

Intelligent Web Navigation *

Inma Hernández

Departamento de Lenguajes y Sistemas Informáticos
Universidad de Sevilla
Avda. Reina Mercedes s/n
41012 Sevilla Spain
{inmahernandez}@us.es

Abstract. Virtual integration systems retrieve information from several web applications according to a user's interests. Being an online process, response time is a significant factor. Usually web pages contain a high number of links, some of them leading to interesting information, but most of them having other purposes, like advertising or internal site navigation. Traditional crawlers follow every link in each page, retrieving, analysing and classifying thousands of pages for every single site, which is a costly task. This problem can be solved with the combination of a web page classifier, to distinguish between interesting and irrelevant pages, and a link classifier, which automatically identifies links leading to interesting pages. This kind of navigation is more efficient and has a lower cost than traditional crawlers.

Key words: Enterprise Information Integration, Navigation, Information Retrieval, Virtual Integration

1 Motivation

There are two approaches to automated access to deep web information, namely Virtual Integration (also known as Metasearch) and Surfacing (also known as Crawling) [24]. In virtual integration, users define their interests using high-level structured queries, and the system retrieves information related to this query in response. This information is retrieved from many different web application sources, but it is presented uniformly to the users in a transparent way. This process is online, and therefore response time is an important issue. As opposed to virtual integration, surfacing is an off-line process that collects all pages behind a web form by submitting pre-computed queries, and not taking the user's requirements into account. In this case, minimising response time is not a priority.

Both previous approaches have a common phase: once the form has been submitted, a response page is retrieved and processed in order to check if it

* Partially funded by the Spanish National R&D&I Plan under grants TIN2008-04718, and the Andalusian Local Government under grants P07-TIC-02602, and P08-TIC-4100.

is a relevant page, and to use the information contained in it to reach further relevant pages. Traditional exhaustive crawlers use a blind approach to navigation, following every link in every page. This approach has a drawback: usually, web pages contain a large number of links, most of them non-relevant to the user (e.g., advertising or links to partner web sites). Too much time is wasted following all those links, making it less efficient.

As opposed to blind navigation, other approaches include some criteria to decide which links must be visited and which ones not, therefore reducing the number of irrelevant visited links. Focused crawlers, for instance, hold the criterion of avoiding pages not related to a certain topic (and therefore links leading to them). Relevancy criteria can be handcrafted by the user or automatically decided by the navigator, with the support of some reasoning process, usually in the form of a classifier. The latter is what we consider an automated intelligent navigator.

Figure 1 shows a virtual integration process with intelligent navigation, starting with a set of user queries which leads to a collection of response pages. Each one of these pages can be a Web page containing the answer to the query made by the user (detail page), or a paginated list of links to detail pages (hub page). In case the server lacks the answer to the query, the response page usually shows an informative message, and optionally some suggestions (no-results page). Finally, when some unexpected error arises, the response page may just contain some error description (error pages). Hence, a classifier is employed to distinguish between the different kinds of response pages, discarding both no-results and error pages and retrieving detail pages.

If a hub page is detected, it is further analysed in order to discriminate between the links it contains. A usual hub page has different types of links, including advertising, internal site navigation, links leading to detail pages, and links to the previous/following hub page (e.g., “More” or “Next” links). A link classifier, makes this distinction in order to follow only relevant links. Finally, when detail pages have been identified and retrieved, an information extractor is used to extract relevant structured data. This final step is beyond our scope.

The rest of the article is structured as follows. Section 2 describes related work in the navigation and web page classification areas; Section 3 shows the experimental study; Section 4 lists some of the conclusions extracted from the research and concludes the article.

2 Related Work

Next, we briefly describe and analyse the existing proposals in the Web Page Classification and Navigation areas.

2.1 Web Page Classification

Web page classification has been extensively researched, and several techniques have been applied with successful experimental results. In general, we classify

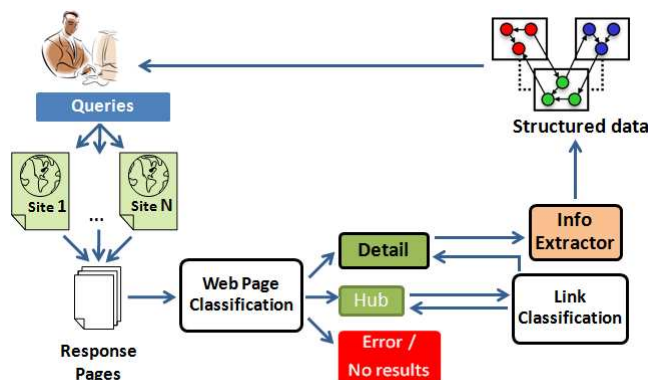


Fig. 1: Virtual Integration example.

them according to the type and location of the classification features. There are three main trends in feature types: content-based ([21], [32] and [34]), structure-based ([3], [4], [9], [15], [20], [18], [36] and [37]) and hybrid classifiers ([11] and [25]). There are also some abstract classification techniques that sort out pages on the basis of any given feature, e.g., PEBL [39]. As for feature location, most approaches obtain features from the page to be classified, whilst others get them from neighbouring pages. These proposals are content-based, and usually rely on features such as the link anchor text, the paragraph text surrounding the anchor [14], the headers preceding the anchor, or a combination of these [19].

2.2 Navigation

Traditional crawlers ([33]) navigate pages by following every link they find. This is useful for certain tasks, like indexing pages for a searching engine, but for virtual integration purposes it is not an efficient approach.

Recorders are also blind navigators, although they are more specific than crawlers, and less flexible. Anupam et. al. [2] present WebVCR, a recording tool with a VCR-like interface to record a user's navigation steps through a web site and store them in a file, in order to replay them automatically in future. Baumgartner et. al. [7] propose another recording system, based on Lixto, which includes an embedded browser in which the user can perform navigation steps. Pan et. al. [28] introduced the WARGO system, a wrapper creation tool, and a language to express navigation sequences called NSEQL. The system offers a browser-based interface to record the user navigation steps, which are transformed in sequences expressed in NSEQL. It is possible to leave some steps undefined until runtime, e.g., the number of clicks the system has to perform on a Next link depending on how many result pages the search returned. To solve this problem, NSEQL is endowed with some extraction capabilities.

The previous proposals interact directly with the web browser interface, so they do not deal with issues like scripts, posting forms, required authentication,

or sites that keep session information. They depend completely upon the user's knowledge, who is responsible for defining the navigation sequences, providing the values to fill in the forms and redefining the sequences whenever the target web site changes. Recording navigation steps makes the navigation inflexible, and error-prone.

Focused Crawling ([1], [17], [5], [6], [12], [13], [27], [29], [30] and [31]) is an improvement over traditional crawlers, in the sense that it combines a traditional crawler with a topical web page classifier, keeping the focus on pages with topics that have some interest for the user, and therefore reducing the number of useless retrieved pages.

However, focused crawlers improvement is limited to the reduction of useless pages retrieved, but the number of useless visited links is still an issue. Even when a high number of pages are discarded, they have to be visited and analysed previously. Moreover, focused crawlers do not classify the functionality of a page, but its topic. However, in a virtual integration system, every page should be processed differently according to the role it plays, like we mentioned before: hubs should be iterated, error pages should be discarded, and detail pages should be processed by an information extractor.

User-Defined Navigation is learned by the system in a supervised way, i.e., the user demonstrates how to obtain the interesting information, and the system is able to generalise this knowledge. EzBuilder [10] is a virtual integration tool used to create information extraction procedures. In order to learn navigation for a web site, the user needs to give an example, submitting forms and navigating to several of the desired pages, and specifying the structure of the data to be extracted. Then, the system builds a model of the user behaviour, that is employed to navigate automatically. This proposal relies on the user, from whom the system learns a navigation model for each site. This model cannot be reused in other sites automatically and the authors do not report any technique to update it.

Davulcu et. al. [16] present a technique to express navigation sequences using navigation maps. These are labelled directed graphs in which nodes represent web pages and edges represent actions that can be executed to navigate from one page to another (submitting a form or following a link). User navigation steps through the site, and form fields information is captured, and added to the navigation map. Some structural changes in web sites can be handled by the system automatically, whilst other changes makes it necessary to update the maps manually.

Other proposals take a workflow-based approach to represent navigation sequences, although the definition of these sequences is performed by the user. Montoto et. al. [26] model navigation sequences as activities in a work flow diagram. They detect some structural page patterns which frequently occur in web sites. However, they do not use classifiers to automatically distinguish between them, and furthermore the sequence of navigation steps to reach these pages is defined by the user; actually their technique is an extension of [28].

[38] is an example of a semi-automated navigator based on a content classifier. Navigation is defined as a sequence of pages, in which the last one is the goal, and the rest are intermediate steps. Each page belongs to a class, which the user defines using a set of keywords, and a set of actions. The system analyses the words in every page looking for class keywords. When a page belongs to an intermediate class, the actions mark the steps to get to the next page in the sequence. However, if the page belongs to a goal class, the actions are extraction rules, in order to extract desired information.

Automated navigators are similar to the former proposals, except for the fact that navigation patterns are automatically extracted from web sites, instead of learnt from the user. Liddle et. al. [23] propose a crawling tool, in which forms are submitted, but no specific information is filled in, in order to retrieve as many results as possible. Once the form is submitted, the system is able to automatically distinguish between an indexed list of results, a complete list of results, an informative page with a no-result message, and an error page. When a hub page is detected due to the presence of a Next link, this link is followed iteratively until the last page (or until a sufficiently large number of pages is found), and all retrieved pages are concatenated in a single result page.

This proposal is very simple, since it is only applicable if we wish to retrieve the summarised information contained in hub pages, but not the extended information in detail pages. Furthermore, this tool retrieves all pages, regardless of their relevance to the user's keywords.

For Palmieri et. al., [22], navigation patterns are a graph-based description of the different stages of a user navigation in a web site. They state that most web sites are designed according to the same navigation patterns, e.g., a start page with a link to an advanced search form, which is submitted to obtain a response page. They consider both HTML submission methods, POST and GET, which are expressed as different patterns. Once the form has been submitted, this approach analyses the response page to check if data objects of interest are present, in which case it is considered a hub page. Every hub page has a link leading to the next hub page, until all results are exposed to the user. Some heuristics are presented in order to detect and deal with these links.

However, this approach does not allow for the fact that that hub pages may only contain a brief summary of the data object and a link to a detail page. This proposal limits the possible navigation patterns to a couple of handmade samples, and therefore is not applicable to all web sites.

Vidal et. al. [36], consider that navigation patterns are automatically detected sequences of regular expression URLs that lead to relevant pages. This proposal receives a sample page, which represents the class of relevant pages, and it returns a navigation pattern, composed of the sequence of regular expressions that describe URLs that lead to the largest number of relevant pages. Just like the former approach, forms are submitted automatically, and the response page is analysed. If the page belongs to the same class as the one given as example (using a structural classifier), it is considered a relevant page, and it is then retrieved. Small pages are considered to be error pages and they are

not processed. Whenever a form is detected, it is submitted again, and process continues recursively. If a large number of links are detected, it is considered a hub page, and a further analysis is needed to select which links will be followed. Links are grouped by their URL similarity, and only one link from each group is followed to check if it leads to a relevant page.

This proposal is semi-supervised, since the process is automated, but the user has to provide a sample detail page, aside from filling in the parameter values for every form submission. This navigator can only reach a small number of possible results, for two reasons: it only keeps navigation patterns leading to the largest number of relevant pages, and besides, links leading to other hub pages are not considered, only links to detail pages, so only relevant pages referenced in the first hub page are retrieved. This navigator has to be combined with an iterator in order to retrieve all possible results. Another question to bear in mind is that navigation systems are ad-hoc. If the site changes its URL nomenclature, the system has to be trained again, thus requiring user intervention, to update navigation patterns.

This proposal has some technical problems to be solved, namely, the heuristic for the treatment of form pages can lead to an infinite loop, in cases in which the server includes the original form in every response page. It does not support form submission using the POST method. Finally, it does not support web sites that keep user session information.

[8] is another automated navigator. It is similar to the former: it relies on a structural-based clustering of pages, and it requires a sample relevant page. All pages are grouped in clusters, according to their structure, and links between pages are analysed in order to find relationships between clusters. Whenever a cluster has a high number of outgoing links to the cluster containing the sample, it is considered a hub cluster. This proposal is not designed to crawl pages behind forms, and therefore should be adapted in order to retrieve pages from the deep web.

3 Experimental Study

In order to justify the need for an intelligent navigator, we analysed a sample set of real web pages, and calculated the percentage of relevant links they included. Sample pages were extracted from Alexa Web Directory¹. Alexa is supported by a traditional crawler, which collects publicly available web pages, rank them and offer additional information about them, like traffic statistics, demographic distribution of page viewers, keywords used to find that page in search engines, and related links. The first ten sites from each category were chosen, excluding Adults and Reference categories. Sites without hubs were discarded too, as well as sites that required user session information, or that enclosed relevant information in iframes (due to some conflicts with the Selenium XPath API), as they are not useful for our purposes. For every web site, a query was issued in order to

¹ <http://www.alexa.com/topsites>

obtain a response hub page. XPath expressions for relevant links in these pages were extracted by hand, including those that lead from one paginated hub to the next one. We developed a Java application using Selenium Core Reference Library², which automatically visited every page, calculated the number of relevant links defined by XPath expressions, as well as the total number of links in the page, and stored the page in a local repository. In order to analyse the

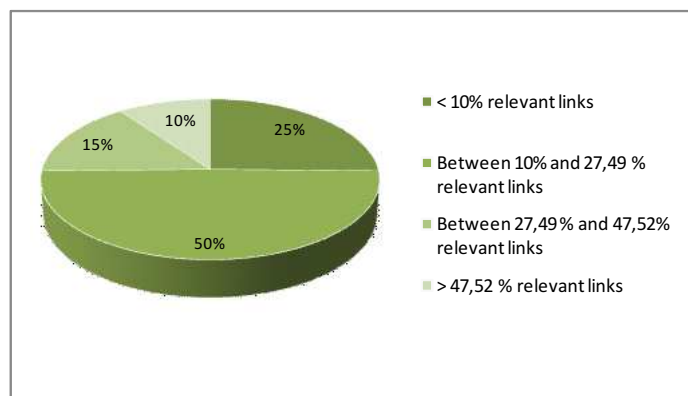


Fig. 2: Percentage Distribution

results, we calculated the percentage of relevant links in every page, and the average percentage of relevant links per category.

Figure 2 shows the percentage distribution we obtained with a sample set of 150 sites. Barely a 10% of the pages contained a percentage of relevant links higher than 47,52%. In fact, the quartile distribution was: (10,01%; 15,19%; 27,49%; 95,67%). The average percentage value of relevant links for all analysed pages was 22,35% with a standard deviation equal to 19,85. Figure 3 left shows the percentage of relevant links for every page in a random category (Arts). Figure 3 right shows the average percentage values by category. Values for all categories were similar, around 20%, except for Health category, which had the highest percentage value, with a 43%. These results confirm that average pages contain a high percentage of useless links, which should be avoided by the crawler in order to improve its efficiency.

4 Conclusions and Research Questions

The main problem posed in this paper is the development of an intelligent navigation system for virtual integration within the *Deep Web*, as opposed to the blind navigation performed by most of the previous works. In this context, users specify their interests by means of queries, and information is retrieved from several

² <http://seleniumhq.org/>

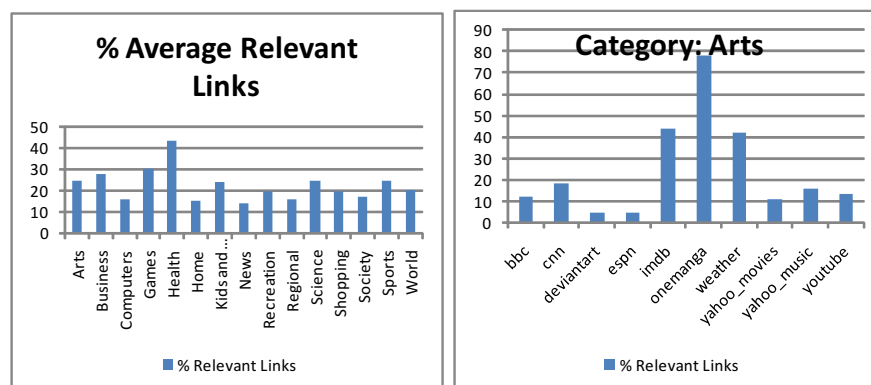


Fig. 3: Experimental Results

web applications. Response pages are analysed, navigating through their links and collecting only those pages that contain relevant information. In contrast, traditional crawlers follow every link and collect every possible page, wasting a lot of time and resources.

In order to support our thesis, we presented an experimental study, in which we estimated the average percentage of relevant links included in every page. We came to the conclusion that most pages in our sample set have a low percentage of relevant links. As we mentioned before, virtual integration is an online process, hence information should be presented to the user in a reasonable response time. Therefore, these systems will indeed find benefits in the application of an intelligent navigator, which avoids visiting useless pages by following only relevant links, reducing the cost and improving the efficiency of traditional crawling systems.

Summarising, these are some other research challenges concerning intelligent navigation:

1. Response pages have to be classified into the different roles that they play, in order to automatically deal with them.
2. Links in hub pages have to be classified before clicking on them, to identify those that lead to relevant pages.
3. Lazy link and web page classifiers work better in this context, i.e., classification model is built and updated progressively during the process.
4. Navigator should interact with the user as little as possible. Therefore, learning is unsupervised, or at least, very little supervised. Also, it should be as general as possible, not having to build an ad-hoc model for every site, but instead developing a general model that can adapt to most sites just by tuning some parameters.
5. Advanced post-filtering of hub pages is also a desirable feature. For example, when looking for products in an online store, a user may wish to retrieve only those products whose price lay within a certain range. This means that the navigator needs to be endowed with a lite extraction tool.

6. It is necessary to create a standard data set to evaluate classification and navigation proposals. Most proposals do not use a well-known data set in their experiments; instead, they collect their own set of pages by issuing queries to a search engine, or using a blind crawler. As the experiments are performed on different sets of pages, the experimental results cannot be compared. There are some well known data sets of already classified pages available for this kind of experiments, e.g., the WebKB collection³, the Reuters-21578⁴ news collection or the TEL-8 query interfaces collection⁵. [35] reports an attempt to collect a standard data set. Unfortunately, these collections are outdated and updating them is a costly task. We argue that more effort on archiving needs to be done so that new proposals can be compared from an empirical point of view.

References

1. C. C. Aggarwal et al. On the design of a learning crawler for topical resource discovery. *ACM Trans. Inf. Syst.*, 19(3), 2001.
2. V. Anupam et al. Automating web navigation with the webvcr. *Computer Networks*, 33(1-6), 2000.
3. A. Arasu and H. Garcia-Molina. Extracting structured data from web pages. In *SIGMOD Conference*, 2003.
4. Z. Bar-Yossef and S. Rajagopalan. Template detection via data mining and its applications. In *WWW*, 2002.
5. L. Barbosa and J. Freire. Searching for hidden-web databases. In *WebDB*, 2005.
6. S. Batsakis et al. Improving the performance of focused web crawlers. *Data & Knowledge Engineering*, In Press, Corrected Proof, 2009.
7. R. Baumgartner et al. Deep web navigation in web data extraction. In *CIMCA/IAWTIC*, 2005.
8. L. Blanco et al. Efficiently locating collections of web pages to wrap. In *WEBIST*, 2005.
9. L. Blanco et al. Structure and semantics of data-intensiveweb pages: An experimental study on their relationships. *J. UCS*, 14(11), 2008.
10. J. Blythe et al. Information integration for the masses. *J. UCS*, 14(11), 2008.
11. J. Caverlee and L. Liu. Qa-pagelet: Data preparation techniques for large-scale data analysis of the deep web. *IEEE Trans. Knowl. Data Eng.*, 17(9), 2005.
12. S. Chakrabarti et al. Automatic resource compilation by analyzing hyperlink structure and associated text. *Computer Networks*, 30(1-7), 1998.
13. S. Chakrabarti et al. Focused crawling: A new approach to topic-specific web resource discovery. *Computer Networks*, 31(11-16), 1999.
14. W. W. Cohen et al. Improving a page classifier with anchor extraction and link analysis. In *NIPS*, 2002.
15. V. Crescenzi et al. Roadrunner: Towards automatic data extraction from large web sites. In *VLDB*, 2001.
16. H. Davulcu et al. A layered architecture for querying dynamic web content. In *SIGMOD Conference*, 1999.

³ <http://www.cs.cmu.edu/afs/cs/project/theo-20/www/data/>

⁴ <http://www.daviddlewis.com/resources/testcollections/reuters21578/>

⁵ <http://metaquerier.cs.uiuc.edu/repository/datasets/tel-8/>

17. G. T. de Assis et al. Exploiting genre in focused crawling. In *SPIRE*, 2007.
18. D. de Castro Reis et al. Automatic web news extraction using tree edit distance. In *WWW*, 2004.
19. J. Fürnkranz et al. Hyperlink ensembles: a case study in hypertext classification. *Information Fusion*, 3(4), 2002.
20. S. Grumbach and G. Mecca. In search of the lost schema. In *ICDT*, 1999.
21. A. Hotho et al. Ontology-based text document clustering. *KI*, 16(4), 2002.
22. J. P. Lage et al. Automatic generation of agents for collecting hidden web pages for data extraction. *Data Knowl. Eng.*, 49(2), 2004.
23. S. W. Liddle et al. Extracting data behind web forms. In *ER (Workshops)*, 2002.
24. J. Madhavan et al. Harnessing the deep web: Present and future. In *CIDR*, 2009.
25. A. Markov et al. The hybrid representation model for web document classification. *Int. J. Intell. Syst.*, 23(6), 2008.
26. P. Montoto et al. A workflow language for web automation. *J. UCS*, 14(11), 2008.
27. S. Mukherjea et al. Discovering and analyzing world wide web collections. *Knowl. Inf. Syst.*, 6(2), 2004.
28. A. Pan et al. Semi-automatic wrapper generation for commercial web sources. In *Engineering Information Systems in the Internet Context*, 2002.
29. G. Pant and P. Srinivasan. Learning to crawl: Comparing classification schemes. *ACM Trans. Inf. Syst.*, 23(4), 2005.
30. G. Pant and P. Srinivasan. Link contexts in classifier-guided topical crawlers. *IEEE Trans. Knowl. Data Eng.*, 18(1), 2006.
31. I. Partalas et al. Reinforcement learning with classifier selection for focused crawling. In *ECAI*, 2008.
32. J. M. Pierre et al. On the automated classification of web sites. *CoRR*, cs.IR/0102002, 2001.
33. S. Raghavan and H. Garcia-Molina. Crawling the hidden web. In *VLDB*, 2001.
34. A. Selamat and S. Omatu. Web page feature selection and classification using neural networks. *Inf. Sci.*, 158, 2004.
35. M. Sinka and D. Corne. A large benchmark dataset for web document clustering. In *Soft Computing Systems: Design, Management and Applications, Volume 87 of Frontiers in Artificial Intelligence and Applications*. 2002.
36. M. L. A. Vidal et al. Structure-based crawling in the hidden web. *J. UCS*, 14(11), 2008.
37. K. Vieira et al. A fast and robust method for web page template detection and removal. In *CIKM*, 2006.
38. Y. Wang and T. Hornung. Deep web navigation by example. In *BIS (Workshops)*, 2008.
39. H. Yu et al. Pebl: Web page classification without negative examples. *IEEE Trans. Knowl. Data Eng.*, 16(1), 2004.