Web crawler

Timotej Kovač

I. Introduction

In this paper we present the structure, challenges and results of a web crawler implementation in Java. It was used to crawl the *.gov.si websites and has fetched 50.000 web pages in total. It uses some support libraries mostly for handling the robots.txt files, and is able to simulate a web browser and so running JavaScript.

II. Web crawler

A. Dependencies

For the implementation of the web crawler we used Java and the following support libraries:

- Jauntium (which uses Jaunt and Selenium libraries) in order to retrieve web pages and process them using a browser driver and so we could also process JavaScript code [1],
- RobotsTxt which was used in order to process the robots.txt files for the domains that were crawled [2],
- PostgreSQL which was used to connect and modify the postgres database [3].

Some other libraries have also been included mainly because the above mentioned ones depended on them.

B. Implementation

Our implementation consisted of:

The frontier which is a list that consists of URLs that have yet to be visited. The initial frontier consists of the four starting URLs which are gov.si, evem.gov.si, e-prostor.gov.si and e-uprava.gov.si. It mainly behaves as a FIFO queue, but because of the delay limitations that the crawler must obey the process of removing an URL starts as FIFO but skips every record that is not permitted to be visited at this time.

The crawling process begins with the initialization of multiple workers on separate threads which then start the process of crawling. First they gain access to a locked frontier from where they take an available URL that has not yet been processed. Then they update the next permitted window when the domain can be accessed again and proceed with the link processing.

First the program tries to send a HEAD request to get the content type of the target URL. Because this part of the code offers a much more low level response then when using Jauntium the following redirects must be followed which happens in recursion. After that the robots.txt file is fetched if the web page is on a never before visited domain. Every URL is here also checked if it has been already crawled and if so the algorithm ends. If not the data type is extracted from the head of HEAD request or the extension of the URL itself.

The next step is left to the Jauntium to visit the website and it to finish rendering the web page. The permitted time for fetching is set to 5000 ms and the time to finish the loading of the web site is another 3000 ms.

After that the text content of the website is processed so that all of the white space characters are removed. If this content matches any other web site in the database the algorithm ends. If not the websites record in the database is updated with the retrieved information.

The next step are the links and images extraction. Links are extracted from the website and are added to the frontier if multiple conditions are met. Some of these are that the URL is a valid URL and not a JavaScript code or anything else, that we haven't visited it yet, that we can visit it based on the robots.txt permissions and so on. Images are extracted as well and only their source address is added to the database.

After that all of the links that have been extracted are added to the database as well and the crawler thread starts the process again for the next link that is available in the frontier.

C. Parameters

The crawler implementation does not require any parameters to be set to be able to perform web crawling on the *.gov.si domains. There are constants though that can be changed if the user want's to alter the programs result. These are:

- URL_* strings that define the starting domains that the crawler should start with. Default ones are the gov.si ones.
- THREAD_COUNT that specifies the number of threads the program will use. It makes sense that this number is at least the same as the number of unique domain names a program will be crawling.
- DELAY specifies the default delay which is used if there isn't a robots.txt file delay specified for that website.
- MAX_LINKS is used to tell the program when it should stop crawling.
- USER_AGENT specifies the name of the crawler that will be seen to website servers.

D. Issues

During the implementation of this solution we had many problems. It took us a lot of time to make this solution robust to various problems that the crawler encountered while visiting web sites.

One of the major ones was the retrieval of robots.txt file with robots library. Though simple library the URL argument that it requires must be exact. It does not support any redirects and gov.si websites sometimes start with 'www' and sometimes they don't. So we had to check 5 different variants of the domain URL (with added 'https', 'http', 'www', etc.). And even if we got some content back this content was often a genuine website (for instance 'arsq.gov.si') .

The next major thing was the inconsistency of web pages responses to HEAD requests, GET requests, filled out content-type (and other) fields, masking of binary data in URLs that didn't end with any extension, requests for certificates and redirects to completely different pages which started in URLs with their domain. This caused most of the effort being spent on implementing a robust way of handling all of the exceptions which appeared. Sometimes the website would be fetched but the response part of the browser object would be null. Other times just getting the list of the available attributes of an HTML element would cause an exception which shouldn't have happened.

Regardless the whole implementation has kept a great design even though multiple exceptions had to be made in order to make the crawler robust.

E. Data analysis

As the result of this crawler working on 6 threads it has taken him 13 hours and 13 minutes to crawl thought 50.000 websites and retrieve their content. It has crawled though 150 unique gov.si domains and gathered the following amount of content and information:

METRIC	# OF WEB PAGES
site	150
page	50.002
binary page	284
...pptx	29
...docx	5
...pdf	204
...doc	10
...ppt	36
duplicate page	10.110
image	339.688

Table I
STATISTICS OF THE WEB PAGES CRAWLED

As it can be seen from table I there were many unique sites visited. There were quite a lot of duplicate pages detected.

Also the number of binary pages was quite low. This was mainly because our previous version of the crawler tended to follow many binary pages that weren't desired. That mainly included images and sounds. And because of this the actual retrieval of web pages contents was very low as majority of them were just images and so no data was saved. An improvement was made to reject these pages from the frontier but the condition wasn't correctly defined and so all of the binary pages that ended in some kind of a binary extension were kept out of the frontier. That was why that number was so low as only the URLs that didn't contain the binary extension in it were passed though.

There were also quite a lot of images fetched. This number would be far smaller if the database structure would be different. But because of that these images are mostly just duplicates like logos of websites. Just the top 5 duplicated images consist of a third of the entries or 117.671 entries.

F. Visualization

The figure 1 shows the connections between specific websites and their domains. Each domain has it's own color and there are four of them. From it we can see that the green podatki.gov.si has the most compact structure where the majority of the web pages don't point anywhere else. The opposite is true for the purple gov.si site which is spread out over the majority of the inner circle. This site points to many others that is why it is in the center and why there are no groups of pages that are packed close together.

III. CONCLUSION

The web crawler was successful in obtaining the content of a large portion of the gov.si websites. It showed the robustness of it's implementation as the targeted websites had a lot of problems and inconsistencies, diverging from standards and mishandling the requests. It proved as a robust crawler although there are some exceptions that it hasn't encounter yet. So more work would had to be done regarding these cases.
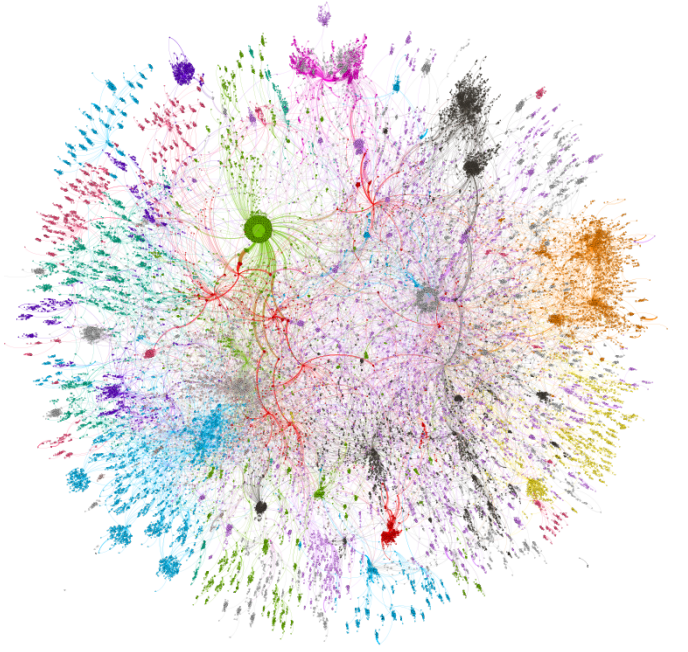

Figure 1. Graph of links between the pages, where the color represents the domain. Purple is gov.si, green is podatki.gov.si, light blue is gisportal.gov.si, black is evem.gov.si and orange is eugo.gov.si.

REFERENCES

[1] "Jauntium Java Browser Automation," Accessable: https://jauntium.com/, 2020, [Accessed: 27. 03. 2020].

[2] "RobotsTxt: Java library to read and parse robots.txt file." Accessable: https://github.com/pandzel/RobotsTxt, 2020, [Accessed: 27. 03. 2020].

[3] "PostgreSQL: The World's Most Advanced Open Source Relational Database," Accessable: https://www.postgresql.org/, 2020, [Accessed: 27. 03. 2020].