROPOSAL	FORM FILLING		EFFICIENCY	FE AT	TS
		TECHNIQUE	KEYW			
TRADITION.	Ravaghan01	Matching fields - values	DB	-	-	L
	Barbosa04	Automated	FA	-	-	U
	Madhavan08	Automated	TF-IDF	-	-	U
	Ntoulas05	Query selection algorithm	DB, FA	-	-	U
FOCUSED	Chakrabarti98	-	-	-	LP	U
	Chakrabarti99	-	-	Content classifier & hubs and authorities location	TP	L
	Aggarwal01	-	-	Multiple features classifier	LP	L
	Mukherjea04	-	-	Nearness in directory structure and discarding useless directories	TP	L
	Pant05	-	-	Pre-crawled classifiers	TP	L
	Barbosa05	-	-	Link classifier	LP	L
	Pant06	-	-	Link context classifier	LP	L
	Assis07	-	-	No training, genre & topic classifier	TP	-
	Partalas08	-	-	Reinforcement Learning	TP	L
RECORDER	Anupam00	UD	UD	UD	-	-
	Davulcu99	UD	UD	UD	-	-
	Pan02	UD	UD	UD	-	-
	Baumgartner05	UD	UD	Links matching XPATH given by user	-	L
	Blythe07	UD	UD	Links matching model from users actions	-	L
	Bertoli08	Discard password and keyword forms	-	Links matching model from users actions	-	L
	Wang08	Automated	UD	Paradigm Page-Keyword-Action	-	L
OTHERS	Liddle02	Default Query (empty fields)	-	-	-	
	Lage04	Matching fields - values	UD	Object-rich page classifer; detect Next links	-	
	Vidal07	Automated	FA	Structural classifier	TP	L

A virtual integration process starts with an enquirer, which translates the user interests into queries that are issued to forms. Usually, responses to queries are hub pages, lists of results ordered and indexed, each of them showing a link to another page with detailed information. Relevant pages, when found, are passed on to the information extractor, which obtains and structures the information, that is returned to the user.

We distinguish two phases: the training and the normal execution phase. In the training phase, the keyword manager and form filler focus on obtaining automatically a set of links from the site under analysis, which is later used to train the classifier. In the latter phase, the form filler is used to reach pages behind the forms, and then the crawler uses the trained classifier to select which links to follow. In this paper we focus on the training phase of the framework, namely in the setup and classifier modules.

The only requirement for the training set is to be representative of the site under analysis, hence it is extracted from hub pages, which contain a high number of links in comparison with the rest of pages in any site. Furthermore, they are pages linking directly to pages containing the relevant information, so they assure that we have examples of links leading to relevant pages.

2.1 Keyword Manager

The keyword manager is responsible for finding a list of keywords that allow to obtain a representative collection of links when performing the corresponding searches in a given web site. As an example, to obtain a collection of links from Amazon.com, which offers a variety of products, the keyword manager chooses

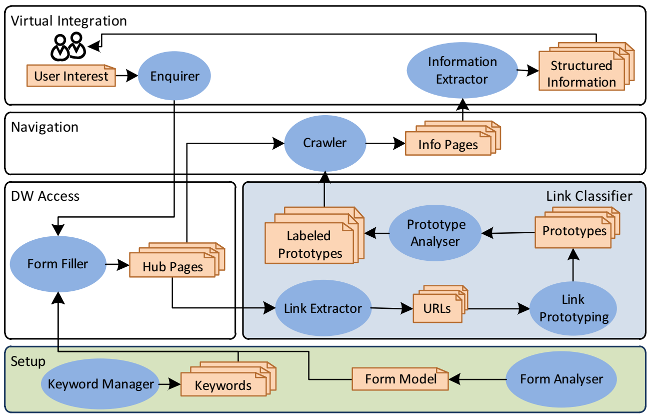

Fig. 1. Conceptual framework diagram

a list of the most common English words [9]. Instead, another site like Microsoft Academic Search belongs to a more specific domain, so a list of the most cited authors, for example, would be more useful for this purpose.

The main concerns in this module are related to the language and type of words that are accepted by each site, specially stop words. For instance, stop words tend to have a higher frequency, and they usually yield a higher number of links, but not every site takes stops words into account. Consider Wordpress.com, which is unable to find any result related to the keywords 'a' or 'the', while the same words in Youtube.com yield respectively 32,800,000 and 40,000,000 results. Furthermore, stop words may deviate the search and deteriorate results. The lexical type of word must also be considered, given that verbs are not as frequent as nouns, for example, so they may yield a smaller number of results. Other important factor is the domain to which the site belongs, since it defines a specific vocabulary.

The simplest solution consists of finding a public or well-known corpus, like the Oxford English Dictionary [9]. In some specifical domains, it may be more difficult to find such a list. For example, to search in Apple Store, we need to find a list of the most frequent words in English related to media and technology. If we try to use the list of English most common nouns, we find that for 'week' (17th position), the Apple store is unable to find any related results. However, in a more general site like Amazon, the same keyword yields 61,494 results.

The last resort is to use the pages of a site to extract the list of keywords, by performing a frequency analysis of the words in the site pages.

2.2 Form Analyser

The form analyser is responsible for visiting a given site, obtaining all the information about the forms and fields that it contains and using that information to build a site form model. For example, to extract information from Amazon, the form analyser opens the Amazon home page, where it finds the following form:

```
<form action="searchAction" name="site-search">
 <select name="url" id="searchDropdownBox" >
  <option value="aps">...</option>
 </select>
 <input type="text" id="twotabsearchtextbox"
name="field-keywords" />
 <input type="image"
src="http://images-amazon.com/images/..."/>
</form>
```

The analyser then obtains a model that includes the form, with a name attribute with value site-search, and no id attribute, its three fields, and its submission method, which consists of clicking over the image.

One of the main issues to be solved by the analyser is the lack of standardisation in forms and fields identification. The analyser (and later, the form filler) has to deal with the problem of referencing that HTML element for later processing. HTML standard includes identification attributes for each element ("id" and "name"), but actually in some web sites we find form elements with none of them. In the latter case, the analyser needs to use a different location strategy, usually in the form of an XPATH expression.

However, although XPATH is flexible and allows to define an expression for every element, this expression can be hard to understand and handle, and also sensitive to small changes in the HTML page. Of course, changes in an element id or name can also invalidate the former location strategy, but it is more usual to change an HTML page by inserting a new paragraph, or deleting an image, than to alter an element id or name attributes.

Youtube.com is a good example of the variety of ways to identify form elements. In its home page we can find, amongst others, a form with id and no name: <form id="masthead-search" action="/results" >, a form with name and no id <form name="logoutForm" action="/">, and a form without name nor id <form action= "/addtoajax">

There are many form modeling proposals, ranging from simple models that just keep a record of all the fields to more complex models that add semantics to each field, analysing field tags [2], [10], [12], [18] surrounding text, identifying mandatory fields [19] or relationships between fields [10].

2.3 Link Extractor

The link extractor is responsible for the extraction of links from hub pages. In the Amazon example, for every page retrieved by the form filler, the extractor analyses all the anchors in the page, including the following:

```
<a id="navLogo" href="/ref=logo">
<a href="http://www.amazon.com/Time-1-year-
auto-renewal/dp/ref=sr1?ie=UTF8"><img
src="http://images-amazon.com/AA160.jpg" /></a>
```

```
3: <a href="http://www.amazon.com/Time-1-year-
   auto-renewal/dp/ref=sr1?ie=UTF8">Time (1-year
   auto-renewal)</a>
4: <a href="http://www.amazon.com/Blind-Side
   -Sandra-Bullock/dp/ref=sr7?ie=UTF8&s=dvd">The Blind Side</a>
5: <a href="http://www.amazon.com/Time-1-year-
   auto-renewal/product-reviews/ref=sr1img?ie=UTF8"/>
6: <a href="javascript:void(0);">Let us know</a>
7: <a href="http://ad.doubleclick.net/clk;...."> <img
   id="nav-site" src="http://images -amazon.com/img2.jpg"></a>
8: <a href="http://www.amazon.com/gp/offer-
   listing/ref=olp?ie=UTF8&condition= new">79 new</a>
9: <a href="http://www.amazon.com/gp/offer-
   listing/ref=olp?ie=UTF8&condition= used">43 used</a>
```

The main issue for the extractor is that a single URL may be written in different formats, either in relative or absolute form. For example, in Amazon, link 1 can be written http://www.amazon.com/ref=logo, /ref=logo or ./ref=logo. Therefore, the system needs to transform all links to their absolute form.

A related issue are links that do not lead to a page, e.g, links to JavaScript functions (in the previous example, line 6), and hence should be discarded, as they are not useful for our purposes. Duplicated links have to be discarded as well.

2.4 Link Prototyping

Its goal is to build a collection of prototypes from the set of extracted links. Each prototype represents a subset of URLs, hence it is defined by a regular expression, and the set of all prototypes is later used to build a link classifier. In traditional machine learning approaches, prototype based classifiers classify elements by computing the distance between the element to be classified and each prototype, and assigning the element to the cluster of the prototype whose distance is lower. In our case, instead, whenever a link matches the regular expression of a prototype, it is assigned to its related cluster. In addition, for each prototype, a coverage value is estimated that gives a hint about the importance of the cluster, by counting the number of URLs in the link training set that match its regular expression. In the Amazon example, analysis of training links yields the prototypes in Table 2.

The link analysis is supported by a tokeniser, which parses every URL and splits it into its different components according to RFC 3986. Sometimes URLs include special characters, spaces and other symbols that make it difficult parsing URLs. Furthermore, URL query strings contain parameters, which may be optional or mandatory, and which may be arranged in different orders. The generator has to detect this in order to make a more accurate regular expression.

URL rewriting makes things worse by eliminating the structure of a query string in exchange for 'friendly' URLs with a better readability. URL rewriting is not a standard procedure, but only a concept that is being adopted lately by many popular web sites, each of them defining their own friendly URL format.

Table 2. Prototypes obtained from Amazon.com, before user labelling

Id	Prototype (Regular Expression)	Coverage
P0	^http://www.amazon.com/.+/product-reviews/.+?ie=UTF8$	30%
P1	^http://www.amazon.com/.+/dp/.+?ie=UTF8&s=dvd$	17%
P2	^http://www.amazon.com/.+/dp/.+?ie=UTF8$	13%
P3	^http://www.amazon.com/gp/offer-listing/ref=olp?ie=UTF8$	14%
...		
Pn	^http://ad.doubleclick.net/.+/feature.html?docid=.+$	0.5%

Table 3. Labelled prototypes obtained from Amazon.com

Label	Prototype (Regular Expression)	Cov.
Product Reviews	^http://www.amazon.com/.+/product-reviews/.+?ie=UTF8$	30%
Product Descriptions	^http://www.amazon.com/.+/dp/.+?ie=UTF8$	30%
Buy New Products	^http://www.amazon.com/gp/offer-listing/ref=olp?ie=UTF8&condition=new$	8%
Buy Used Products	^http://www.amazon.com/gp/offer-listing/ref=olp?ie=UTF8&condition=used$	6%

2.5 Prototype Analyser

Once the prototyping is complete, the prototypes are analysed and improved so that the result is more accurate. Finally, the analyser helps the user to assign a label to each prototype, defining the semantic concept contained in the links of the cluster that the prototype represents. Moreover, the user selects from the set of prototypes those representing relevant concepts, which will be considered during later crawling.

In the Amazon example, after processing the prototypes the analyser outputs, amongst others, the labelled prototypes included in Table 3.

There are some ways to improve prototypes accuracy, mainly: prototype joining, prototype splitting and prototype discarding.

Joining two different prototypes representing the same concept results in a single prototype with a more general regular expression. For example, prototype P1 is composed of pages with DVD products information, while P2 represent pages about any other type of products. They are joined to form Product Description prototype in Table 3.

Splitting a prototype results in two smaller and more cohesive prototypes. For example, in Amazon prototype P3 includes both links to buy new and used products. The analyser splits this prototype into two different prototypes: one for buying used products and one for buying new products.

URLs that appear only a few times in the training set and whose format is completely different from the other URLs, lead to a prototype with low coverage, which are seldom helpful for the crawler, (e.g., advertising URLs), so they can be discarded without diminishing the classification power of the prototype set. For example, the last prototype in Table 2, whose coverage is low in comparison with the others (0.5 %), is excluded from the labelled prototype set in Table 3.

3 Conclusions

In this paper, we present a conceptual framework for designing crawlers supported by a prototype based link classifier, that classifies pages according to their URL format without downloading them beforehand. Parting from an unlabelled set of links, a set of prototypes is built, each of them representing one of the different concepts embodied in a particular web site. Then, the user selects those concepts that are relevant, and the crawler uses the related prototypes to reach only pages containing those concepts. With respect to the requirements we mentioned in section 1, we observe the following:

Efficiency: Our proposal classifies web pages depending on the link URL format, it is not only efficient, but also generic and applicable in different domains.

Traditional crawlers browse the whole site, retrieving all pages, and spending a significant time and bandwidth while downloading them. Focused crawlers retrieve pages belonging to a topic more efficiently than traditional crawlers do, but still a page has to be classified to know if the crawler must follow that path, and that requires the page to be downloaded in most cases.

In general, using features located on a page to classify it requires downloading it previously, which results in wasted bandwidth. There are, some proposals that classify pages according to the anchor text and text surrounding the link in the referring page. However, not all sites include in their links and their surroundings words useful for classification. The same problem of specificity can be noted in ad-hoc techniques (classifiers designed for a specific site), and also in recorders.

Form Filling: Our goal is to adapt existing form modeling proposals and integrate them into our crawler. As we explained in Section 1, form filling has been studied thoroughly in the literature, so it is not our matter of research.

Unlabelled training set: In this proposal, the classifier is trained using a set of links collected automatically. The system analyses them and gives the user a list of prototypes representing concepts, while the user is only responsible for defining his or her interest, by labeling and picking one or more prototypes. Users intervention is unavoidable, given that the relevancy criteria depends solely on them. Recorders provide efficient crawlers that retrieve only relevant pages, but they depend entirely on the user.

As a result, we designed an efficient crawler, able to access web pages automatically, while requiring as little intervention as possible from the users. Some preliminary results of our implementation of the link classifier are shown in [11].

References

1. Aggarwal, C.C., Al-Garawi, F., Yu, P.S.: On the design of a learning crawler for topical resource discovery. ACM Trans. Inf. Syst. 19(3), 286–309 (2001)
2. Álvarez, M., Raposo, J., Pan, A., Cacheda, F., Bellas, F., Carneiro, V.: Crawling the content hidden behind web forms. In: ICCSA (2), pp. 322–333 (2007)
3. Anupam, V., Freire, J., Kumar, B., Lieuwen, D.F.: Automating web navigation with the webvcr. Computer Networks 33(1-6), 503–517 (2000)

4. Blythe, J., Kapoor, D., Knoblock, C.A., Lerman, K., Minton, S.: Information integration for the masses. J. UCS 14(11), 1811–1837 (2008)
5. Chakrabarti, S.: Focused web crawling. In: Encyclopedia of Database Systems, pp. 1147–1155 (2009)
6. Davulcu, H., Freire, J., Kifer, M., Ramakrishnan, I.V.: A layered architecture for querying dynamic web content. In: SIGMOD Conference, pp. 491–502 (1999)
7. de Assis, G.T., Laender, A.H.F., Gonçalves, M.A., da Silva, A.S.: Exploiting genre in focused crawling. In: String Processing and Information Retrieval, pp. 62–73 (2007)
8. Edwards, J., McCurley, K.S., Tomlin, J.A.: An adaptive model for optimizing performance of an incremental web crawler. In: WWW, pp. 106–113 (2001)
9. Fowler, H.W., Fowler, F.G.: Concise Oxford English Dictionary, 11th edn. revised. Oxford University Press, Oxford (2008)
10. He, H., Meng, W., Lu, Y., Yu, C.T., Wu, Z.: Towards deeper understanding of the search interfaces of the deep web. In: WWW, pp. 133–155 (2007)
11. Hernández, I.: Relc demo (2011), `http://www.tdg-seville.info/inmahernandez/Thesis+Demo`,
12. Lage, J.P., da Silva, A.S., Golgher, P.B., Laender, A.H.F.: Automatic generation of agents for collecting hidden web pages for data extraction. Data Knowl. Eng. 49(2), 177–196 (2004)
13. Liddle, S.W., Embley, D.W., Scott, D.T., Yau, S.H.: Extracting data behind web forms. In: ER (Workshops), pp. 402–413 (2002)
14. Mukherjea, S.: Discovering and analyzing world wide web collections. Knowl. Inf. Syst. 6(2), 230–241 (2004)
15. Pan, A., Raposo, J., Álvarez, M., Hidalgo, J., Viña, Á.: Semi-automatic wrapper generation for commercial web sources. In: Engineering Information Systems in the Internet Context, pp. 265–283 (2002)
16. Pant, G., Srinivasan, P.: Link contexts in classifier-guided topical crawlers. IEEE Trans. Knowl. Data Eng. 18(1), 107–122 (2006)
17. Partalas, I., Paliouras, G., Vlahavas, I.P.: Reinforcement learning with classifier selection for focused crawling. In: European Conference on Artificial Intelligence, pp. 759–760 (2008)
18. Raghavan, S., Garcia-Molina, H.: Crawling the hidden web. In: World Wide Web Conference Series (2001)
19. Shu, L., Meng, W., He, H., Yu, C.T.: Querying capability modeling and construction of deep web sources. In: Web Information Systems Engineering, pp. 13–25 (2007)
20. Vidal, M.L.A., da Silva, A.S., de Moura, E.S., Cavalcanti, J.M.B.: Structure-based crawling in the hidden web. J. UCS 14(11), 1857–1876 (2008)
21. Wang, Y., Hornung, T.: Deep web navigation by example. In: BIS (Workshops), pp. 131–140 (2008)