Assignment 1: Web crawler

Gal Lindič, Andrej Drofenik, Ožbej Golob

Abstract

*A web crawler crawls across the web to find and index pages for search engines. In this paper, we implemented a web crawler that only crawls *.gov.si web sites. We ran our crawler for approximately 50 hours and were able to crawl through 49498 websites which contained 21988 binary files and 163273 images.*

I. Introduction

Search engines are the gateway of easy-access information, but web crawlers play a crucial role in rounding up online content. Plus, they are essential to your search engine optimization (SEO) strategy. A web crawler is a digital bot that crawls across the World Wide Web to find and index pages for search engines. Search engines crawl or visit sites by passing between the links on pages. They're always looking for discoverable links on pages and jotting them down on their map once they understand their features. Web crawlers, while they're on the page, gather information about the page like the copy and meta tags.

In this paper, we implemented a web crawler that crawls through Slovenian government websites.

II. Implementation

A. Web crawler implementation

We implemented a web crawler that only crawls **.gov.si* web sites. The main seeds are *gov.si, evem.gov.si, e-uprava.gov.si,* and *e-prostor.gov.si.* For the implementation, we chose the Python programming language.

We initialize the crawler with parameter *–initFrontier=1.* Function *init_frontier()* processes the *robots.txt* file and sitemap, establishes the connection with the database, and inserts top-level domains into the frontier. If a *robots.txt* file exists, we process it in order to extract the pages where the crawler is not allowed. After the file is processed, the forbidden pages are saved into the database. Before any new page is fetched by the crawler, we consult the database for the forbidden pages on that specific domain. In case the page forbids the crawler from entering, the page is not fetched by the crawler.

The crawler is implemented with multiple workers that retrieve different web pages in parallel. The number of workers is passed as a parameter when starting the crawler. Each worker gets a new URL to crawl from the frontier. First, a request is sent to the web page and if a response is received (no exceptions were thrown) we can further process the page. Then we check if the returned response is an HTML web page or a binary file. We achieve this by checking the response header. HTML page responses have content type of *text/html* in the header, while binary files usually have *application/x-binary*, etc. Now that we could distinguish an HTML page from a file we could start parsing HTML content. We extract the content from the response and pass it to *beautiful soup* module. From this we extract links (<a> href tags), images (<img> src tags) and *onclick* javascript events. Links are first processed, canonicalized, and then cross-checked with the URLs in the database. We do this in case there is already an entry with the same canonicalized URL. In such cases, we only insert a new link entry in the link table. If the processed link was never seen before we insert it into the database as a frontier so that it can be parsed later. The *onclick* events need to be processed in a bit more complex way than a normal *href* link. We need the *webdriver* to simulate the click event and then extract the URL, which is processed in the same way as a normal *href* link. The crawler also checks for duplicate HTML content pages (with different URLs). This is done by crosschecking each hash in the database with the one we just hashed. If there is a match, the current page is marked as a duplicate.

The crawler can access the same domain only once in 5 seconds. In the database, we created a table called *scheduler*. When a new request was sent to a specific domain, this domain was inserted into the scheduler. We also created a special *manager* thread, that removed domains from the scheduler after 5 seconds. When getting the next seed from the frontier, we selected only the websites whose domain was not in the scheduler.

Whenever the response contains a binary file instead of an HTML file, the file must be processed differently. Since we cannot parse the file like an HTML page, we store its URL request in the database with the appropriate suffix. By default, the crawler does not save the files themselves on the hard drive. However, if the user chooses to save them, they are placed into separate folders depending on the site that provided them.

B. Frontier implementation

We implemented the frontier with the help of the database. The frontier follows the First In First Out (FIFO) approach. When the crawler is processing a page and encounters a link or an onclick event, they are stored in the database with *page_type_code* = FRONTIER. As soon as the link to another page is found, it is inserted into the database. This means that the link that is found earlier has a lower ID than other links, which enables the FIFO approach. This property enabled us to process pages from frontier with the breadth-first strategy. We get the next seed from frontier with the function *get_next_seed()*, which reads the page with lowest id with *page_type_code* = FRONTIER and updates the *page_type_code* to HTML. To prevent different threads from accessing the same record simultaneously, we use *threading.Lock()* functions *acquire()* and *release()*.

C. Implementation problems

The first problem we tackled was how to implement the frontier inside the page table in the database. We had to logistically figure out how to handle database entries that will represent frontier entries and entries which will represent *HTML/binary* content. We had a problem with implementing a crawling delay. We wanted to use the *time.sleep()* function, but we encountered problems with multiple thread timings. The requests were sent in 5-second intervals, but all the threads were sending requests simultaneously. We solved this problem by implementing the *scheduler* table that is preventing requests from accessing the same domain in less than 5-second intervals. The requests are now sent in 5-second intervals. Another problem we faced was how to handle *onclick* events. Since this is a javascript action, we couldn't just process it like a regular *href* link. We had some trouble with how to trigger the *webdriver* to click on the element which has the event. The problems had arisen when doing it in headless mode and simulating clicking, window opening/closing. We were able to bypass these troubles and have successfully implemented *onclick* event handler.

III. Results

A. Statistics

We ran our crawler for approximately 50 hours and were able to crawl through 49498 websites which contained 21988 binary files and 163273 images. Figure 1 shows the *crawldb* statistics.

Seed site statistics:

Crawldb

Data type	Number
Websites	30239
Domains	4
Duplicates	1283
Binaries	15263
Images	16504
Images/Website	0.6

Binaries

Data type	Number
PDF	8518
DOC	3720
DOCX	2420
XLS	1073
CSV	444
ZIP	163
MP4	0
PPT	43
PPTX	38
XSD	10
MP3	11

All site statistics:

Crawldb

Data type	Number
Websites	49498
Domains	269
Duplicates	2348
Binaries	21988
Images	163273
Images/Website	3.3

Binaries

Data type	Number
PDF	11421
DOC	4532
DOCX	2896
XLS	1429
CSV	494
ZIP	351
MP4	104
PPT	69
PPTX	49
XSD	22
MP3	22

Figure 1. Crawldb statistics.

B. Visualization

We visualized our results with the D3js library. Figures 2 and 3 display the links between a selected number of pages from our database. Figure 4 displays all of the links inside our database.

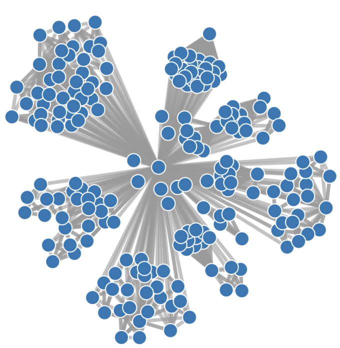

Figure 2. Links that include 'e-uprava.gov.si/podrocja/drzava-druzba' in their URL.

IV. Conclusion

In this paper, we implemented a web crawler that only crawls **.gov.si* web sites. The crawler is implemented in Python and employs multiple workers to retrieve pages in parallel. Each worker gets a seed from the frontier and processes it to extract links, images, and *onclick* events. They are then added to the frontier and processed later. We ran our crawler for approximately 50 hours and were able to crawl through 49498 websites which contained 21988 binary files and 163273 images.

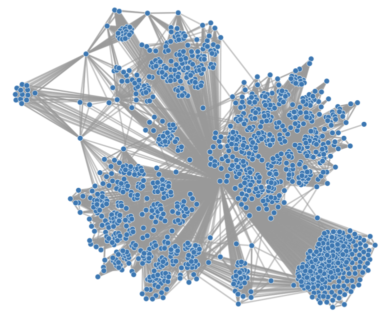

Figure 3. Links that include 'www.gov.si/drzavni-organi' in their URL.

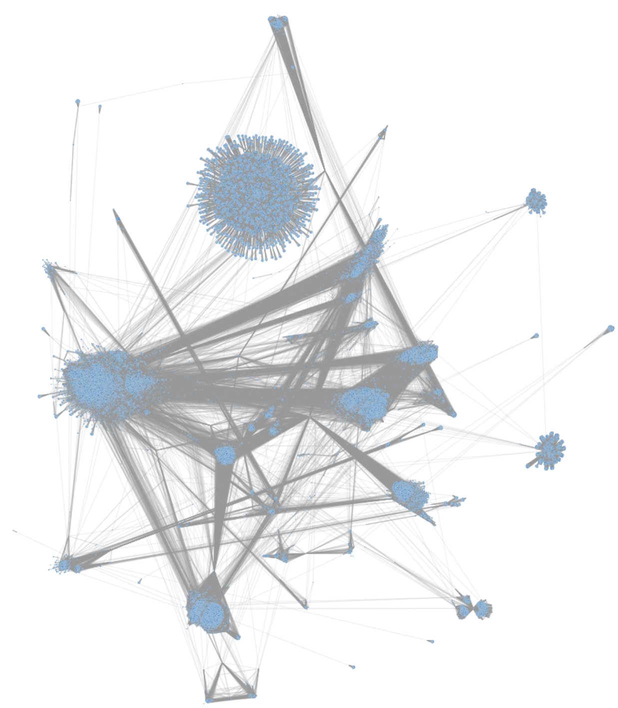

Figure 4. The entire database of 'gov.si' pages across all domains.