Information Retrieval

Project report

Koltsov Mikhail
Kravtsun Andrey

1. INTRODUCTION

There is a problem with searching shoes when you need to buy one. We decided to implement a search engine for simplifying this task so you won't need to search on your favourite sites selling shoes and some clothes stuff.

Desired functionality: given a query with shoes description, like *"туфли спортивные с закрытым носом"*, show only relevant e-shop pages.

Features (may change later during development):

(1) system understands queries like *"туфли с закрытым носом"* and *"туфли на танкетке"* as different, and results in precisely relevant pages with user-specified filters applied;
(2) filtering on preferred e-shops;
(3) filtering and sorting by price.

2. DATA ACQUISITION

2.1. Source code

We have our repository at GitHub[1]. Code is in Python 3.5 and package requirements are stored in `requirements.txt`

2.2. Architecture

We aimed for the distributed crawling from the start, so our architecture was largely influenced by it. Also, we wanted components to be abstracted from each other. For those interested package diagrams were generated and put into docs[2].

2.2.1. Main package.

- `Page`[3] - entity that represents one page. It knows whether the page was downloaded, what are its children (sites that this page refers to inside its domain) and its last fetch time.
- `DomainQueue`[4] - shared queue of seed domains. All workers read from it. Each worker gets one domain, removes it from the queue and starts breadth-first search (BFS). Access to this queue is not protected by mutexes, because it is very rare: one worker spends most of its time doing BFS rather than reading new domains. So, we treat the situation when two workers are reading from DomainQueue simultaneously as low-probable.
- `CrawlQueue`[5] - worker-private queue of pages. Each worker performs Breadth-First Search from the received seed domain. Each worker operates in Firewall-mode: only crawls pages from one domain at a time.

2.2.2. Package `'storage'`.

- `BasicStorage`[6] - abstraction of storage. It has its implementations as:
- `LocalStorage` and
- `GDriveStorage`.

Along with page, meta-information is stored: URL, size, path to file. Storage also filters duplicate pages by content using MD5 hash (it is sufficiently long to not bother about hash collisions, also it is reasonably fast).

2.2.3. Package `'robots'`.

- `RobotsProvider`[7] - static entity that knows about `robots.txt` of certain domains. Functionality of `RobotsProvider` is implemented through usage of:
- `MemberGroupData`, which corresponds to rule for specific member group at `robots.txt`), and
- `RobotsParser` that generally converts raw text from `robots.txt` into viable parameters.

2.3. Politeness

Crawlers can retrieve data much quicker than human searchers, so they can influence badly on a site performance. In order not to bother any site with our crawler's activity and have no impact on its performance and throughput we use several commonly used guidelines.

2.3.1. Robots exclusion protocol.

- Our crawler does not query a page's children if it contains `nofollow` word in tag's `<meta name="robots">` key `content`.
- We neither fetch nor store pages with `noindex` in the tag mentioned above.

2.3.2. Server-wise exclusion.

- We do not request any pages declared under `Disallow` directive in `robots.txt` for common member group (`user-agent: *`).
- We respect site performance diversity and use `'crawl-delay'` parameter in `robots.txt` for setting pause

[1] https://github.com/ItsLastDay/FindMyShoes
[2] https://github.com/ItsLastDay/FindMyShoes/tree/master/docs
[3] https://github.com/ItsLastDay/FindMyShoes/blob/master/src/crawler/page.py#L19
[4] https://github.com/ItsLastDay/FindMyShoes/blob/master/src/crawler/queues.py#L9
[5] https://github.com/ItsLastDay/FindMyShoes/blob/master/src/crawler/queues.py#L51
[6] https://github.com/ItsLastDay/FindMyShoes/blob/master/src/crawler/storage/basic_storage.py#L9
[7] https://github.com/ItsLastDay/FindMyShoes/blob/master/src/crawler/robots/provider.py#L13

IR'17, Fall 2017, Saint-Petersburg, RU
.

between consecutive queries. If it is not present, we pause for 1 second between queries.

2.3.3. Crawler identification.

- Our crawler declares itself as `'findmyshoes_bot'` in HTTP requests to server.

2.4. Flow

There can be many participating processes in our crawling. Each computer must have a list of seed domains assigned and placed into `domain_queue.txt` file next to `crawler.py`. One can start arbitrary number of workers with `python3 crawler.py`. Then, firewall strategy will took place and every domain will be assigned to one specific worker (so that one worker still can crawl several domains if it's fast enough).

Workers have an algorithm, as follows:

(1) read line from `domain_queue.txt` and erase that line from the queue. If the queue is empty, exit.
(2) do a BFS on this domain, maintaining delays and storing pages with `storage.BasicStorage` instance. Each page is checked whether it can be crawled or stored.
(3) return to step 1

2.5. Problems

- Google Drive for storing pages was overall a bad idea. It works slower than storing pages locally, because each store is >= 1 HTTP-requests. Also, GDrive API has rate limits. We managed to store about 9.000 pages in 6 hours, whereas storing pages locally could achieve > 100.000 in one night run.
- Moreover, there were problems with storing Russian letters through GDrive API, so we stored base64-encoded pages. And the API seems not to be well-documented itself.
- We tried to use standard solutions for `robots.txt` parser, like `urllib.robotparser`. But they surpisingly failed even on first test page[8], which is allowed to be crawled in corresponding `robots.txt`[9] but not-fetchable as those parsers say.

2.6. Results

At night 115793 pages were downloaded while running 3 crawlers on several different domains, as our expert colleague proposed:

- https://www.bonprix.ru/
- https://www.lamoda.ru/
- http://www.asos.com/
- http://www.ecco-shoes.ru/
- https://respect-shoes.ru/
- https://ru.antoniobiaggi.com/
- http://www.rieker.com/russisch
- https://www.net-a-porter.com/ru/en/

Our crawler proves itself to be polite and distributed. Besides, we crawled more than 100k documents. Therefore, we hope for an excellent mark.

[8] https://www.google.com/maps/reserve/partners
[9] https://www.google.com/robots.txt