J. Hlaváčová (Ed.): ITAT 2017 Proceedings, pp. 35–39
CEUR Workshop Proceedings Vol. 1885, ISSN 1613-0073, © 2017 R. Pavel, P. Gurský

Focused Web Crawling of Relevant Pages on e-Shops

Rudolf Pavel, Peter Gurský
Institute of Computer Science
Faculty of Science, P.J.Šafárik University in Košice
Jesenná 5, 040 01 Košice, Slovakia
rudolf.pavel@student.upjs.sk, peter.gursky@upjs.sk

Abstract: E-shop data extraction requires all detail pages of products to be crawled. To maintain extracted data actual, the crawling needs to be periodical. E-shops contain many irrelevant pages such as ads, basket or contact information that are good to avoid during a crawling process. This paper presents a focused web crawling method based on an analysis of a previous initial crawling that eliminates irrelevant paths from the following crawls of the e-shop. The method preserves the ability to collect all products of all product domains, even the new ones.

1 Introduction

Focused crawling is a common way to search and collect data relevant to user's needs on the web. The scope of project Kapsa [1] is to retrieve information about e-shop products by crawling and extracting, and presenting them in unified form, which simplifies user's choice of preferred product. This paper focuses on the beginning of the process, i.e. crawling and extracting products data from e-shops.

Crawling all objects (HTML files, images, scripts, CSS files …) from e-shop portal generates quite extensive data traffic. Polite crawling handles this issue by download speed restriction. Polite crawling of even small complete e-shop can easily take more than one hour.

The aim of our crawling is not retrieving all pages of the e-shop, just the pages that contain detail data about products. Any other downloaded page increases crawling time and e-shop's traffic.

To extract the most detailed information about a product on e-shop, the crawler needs to navigate itself to product's detail page. Detail page usually contains product name, price, photos, product properties, product description, customer reviews, etc. Useful property of detail pages is, that they usually have uniform design, different from any other page on e-shop. Thus the identification of a detail page can be done by few simple rules applicable for every product on e-shop: presence of an element on special position of the HTML source (product name, price tag, product image, etc.) and/or special URL form. These rules can be easily created by administrator together with data extraction rules in annotation web browser extension Exago [2]. We believe that automatic content based detection of detail page is also possible, but we didn't focus on it.

Product prices are changing in time, some products can be removed form an e-shop, new products can be added and sometimes a completely new product domain can be inserted to an e-shop portfolio. To keep data about offered products actual, crawling of e-shops needs to be periodical.

The naïve approach to decrease the amount of pages of periodical crawling, would be to remember URLs of all detail pages of an e-shop and download them the next time. This approach decreases the amount of downloaded pages drastically. On the other hand, new products and product domains would not be detected, which is undesirable.

To retrieve all offered products and product domains, the crawling method must fetch also webpages containing lists of products as well as webpages that lead to them. We call these pages, together with detail pages, the *relevant pages*. All other pages and objects are irrelevant and we want to avoid downloading them.

Navigation through relevant pages can be configured by a set of manually created rules too. This approach is typical for most web scrappers. Manual creation of such navigation rules is usually not an easy task, if we want all relevant pages to be downloaded.

In this paper we present our automatic approach to crawl all relevant pages based on analysis of initial complete crawl of an e-shop. Following crawlings of the e-shop navigates on every relevant page excluding irrelevant pages, until e-shop design is changed. The changed design is easily detected, because the original extraction rules cease to function. In this case, crawler can switch to classic crawling approach and inform administrator that the extraction rules need to be changed.

2 Related Work

Focused crawling systems are usually designed to collect web pages with a certain topic. Such crawlers guess the relevance of the page based on anchor texts and PageRank and prioritize the URLs to crawl like in [4,7]. These crawlers do not care about site map.

Methods based on context graphs [12], learning automata [8] and Hidden Markov Models (HMM) [5,6] analyzes downloaded pages with classifiers and set the priority of all URLs found on them uniformly, based on score of the page. The classifiers give high rates to the pages that are similar to pages that leaded to desired goals. The position of the links on the page is not considered.

The method in [9] analyzes the relevance of parts of downloaded pages separately using their HTML structure position and prefers the URLs found in relevant parts of the pages.

Periodical crawling research is mainly focused on estimation of frequency of repeatable crawls to maintain up-to-date data [10].

The combined task of downloading and extracting data from web pages is called Web scrapping. There are a lot of web scrappers on the market. We haven't found any web scrapper, which cooperates with optimized crawling as the one, presented in this paper.

3 Initial Crawling and Its Analysis

Project Kapsa uses a modification of open source web crawler Crawler4j [3]. It's a multithreaded crawler written in Java. If there is a need to simulate clicks and/or scrolling

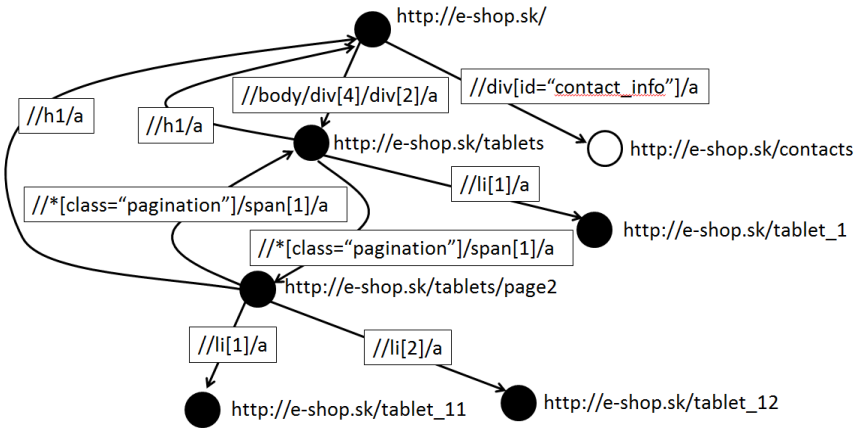

Figure 1: Initial crawling graph of a fictive e-shop http://e-shop.sk. Black vertexes represent relevant pages, white vertex represents a sample of irrelevant pages. Elements on pages containing links to other pages are represented as outgoing edges, labeled with XPath localization of the elements, heading to those pages.

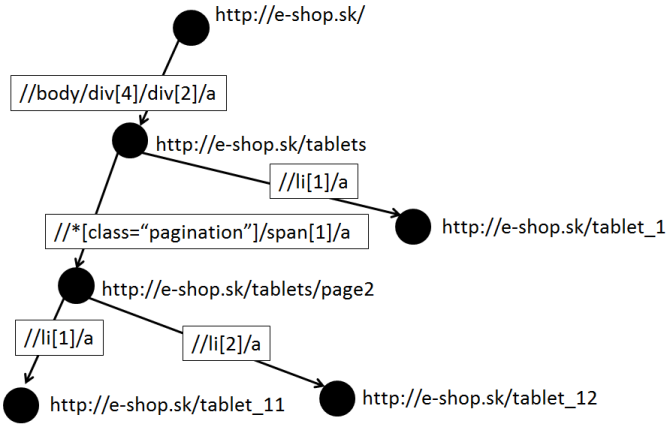

Figure 2: DAG with relevant pages and relevant XPaths extracted from the graph in Figure 1 in bottom-up analysis.

actions on the web page (page content is changed by JavaScript calls), Selenium-WebDriver is used.

Crawling is configured by wrapper – the result of annotation process in web browser extension Exago [2]. Wrapper consists of the following data:

- *seed_URL*, which is the starting page of the crawling process, usually the home page,
- detail page identification rules, and
- product data extraction rules.

The detail page identification rules are a combination of XPaths or/and regular expressions. XPath can localize an element or elements on a web page and regular expression can locate special substrings of the element content. Regular expressions are also applicable to URL. Each rule can be set to be mandatory (the rule must match) or forbidden (the rule may not match).

Initial crawling starts on *seed_URL* and navigates to every object of the same domain recursively. Unlike the standard crawling, we do not store downloaded pages. Every page is checked to be detail page. Detail pages are sent to Extractor module that extracts all product details and stores them to storage. If the page is not a detail page, the crawler analyzes the page source to extract all links together with their XPath position on the page. It is important to note, that every URL is downloaded only once, even if it is present on many pages.

It is usual that there are links to other products on detail pages, typically they are recommended using collaborative filtering ("people that bought this also bought:" section). Our method does not analyze source of detail pages for new URL links on them. There are two main reasons for that:

- All products on a common e-shop are accessible from some product list page, i.e. a page containing list of links to subset of products. It is highly unlikely that there would be some product on e-shop accessible only from other product detail page.
- The navigation graph would contain more edges with no positive effect. With some effort a situation can be found, when we can eliminate a download of some product list page, because all products on it are

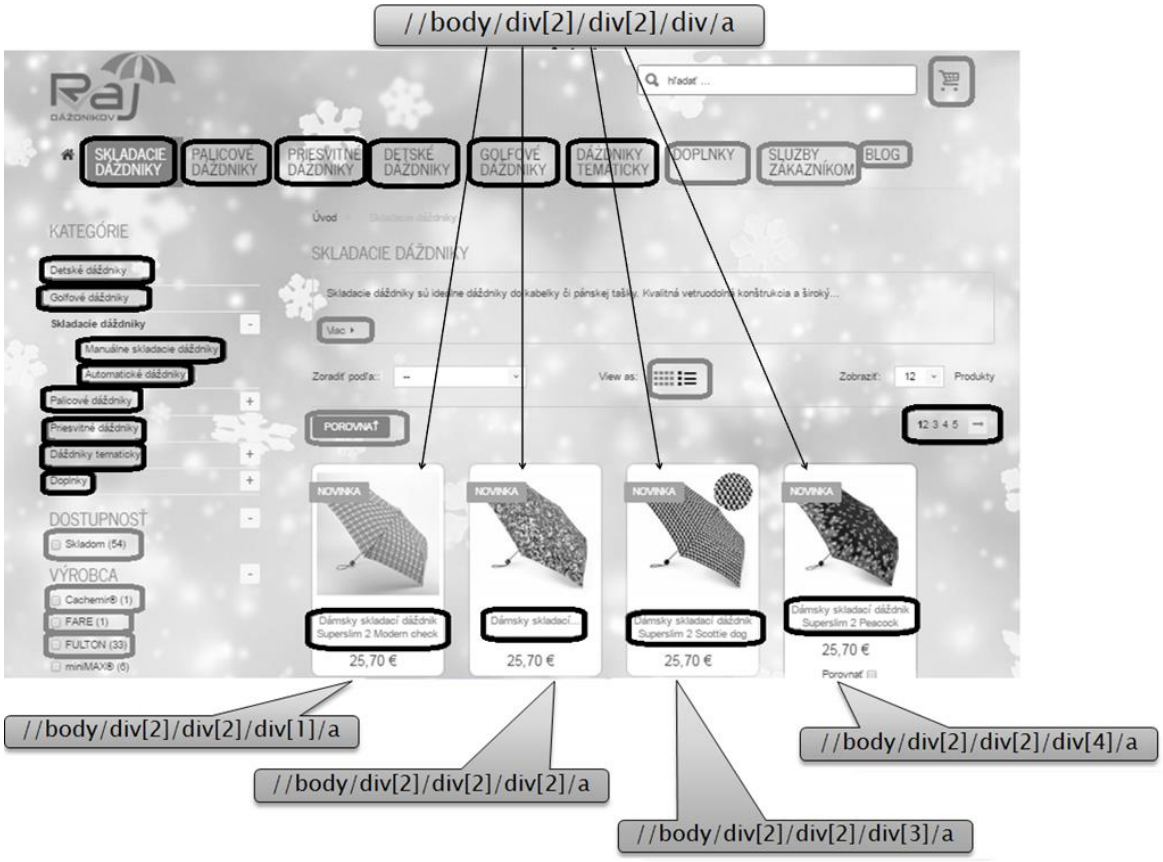

Figure 3: XPaths and their generalized XPath of elements representing list of products on www.rajdazdnikov.sk

accessible from other detail pages, but this can be only temporary status. If a new product would be added, it could be missed out in the next crawl.

The results of the initial crawling are extracted products and a directed graph of the e-shop links. Edges are labeled by XPaths that leads to HTML elements where the link was found. Detail pages have no outgoing edges – thus they are leafs of the graph.

3.1 Bottom-up Analysis: Creation of DAG with Relevant Pages and Relevant Links

When the whole e-shop is crawled and all products are extracted, the analysis of collected crawling data prepares the next optimized crawling, which usually takes place a few days later.

Bottom-up analysis creates a reduced graph that contains only vertexes with relevant pages. This graph is always a directed acyclic graph (DAG).

The analysis follows edges of the initial crawling graph in opposite direction.

- Initial step: every detail page vertex is added to a result graph.
- Iteration step: Let B be a set of all vertexes added in the previous step. Let A be a set of all vertexes, not present in a result graph, which have an outgoing edge to any vertex in B. Add set A and edges from A to B to the result graph.
- The iteration stops, if DAG contains seed_URL or no vertexes were added in the previous step, i.e. set A was empty.

Since we do not add all outgoing edges of vertexes in set A, just the ones that lead to vertexes in B, no cycle can be present in the result graph.

Unfortunately, this graph is not necessary connected. We need a connected graph to reach every detail page by navigation from seed_URL. Therefore, in the final step:

- Let M be a set of all vertexes in DAG with no incoming edges. Add all edges and vertexes on the shortest paths in from seed_URL to every vertex in M in original graph to result DAG.

3.2 Generalization of Relevant XPaths on Relevant Pages

During the initial crawling, each visited web page, except the detail pages, is processed to localize URL links in the HTML source. For every element containing URL, the pair of URL and XPath localizing the element in HTML is stored. Bottom-up analysis divides pages and XPaths to relevant and irrelevant. Fig. 3. shows a print screen of e-shop www.rajdazdnikov.sk. Links to relevant pages are encircled by light gray oval and irrelevant ones by a dark gray oval. When crawling, we want to follow every relevant link and no irrelevant links.

If we look at the elements of relevant links, we can see that some of them have very similar XPaths. This is, because they are presented to user in repeating structures,

typically in lists. On Fig. 3 we can see four XPaths of elements with links heading to detail pages of umbrellas. The XPaths differ only in numbers in the last brackets. When we remove the brackets with the numbers, the resulting single generalized XPath localizes all four elements. This generalized XPath will localize all products on any page of the same HTML template, even if it contains less or more products.

Input for the algorithm that extracts generalized XPaths, is a list of XPath-URL pairs of relevant links found on the page. First, the pairs are divided to clusters of pairs, which differ only in one number between brackets on the same position. This is done by iterative partial clustering based on the longest common prefix. Finally, in each cluster, the different numbers and surrounding brackets of XPaths are removed resulting in a single generalized XPath for the cluster.

3.3 Top-Down Analysis: Creation of Generalized XPaths Graph

Having the algorithm that computes a list of generalized XPaths, we can run the algorithm on each relevant page of the DAG except the detail pages. The result is the modified DAG that has all edge labels replaced by generalized versions of the previous XPaths.

There is an important observation that pages accessible by the same generalized XPath, are all detail pages, or they all have (almost) equal set of generalized XPaths on them. This is because the objects accessible from the same list structure are logically of the same type (typically, they are all products, or all of them are product domains with list of products).

Sometimes, children of the same parent in the DAG do not have exactly the same set of generalized XPaths. Usually, some of the generalized XPaths are common, and some of them are missing on few of them. Pagination of the list is usually the reason. Some product domains are large and require paginated list of products and some are so small, that all products can fit into one page, therefore pagination links list and its generalized XPath are missing.

If there are only two pagination pages in product domain, there is only one XPath in its own cluster of similar XPaths and no generalization inside the cluster is possible. In this case the XPath has an extra pair of brackets with a number in it, when compared to corresponding generalized XPath in page's siblings in DAG. The more specific XPath (with extra brackets) can be replaced by more general one from its siblings.

Many times, the parent in DAG has equal (sub)set of generalized XPaths as some of its children, typically they are all lists of products.

The analysis of DAG goes from the root of DAG that is the seed_URL (usually the home page of the e-shop) using breadth-first traverse. In every vertex, we obtain the pairs of generalized XPaths and clusters of vertexes reachable by them. Vertexes in each cluster have computed their own generalized XPaths. The generalized XPaths of vertexes in cluster are unified with each other and possibly with parent vertex, if they have at least half of generalized XPath unifiable.

Next, a hash of the set of its generalized XPaths is computed and stored in each vertex. If the vertex corresponds to detail page, the hash's value is set to 0.

Finally, the generalized crawling graph is created. This is done by unification of the vertexes having the same hash. This graph has usually at least one edge heading to the same vertex.

Consider the DAG on Figure 2. The analysis starts with seed_URL http://e-shop.sk. Since it has only one outgoing edge, no generalization is possible, and the analysis goeas to vertex http://e-shop.sk/tables. This vertex has two outgoing edges, but not unifiable, so it has two pairs of generalized XPaths and set of vertexes reachable by them: {<//li[1]/a, {http://e-shop.sk/tablet_1}>} and {<//*[class="pagination"]/span[1]/a, {http://e-shop.sk/tablets/page2}>}. The first pair has detail page in the cluster, so the analysis continues with the second pair. Vertex http://e-shop.sk/tablets/page2 has only one pair of generalized XPath and set of vertexes reachable by it: {<**//li/a**, {http://e-shop.sk/tablet_11, http://e-shop.sk/tablet_12}>}. Then we try to unify this pair with pairs in parent vertex, which is successful resulting in pairs for both vertexes: {<**//li/a**, {http://e-shop.sk/tablet_1, http://e-shop.sk/tablet_11, http://e-shop.sk/tablet_12}>} and {<//*[class="pagination"]/span[1]/a, {http://e-shop.sk/tablets/page2}>}. Finally each vertex computes hash of its generalized XPaths. The clusters in pairs are replaced by the hash of their representatives which creates the edges of the final generalized crawling graph as depicted in Fig. 4.

The final generalized crawling graph is stored as a configuration for the next focused crawlings of the e-shop.

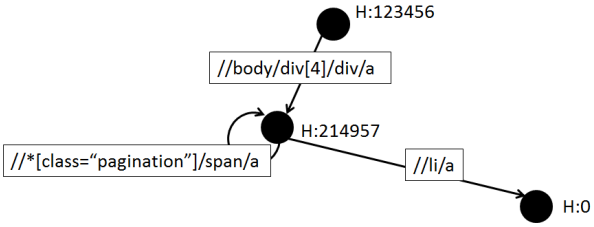

Figure 4: Generalized XPaths graph obtained from the graph in Fig. 2.

3.4 Crawling with Generalized Crawling Graph

Having the generalized crawling graph, the traversal trough the e-shop uses it as a finite-state automaton. Crawling starts at start state of the automaton and the seed_URL. On each downloaded page, all generalized XPaths of the outgoing edges are computed. Found URLs are scheduled to be downloaded together with the state on the end of the corresponding edge.

4 Conclusions

We have tested our approach on two e-shops so far: peazenkyshop.sk and rajdazdnikov.sk. The results of our experiments are in the table below:

Table 1. Number of all pages vs. number of pages downloaded by crawling with generalized crawling graph

e-shop	#products	#pages	#downloaded
penazenkyshop.sk	512	2730	1966
rajdazdnikov.sk	265	303	290

We can see that number of downloaded pages decreased to 72% resp. 96% in our tests. The tests showed that focused crawling is faster than initial crawling and the same set of detail pages was downloaded, which is our goal.

We showed that our automatic creation of crawling strategy is sufficient to eliminate irrelevant pages and download all detail pages.

This work was supported by the Agency of the Slovak Ministry of Education for the Structural Funds of the EU, under project CeZIS, ITMS: 26220220158

References

[1] Project Kapsa web page: http://kapsa.sk/
[2] P. Gurský, M. Vereščák, *Extrakcia štruktúrovaných objektov z webových portálov na pár klikov*, WIKT & DaZ, ISBN 978-80-227-4619-9, pp.225-228, 2016
[3] Crawer4j: Open source web crawler for Java, available on: https://github.com/yasserg/crawler4j
[4] Y. Uemura, T. Itokawa, T. Kitasuka, M. Aritsugi: An Effectively Focused Crawling System. Innovations in Intell. Machines – 2, SCI 376, Springer, pp. 61–76., 2012
[5] H. Liu, J. Janssen, E. Milios: Using HMM to learn user browsing patterns for focused Web crawling. Data & Knowledge Engineering 59, Elsevier, pp. 270–291, 2006
[6] S. Batsakis, E.G.M. Petrakis, E. Milios: Improving the performance of focused web crawlers. Data & Knowledge Engineering 68, Elsevier, pp. 1001-1013, 2009
[7] M.M.G. Farag, S. Lee, E.A. Fox: Focused crawler for events. International Journal on Digital Libraries, DOI 10.1007/s00799-016-0207-1, pp 1–17, 2017
[8] J.A. Torkestani: An adaptive focused Web crawling algorithm based on learning automata. Applied Intelligence, Volume 37, Issue 4, pp 586–601, 2012
[9] A. Patel, N. Schmidt: Application of structured document parsing to focused web crawling. Computer Standards & Interfaces 33, Elsevier, DOI 10.1016/j.csi.2010.08.002, pp. 325–331, 2011
[10] K. S. Kim, K. Y. Kim, K. H. Lee, T. K. Kim, W. S. Cho: Design and Implementation of Web Crawler Based on Dynamic Web Collection Cycle. Computer Standards & Interfaces, Volume 33, Issue 3, pp. 325-331, 2011
[11] G. Gouriten, S. Maniu, P. Senellart: Scalable, Generic, and Adaptive Systems for Focused Crawling. Proceedings of the 25th ACM conference on Hypertext and social media, ISBN: 978-1-4503-2954-5, pp. 35-45, 2014
[12] M. Diligenti, F. Coetzee, S. Lawrence, C. Giles, M. Gori: Focused crawling using context graphs. Proceedings of VLDB, pp. 527–534., 2000