Navigating the government website network using parallel breadth-first crawler

Web Information Extraction and Retrieval 2018/19, Faculty of Computer and Information Science, University of Ljubljana

Matej Klemen, Andraž Povše, Jaka Stavanja

***Abstract*—BOII**

****
****
**********/##**********
**********%%%%**********
//(*********%&&%%%%%%**********
(%%%%(*/##%&&%%%%%&&%%*******
****#//%%%%&&&&%%&&&&%&&%**********,,,
**********/%%&&&&%%%%***,,,,,,,,,,
**,,,,*,,,,,,,,,#%%&&&&&&%%%(,,,,,,,,,,
******,,,*,,,,,,,/#%&&&&&&&&%*,,,,,,,,,,
,,,,,,,,,,,,,,,,,,*(%&&&&&&&&***,,,,,,,,
,,,,,,,,,,,,,,,,,,/,/%&&&&&&%(,,,,,,,,,
,,,,,,,,,,,,,,,,,,,,(*,%%&/%&%(**,,,,,,
,,,,,,,,,,,,,,,,,,,,,,**#&%&&&%%***,,,,,
,,,,,,,*/,,,,,,,,,,,,,,,(#&%&%%%/*,,,,,
,,,,*///(/(/,,,,,,,,,,,,,/#((((((((((**
***/(((/(((((((**,,,,,,,((((((/(((###**
(((((/#((#(((******/(#((((((####/*
*(((##########/********(###((((####//
//(((((#(#((((#//******/(((#(((((####//

I. Introduction

A web crawler is a program that visits web pages and downloads them for some purpose. It begins with some amount of seed URLs, which are put into a frontier. The crawler then selects a web page from frontier according to some strategy, visits the web page and extracts the web page's content and outgoing links. The outgoing links are put into the frontier and the content is used for some purpose or stored. This process is then repeated until a certain stopping criteria is reached (e.g. the frontier being empty) [1]. The crawling can also be a continuous process ("without end"), but we do not study these in this project.

In this project we implement a web crawler that crawls Slovenian government websites. We are given a set of 4 government websites and choose additional 5 of them and proclaim them as seed websites, from which we start the crawling. While crawling these websites, our crawler only follows links to other Slovenian government websites. The crawler crawls these websites, downloads images and files from them and stores them into a database. The crawler visits new links in a breadth-first fashion. We also implement a method to detect near-duplicate websites based on their content. For this, we use a locality-sensitive hashing algorithm based on min-hashing [2].

The rest of the report is structured as follows. In chapter **TODO** we present **TODO**. In chapter **TODO** we present **TODO** ... In chapter IV we present how the database is built.

II. Crawler

In this section we present the architecture of the crawler. The full source code is available online [1].

The crawler consists of multiple components: link, sitemap and robots parser, database manager (and the database itself), content deduplicator and a core unit which uses the other components and adds on top of them to produce a parallel crawler that uses breadth first search to navigate the websites.

It takes 4 parameters in total. These are seed pages, maximum number of workers, maximum crawling depth and a flag, determining whether to get files. The first two parameters are self-explanatory. The maximum crawling depth is optional and specifies at what depth we want our crawler to stop. If this parameter is not provided, the crawler will stop when there are no more links being produced.

[**TODO: describe each component briefly (mention its role).**]

We used Selenium to make requests, because it also renders the Javascript code on the website. It does, however, not include HTTP status code, which is why we also had to make another request to get that information.

EXPLAIN HOW BREADTH FIRST STRATEGY WORKS
The crawler uses (parallel) breadth-first traversal to crawl the websites. Figure 1 shows an example of how work is divided among workers in our implementation.

The master (the initial process) keeps a structure with visited links and links that still need to be visited (frontier). On each level, the frontier is emptied and the links are divided among workers as evenly as possible. Each worker keeps a local copy of its visited websites and a local copy of its newly produced links. Each worker also checks if a newly visited website is present in the global visited structure, but it does not add new links into it as we want to keep the synchronization between threads to a minimum. We must wait for all the workers to finish so that breadth-first strategy is ensured. After they are all done, the master collects newly collected links from workers and checks for duplicate links. It also marks the links from current level as visited. This process is repeated with the newly collected links, which represent the next level of search process.

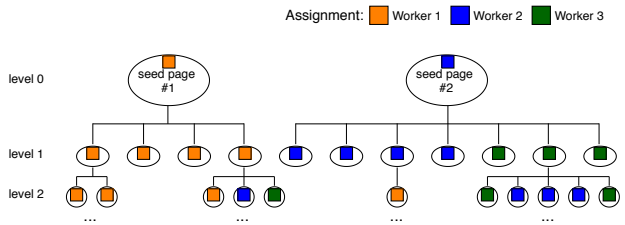

Figure 1. How the work is divided among workers in our implementation of parallel breadth-first traversal. The process starts with a set of seed pages, in this case 2. Note that even though we have 3 workers available, we only actually create 2 workers for level 0 as the 3rd one would not receive any work. Workers then gather new outgoing links, which get collected by the master. The work is divided among workers again every time a new level is started.

We did not use the database as a frontier storage —— We stored our frontier in the program itself.

[1] https://github.com/matejklemen/govrilovic

III. Robots and Sitemaps

Key element of the crawler is also reading the 'robots.txt' file and sitemap. Based on the robots file, we know whether we can visit a certain link or whether it is not allowed for a crawler to visit that link. When we read a sitemap, we add all the URLs mentioned in it, into the frontier.

IV. Database

The database was build inside a Docker container. We used the same schema, with one minor change, which is adding a column 'lsh_hash' into the page table. The column stores the value that LSH hash produced for the content of that website. Access to database is made by each worker when he visits a new page. The database can handle multiple workers making read and write operations by proper locking of the operations.

For viewing the results and current status in the database, we also used Adminer application, which also runs as a docker container and is then accessable on localhost.

V. Link

VI. Image and files extracting

@JAKA — commnet on the issue with absolute path, very interesting:D

We stored images and files into the file system, since it gives a nicer overview of the gathered information. This was made possible by creating a folder for each new site in which we stored the information. We then stored the location of the file/image into the provided data field in the database.

VII. LSH deduplication

We implemented LSH algorithm, which is being used to compare hashed content of different websites.

How it works

For the set of words, we hash it on, we used top 1000 Slovenian words. Or maybe triples of characters?

VIII. Results - 2 seed URLs

For the first run of the crawler, we used 2 seed URLs, which were "http://evem.gov.si"and "http://e-prostor.gov.si". We set the maximum crawl depth to None, which means it will run untill there are no more links produced. The maximum number of workers was **NUM OF WORKERS WE USED**.

The crawling process took XXXX minutes in total and retrieved XXX pages,links, images...

For the sites that are given in the instructions' seed list and also for the whole crawldb together (for both separately) report general statistics of crawldb (number of sites, number of web pages, number of duplicates, number of binary documents by type, number of images, average number of images per web page, ...). Visualize links and include images into the report. If the network is too big, take only a portion of it. Use visualization libraries such as D3js, visjs, sigmajs or gephi.

IX. Results - 9 seed URLs

The second run consisted of 9 seed urls. For the extra 5 urls of our choice, we have choosen "http://www.mz.gov.si/", "http://www.mnz.gov.si/","http://www.up.gov.si/", "http://www.ti.gov.si/" and "http://www.mf.gov.si/".

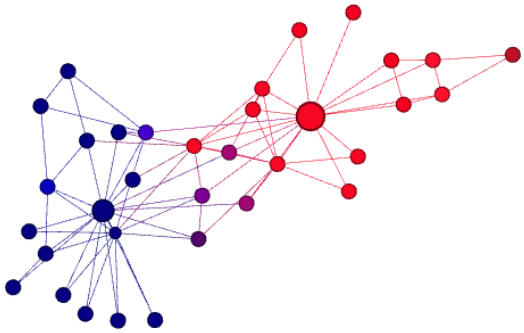

Figure 2. Example of an image.

Table I
Example of a table.

Richard Karstark	0.013661
Jon Arryn	0.012869
Joyeuse Erenford	0.012869
Trystane Martell	0.010761
Willen Lannister	0.010684
Martyn Lannister	0.010684
Robb Stark	0.010304
Joffrey Baratheon	0.009875
Master Caleotte	0.009495
Lysa Arryn	0.009383

References

[1] C. D. Manning, P. Raghavan, and H. Schütze, *Introduction to Information Retrieval*. Cambridge University Press, 2008.

[2] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of the 25th International Conference on Very Large Data Bases*, ser. VLDB '99. Morgan Kaufmann Publishers Inc., 1999, pp. 518–529.