Offline Environment for Focused Crawler Evaluation

Tomáš Gogár

Dept. of Cybernetics , Czech Technical University, Karlovo namesti 13, 121 35 Prague, Czech Republic

gogartom@fel.cvut.cz

Abstract. *Focused crawling is a method for automatic exploration of the World Wide Web focusing on regions, which are relevant to a given topic. Since focused crawlers are often used to collect data for vertical search engines, they received special attention in research community. Unfortunately, dynamic nature of the internet complicates fair comparisons of focused crawlers, because crawling experiments are not repeatable in the longer term. Researchers try to mitigate the impact of a changing environment by running tested crawlers simultaneously. However, this requirement makes extensive comparisons almost impossible. In this work we try to solve this issue by proposing a system for offline evaluation, which allows developers to consistently test their algorithms on fixed subgraph of the internet. This subgraph consists of 3.5 billion web pages and 128.7 billion links. Our experiments with baseline focused crawlers showed that the performance achieved in the offline environment correlates with the performance achieved online. This suggests that the proposed environment, which allows us to carry out repeatable experiments, can be used for focused crawler comparisons.*

Keywords

Focused crawling, Topical crawling, Vertical search, Evaluation, Common Crawl

1. Introduction

An increasing amount of data that are becoming available on the World Wide Web results in a growing need for tools, which effectively organize and search information on the internet. Generic search engines have become indispensable tools for browsing and searching the internet. These engines gather data by using crawlers, which exploit the graph structure of the web to automatically download billions of web pages that are further processed by indexers.

Besides generic search engines (such as Google, Bing, etc.), vertical search engines started to appear recently. Vertical search engines are focused only on some specific domain and they can benefit from such limited scope (usually they can support domain-specific search parameters and therefore provide results with greater precision).

On the other hand vertical search engines still need some input data and therefore they also need to visit topic-relevant web pages with a crawler. But since the relevant web pages usually form only a negligible part of the internet, we need a crawler, which avoids the regions of the web, which are irrelevant with respect to the given domain. For this reason focused crawlers received special attention in the web search community in the last decade. These crawlers prioritize the crawl frontier so that they discover as many relevant pages as possible, while keeping the number of irrelevant pages minimized.

There are many focused crawling algorithms described in literature (some of which are further described in Section 2), each proposing different strategy for prioritizing the crawl frontier. However, it is still unclear, which of these algorithms is the best choice for some specific vertical search engine, because there are no standard testing tasks, which would allow systematical comparison.

Each crawling task is defined by *topic definition*, the associated *relevancy function* and initial *seed URLs*. However, even if we precisely define the testing task, other authors will not be able to compare their results in the longer term, because crawlers operate above the internet and its nature makes consistent comparison difficult. The main reasons are:

- **Internet is dynamic** Content and structure of websites changes very quickly, which prevents consistent repetition of experiments.
- **Internet is huge** Current generic search engines are expected to index about 50 billions of web pages. The whole internet (including the non-indexable part often referred to as the deep net) should in fact be much larger, which makes it difficult to estimate the recall of focused crawling algorithms.

Together with a lack of open-source implementations, this results in a situation where most of the authors cannot compare their results with crawlers, which were proposed a few years ago and they compare with basic best-first-search (BFS) solutions, which are easy to implement [12] [5].

In order to provide developers a way for comparing their work, we propose an offline testing environment, which use a large precrawled subgraph of the internet (induced by

3.5 billions web pages, which are interlinked by 128.7 billion links), which can be easily used for specifying repeatable crawling experiments. The main contributions of this work are:

- **Preprocessed data set for offline crawling** We provide a system for getting original web pages from a static precrawled data set.
- **Nutch plugins** We provide plugins that can be used together with well-established open-source crawler Nutch, so that developers can easily switch between online and offline crawling.
- **Consistency experiments** We show that crawling performance measured on a fixed subgraph of the internet correlates with performance measurements obtained online.

This paper is organized as follows: in Section 2 we briefly summarize most influential focused crawling algorithms and we discuss how they are compared. In Section 3 we describe our offline environment for crawler evaluation. In Section 4 we compare its influence on focused crawlers comparison. Section 6 then summarizes our results.

2. Related work

2.1. Crawling

The crawling process is usually initiated with a set of starting URLs (referred to as seeds) and it explores the web by repeatedly downloading content from newly discovered URLs. Since the number of discovered URLs grows exponentially with the number of downloaded web pages, crawlers need to implement some selection policy in order to decide, which document will be downloaded next. The most simple strategy is breadth-first-search, however, it is usually highly desirable to download the most relevant pages as soon as possible and not waste resources for downloading some random fraction of the internet. Generic crawlers usually prefer important URLs in the sense of PageRank [2][1], while focused crawlers favor thematically relevant web pages.

2.2. Relevancy functions

Focused crawlers usually employ one of the two approaches to estimate topical relevancy of a web page. The first approach is based on cosine similarity between TF-IDF vectors of a web page and a given topic-description vector. The topic-description vector can be extracted as a centroid of sample documents or as a set of keywords provided by human experts [5][12]. Cosine similarity can be easily converted to binary decision by using a predefined similarity threshold. The second approach uses a binary classifier (Naive Bayes, SVM, etc.), which is trained on positive and negative samples [5][8]. The drawback of the second method is that we need a robust classifier for a highly unbalanced classification problem.

2.3. Focused crawling

A focused crawler needs to prioritize the crawl frontier in such a way that it can discover and explore relevant parts of the web. Some approaches are briefly described below. The first focused crawling algorithm Fish Search Algorithm proposed by DeBra et al. in [7] imitates the school of fish, which dies off if it cannot find food (here food means relevant pages and school of fish is a direction of crawling). This approach was than improved in the Shark Search Algorithm by Hersovici et al.[9], where a more precise frontier prioritization was proposed. This approach as well as many others uses best-first-search as an underlying strategy. It means that it predicts the relevancy of unknown links and selects the most promising first. This strategy is based on observation that topically related web pages tend to be clustered close to each other [6]. The same best-first-search strategy was used in [5][4] with the difference that Naive-Bayes classifier trained on ODP[1] data was used.

More sophisticated approaches are able to follow irrelevant links in order to discover new clusters of relevant pages. These algorithms often use backlinks provided by generic search engines to create smaller subgraphs of relevant pages, which lead to relevant clusters. Algorithms then use various heuristics and language models, which can lead the crawl over irrelevant nodes [8][10].

2.4. Crawler evaluation

The following paragraphs summarize performance measures, which are usually used to measure effectiveness of tested crawlers. These measures are often plotted as time series, so that we can capture the dynamic behavior of the tested algorithms. Measures estimating crawling precision are:

- **Harvest rate** It is a percentage (or number) of relevant pages fetched during the crawl. Whether the page is relevant or not is classified by a binary classifier. In order to capture the behavior of the tested crawler, it is often depicted in a cumulative plot [5].
- **Sliding window relevancy** This measure captures the average relevancy of fetched pages in different phases

[1] Open Directory Project (ODP) is an open categorized directory of web pages, see http://www.dmoz.org/

of the crawl. Here the relevancy is a real value (usually between 0 and 1). In order to make results more readable, it is usually plotted as the average relevancy of sliding window. [8][12]

- **Overall topic relevancy** This measure can also be plotted for every phase of the crawl, but unlike the previous one, no sliding window is used and the average is computed from all fetched pages [12].

Estimating recall is very difficult, because we simply do not know the total number of relevant pages on the internet. Srinisavan et al.[12] propose to create small random sample of relevant pages and estimate precision and recall on the samples. However, this approach gives very tentative estimates, since the number of samples is negligible compared to the size of the internet.

The recall measure is usually not needed for crawler comparison because every tested crawler is limited to download the same number of web pages and thus we can compare them using precision measures. However, we have no idea what portion of relevant web the crawlers have downloaded.

2.5. Crawler comparison

Using the measures described above, we can compare different algorithms. However, internet is dynamic and the underlying graph changes over time, so in the longer term the experiments are not repeatable. Researchers are aware of this issue and they solve it by running crawling algorithms simultaneously. However, this requirement makes exhaustive comparisons very difficult, because we need functional implementation of many algorithms. The problem is that most algorithms described in literature lack open-source implementations, their parameters are not precisely described and some of them require complex training with specific data sets (which are not available). As a consequence, absolute majority of works on Focused crawling compare their results with primitive baselines such as unfocused crawlers or very simple best-first-search methods [7][9] (because these methods are well known, easy to implement and do not require complex training).

This results in situations when:

- We do not know what is the state-of-art algorithm of focused crawling.
- Parameter estimation and incremental tests of the same crawling algorithm are complicated.

In this work we try to partially solve this problem by providing a fixed subgraph of the internet that is large enough, so that performance measured on the subgraph corresponds to the performance measured online. This approach brings many advantages such as repeatability of experiments or possibility of estimating recall. However, it should serve just for complementary testing and it should never completely substitute online tests.

Public suffix	Absolute freq.	Relative freq.
.com	2,068,437,976	0.558
.org	224,399,685	0.061
.net	201,192,588	0.054
.de	176,258,358	0.048
.co.uk	111,207,465	0.030
.ru	61,922,226	0.017
.nl	53,187,132	0.014
.pl	52,582,609	0.014
.info	48,803,468	0.013
.fr	47,291,759	0.013

Tab. 1. 10 most frequent public suffixes in 2012 crawl

3. Offline Environment

3.1. Data set

As we mentioned above, we need a fixed set of web pages that is as large as possible. For the purposes of this work we have used data provided by *Common Crawl*[2], particularly their crawl from 2012. *Common Crawl* is a nonprofit organization that provides a large repository of web crawl data to the public. The whole dataset is hosted on Amazon S3 as part of the Amazon Public Datasets program[3], so it can be downloaded for free using HTTP or S3.

Although *Common Crawl* finishes four large crawls every year, we have chosen older crawl from 2012, because it is the largest one it was crawled using breadth-first strategy[4] [11].

This particular crawl consists of 210 terabytes of data, which are divided into 856 thousands ARC files [5]. ARC files are compressed sets of web documents. The whole dataset contains 3.83 billion documents, of which 3.53 billion are of mime-type `text/html` [11]. Table 1 shows 10 most frequent public suffixes in the crawl and we can see that more than half of the documents comes from `.com` domains. The structure of the web graph induced by the crawled pages was thoroughly analyzed by Meusel et al. in [11]. In their work they have identified 128.74 billion links between the crawled pages, they computed that $48 \pm 2.14\%$ pairs of pages have a connecting directed path, which is on average 12.84 ± 0.09 steps long. They have also identified a giant strongly connected component that contains 51.3% of pages. The structure of the web is often depicted using the *bow-tie model* as proposed in [3]. In Figure 1 you can see *bow-tie model* of the 2012 crawl. The model consists of the following components:

[2] http://commoncrawl.org/

[3] http://aws.amazon.com/public-data-sets/

[4] More recent crawls are based on large list of URLs provided by Blekko company (Currently IBM Watson research group).

[5] http://archive.org/web/researcher/ArcFileFormat.php

- **LSCC** is the large strongly connected component (also referred to as core)
- **IN** component contains non-core pages that can reach the core via a directed path
- **OUT** contains non-core pages that are reachable from the core
- **TUBES** are formed by the non-core pages that are reachable from IN pages and that can reach OUT pages
- **TENDRILS** represents pages that are not listed above, but are reachable from IN or that can reach OUT
- **DISCONNECTED** components represent disconnected subgraphs.

Although we have a pretty accurate picture of the structure of the Common Crawl 2012 web graph, we do not know the true structure of the internet. Only the largest search engine providers (such as Google, Microsoft, etc.) can accurately approximate the true structure of the internet. However, our hypothesis is that the Common Crawl dataset is large enough that we can use it for focused crawler evaluation. This hypothesis is further examined in Section 4.

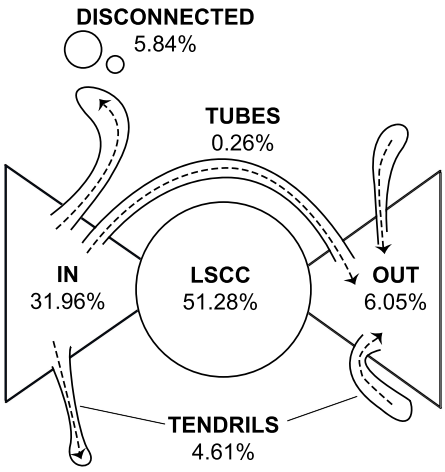

Fig. 1. Bow-tie structure of Common Crawl 2012 (Data adopted from [11]).

3.2. Data preprocessing

We have already mentioned that the original Common Crawl dataset is stored in 856 thousand ARC files in Amazon S3 store. ARC files consist of randomly concatenated ARC records (each record represents a crawled document), which are compressed with gzip. Concatenation of gzipped records is still a legal gzip file. This separation of records allows us to download only a particular document without the need to download the whole ARC file (which takes about 100 MB). However, if we want to download ARC record for particular URL we need to know where it resides. Unfortunately, this is not the case of the Common Crawl 2012 dataset where the ARC records are stored randomly. Therefore, we need to perform a linear search in order to find a web page. Such an approach is highly inefficient and therefore we have preprocessed the dataset and created an index of document positions. We used Amazon EC2 spot instances, went through all ARC files and saved information needed to directly access ARC records for particular URLs. This means that for each URL we stored:

- Name of the ARC file where the document resides (it consists of three parts: segment ID, file date and partition)
- The offset within the ARC file
- And the size of the compressed record

Since this index takes about 300 GB in uncompressed text form, we have indexed it using a NoSQL database so that we can quickly get the location information for each URL from the dataset. Researchers and other developers can download the preprocessed dataset from our webpage [6].

3.3. Architecture Overview

Beside crawl prioritization, robust crawlers need to solve many other problems (such as detection of redirection loops, crawler traps detection, DOM parsing, URL canonicalization, etc.). Therefore, we integrate our work into existing well-established Apache Nutch crawler[7]. Nutch crawler is a production-ready crawler, which uses Apache Hadoop environment, to enable large parallelized crawls. Its modular architecture allows us to adjust its behavior using custom plugins. The crawling lifecycle is depicted in Figure 2a. URLs which should be fetched are passed from CrawlDB to Fetcher, that is responsible for downloading documents, these documents are then parsed, new links are extracted and CrawlDB is updated. Fetcher is implemented as a plugin, thus it can be replaced by our own implementation that downloads documents from the Common Crawl dataset (Figure 2b). Developers of crawling algorithms can then use our plugin to easily switch between online and offline crawling.

The process of fetching document from Common Crawl dataset is depicted in Figure 3. Our fetcher plugin receives request for downloading a document at given URL (1), it asks our database for its position in Amazon S3 (2), then it downloads the compressed ARC file from S3(4), extracts its content and passes it back to the Nutch crawler(6).

[6] All datasets and Nutch plugins are available at github.com/gogartom/Offline-crawling-environment

[7] http://nutch.apache.org/

When the crawler gets the page, it parses the content and extracts all the links. However, some of the links point to URLs, which are not in the dataset. If the crawler later decides to fetch such missing URL, it will get a *404 - Not Found* response. Therefore such responses are much more frequent during offline crawling.

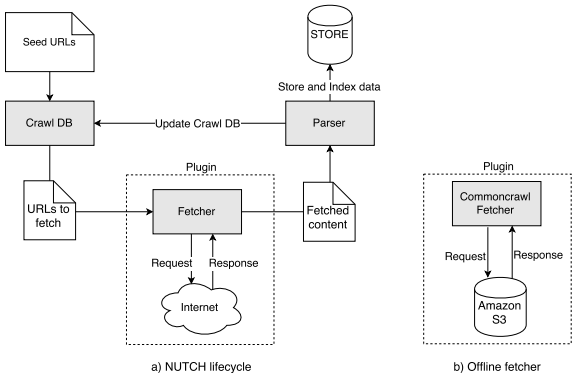

Fig. 2. (a) Basic diagram of the Nutch lifecycle - by default the Fetcher plugin downloads documents from internet. (b) Our fetcher downloads documents from the Commoncrawl dataset. It is easily pluggable into the existing Nutch infrastructure.

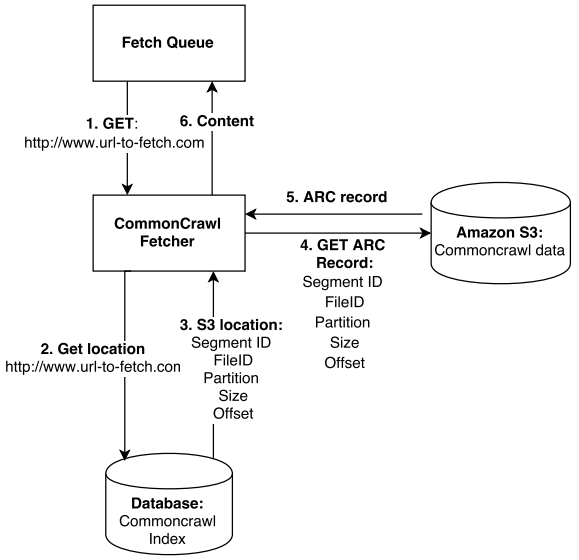

Fig. 3. Common Crawl fetcher: Process of fetching URL from Amazon S3

4. Evaluation

4.1. Algorithms

After preprocessing the dataset and preparing the Nutch fetcher, we examined the influence of offline crawling on measures described in Section 2. For this purposes we implemented two basic focused crawlers, Fish [7] and Shark [9], into Nutch. We have chosen these algorithms for their simplicity and because they are very often used as baselines in the literature. The third algorithm used in our experiments is OPIC algorithm [1], which is a topically unfocused crawling strategy and it is used by default in Nutch.

Topic	Top vector words
Web APIs	api, delete, string, update, type, object, create, file, json, ...
Interior designers	design, interior, bespoke, kitchen, project, designer, ...
Bike shops	bike, bicycle, cyclery, ride, shop, road, mountain, frame, repair, ...
Cooking	recipes, cooking, chicken, healthy, food, chef, salad, cheese, ...
Hospitals	health, medical, hospital, care, center, patient, services, surgery, medicine..
Linux	linux, command, shell, file, directory, files, install, unix, ubuntu, server, ...
History museums	museum, history, exhibit, historic, tours, volunteer, visit, historical, ...
Photographers	photography, photographer, portraits, corporate, commercial, headshots, ...
Pet shelters	shelter, animal, adoption, dogs, pet, adopt, spay, neuter, volunteer, cats, ...
Universities	campus, university, alumni, students, faculty, undergraduate, research, ...

Tab. 2. Topics and their words.

4.2. Crawling tasks

We prepared 10 focused crawling tasks by defining topic descriptions and seed pages. The topic description is a TF-IDF vector, where term frequencies (TF) were extracted from 40 sample web pages and inverse-document-frequencies (IDF) were computed from 370,000 randomly crawled web pages (Table 2 shows topics and words with the highest weight in the vector). In this work we used the cosine distance between the topic vector and a page vector as a measure of relevancy. The seeds URLs were selected in such a way that all of them are reachable online and also present in the offline dataset. We also tried to create crawling tasks of different difficulty by starting some crawls closer to target pages and some crawls further.

4.3. Experiments

After that, for all the predefined tasks and each algorithm, we ran separate crawling experiments online and offline. In every experiment we let the crawler to download 10 thousand pages, hence in total we have fetched 300,000 URLs from internet and the same number from S3. As we have discussed in previous Section, when a crawler fetches URL, which is not in the offline dataset, it gets a *404 - Not Found* response. Since it is not a fault of a crawler that the

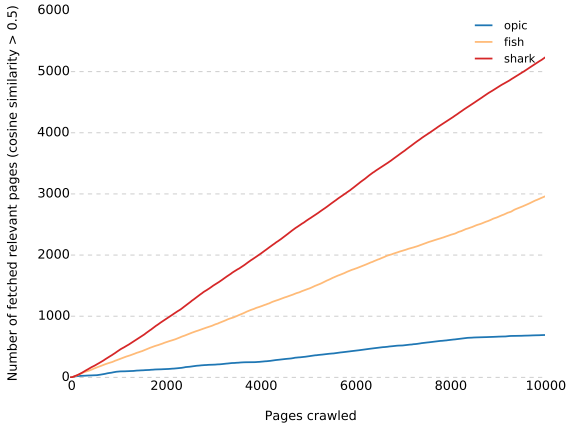

Fig. 4. Internet: Cumulative number of relevant pages fetched during the crawls

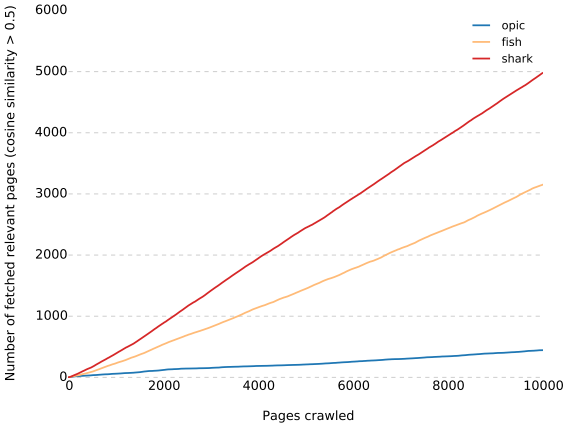

Fig. 5. Offline dataset: Cumulative number of relevant pages fetched during the crawls

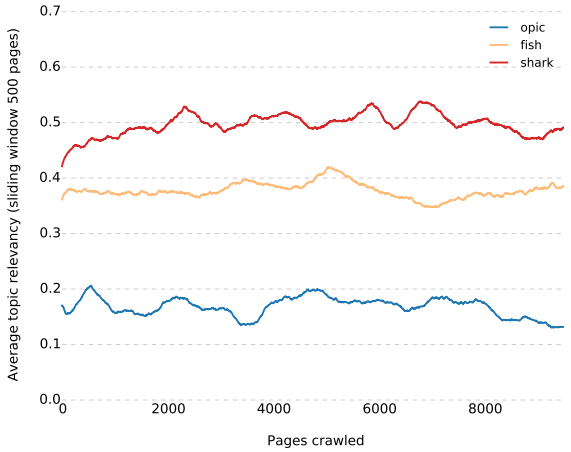

Fig. 6. Internet: Relevancy of fetched pages within a sliding window (window size = 500)

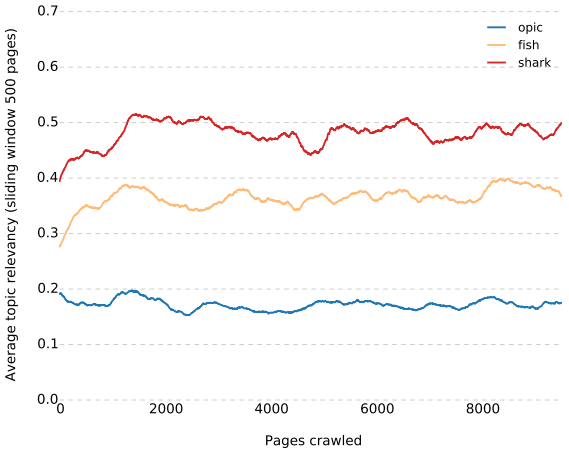

Fig. 7. Offline dataset: Relevancy of fetched pages within a sliding window (window size = 500)

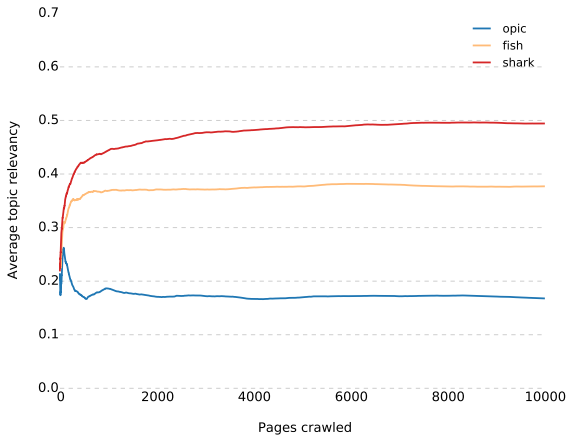

Fig. 8. Internet: Overall topic relevancy during the crawls

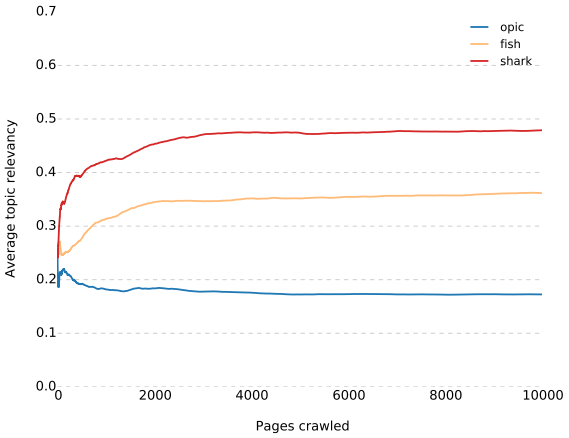

Fig. 9. Offline dataset: Overall topic relevancy during the crawls

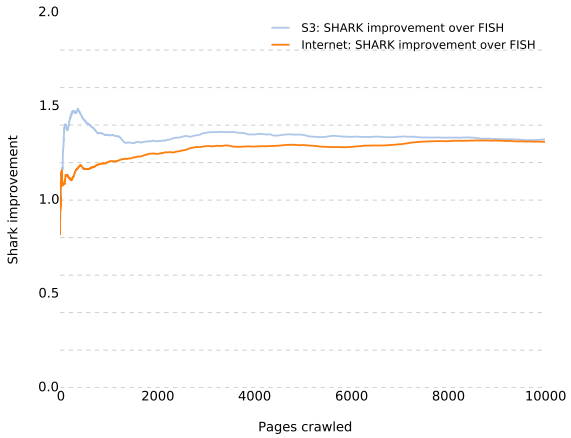

Fig. 10. Performance improvement of Shark search algorithm

URL is missing and we do not want to penalize it for that, we ignore such dead links and we act as they did not exist (i.e. the request is not counted to the download limit).

As in other works on focused crawling, we average the results from all crawling tasks and plot these averaged measures as a time series, so that we capture the temporal behavior of each crawling strategy. For both environments (online and offline) we plotted the measures described in Section 2. In Figures 4 and 5 we plotted the cumulative number of relevant pages, in Figures 6 and 7 the relevancy within a sliding window, and in Figures 8 and 9 the overall topic relevancy.

We can see that the results achieved in the offline environment are consistent with those measured online across all three averaged measures. Both focused crawlers outperform the unfocused OPIC crawler and the Shark algorithm performs better than the Fish crawler, such results are consistent with those reported in literature. We note that the same performance ranking was achieved in almost all of the individual experiments, except *History museums* task, where in offline environment Fish search algorithm outperformed Shark crawler. However, this anomaly did not influenced the average results[8].

We also examined whether the performance improvement of Shark algorithm over Fish algorithm is the same across environments. Figure 10 shows improvement in *Overall topic relevancy* measure plotted in time, we can see that in both environments the improvement converges to the same value (Shark performs 1.35 times better than Fish).

These results look promising and suggest that it should be possible to use large offline graph for comparing focused crawlers. However, it is important to note, that we have tested only best-first-search strategies and that we do not know the impact of smaller subgraph on strategies that can tunnel through irrelevant web pages.

[8] All results are also available at github.com/gogartom/Offline-crawling-environment

5. Future work

As we have mentioned above, we need to test more crawling algorithms in our environment and compare their results with performance measured online. Although there are almost no public implementations of those algorithms, we hope that our open-source environment will motivate other researchers to participate in such comparison (which can be carried out online and offline).

Since the whole Common Crawl dataset can be easily processed by Map-Reduce we can use it to compute relevancy of all 3.5 billions pages and then estimate recall of tested algorithms (influence of seed selection strategy on recall can also be tested). We are also planning to carry out the offline experiments directly in Amazon infrastructure, which should reduce the demands on network traffic and therefore speed up the testing process.

6. Conclusions

In this work we presented an offline environment for focused crawler evaluation. This effort to create an offline testing environment is motivated by difficulties, which developers of focused crawlers face when comparing their works. Dynamic nature of the internet force researchers to simultaneously run all algorithms, which they want to compare. We try to solve this issue by proposing a system for crawling large fixed subgraph of the internet, where the algorithms can be compared.

The environment uses an offline set of 3.5 billions of web pages crawled by Common Crawl foundation. Those pages can be directly accessed using our database, which index positions of particular web documents (represented by URL) within the dataset. We integrated this system into a well-established Nutch crawler and experimented with crawling such fixed subgraph. The first results show that if we compare the performance of basic crawling strategies in our offline environment, the results are consistent with the same comparison, which was carried online. Although these results for best-first-search strategies are promising, we still need to create more consistency experiments with more sophisticated algorithms.

We note that this environment should not substitute online tests completely, but it should be used for preliminary comparisons, incremental testing or for other tasks that are impossible on the internet, such as recall estimation.

All of our work, including Nutch plugins, position index and all results, is publicly available and we hope that it will help to compare and improve future works on focused crawling.

Acknowledgements

This research was financially supported by Department of Cybernetics at Czech Technical University in Prague. Computational resources were provided by the MetaCentrum under the program LM2010005 and the CERIT-SC under the program Centre CERIT Scientific Cloud, part of the Operational Program Research and Development for Innovations, Reg. no. CZ.1.05/3.2.00/08.0144.

References

[1] ABITEBOUL, S., PREDA, M., AND COBENA, G. Adaptive on-line page importance computation. In *Proceedings of the 12th international conference on World Wide Web*, pages 280–290. ACM, 2003.

[2] BRIN, S. AND PAGE, L. Reprint of: The anatomy of a large-scale hypertextual web search engine. *Computer networks*, 56(18):3825–3833, 2012.

[3] BRODER, A., KUMAR, R., MAGHOUL, F., RAGHAVAN, P., RAJAGOPALAN, S., STATA, R., TOMKINS, A., AND WIENER, J. Graph structure in the web. *Computer networks*, 33(1):309–320, 2000.

[4] CHAKRABARTI, S., PUNERA, K., AND SUBRAMANYAM, M. Accelerated focused crawling through online relevance feedback. In *Proceedings of the 11th international conference on World Wide Web*, pages 148–159. ACM, 2002.

[5] CHAKRABARTI, S., VAN DEN BERG, M., AND DOM, B. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31(11):1623–1640, 1999.

[6] DAVISON, B. D. Topical locality in the web. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 272–279. ACM, 2000.

[7] DE BRA, P. AND POST, R. Searching for arbitrary information in the www: The fish-search for mosaic. In *WWW Conference*. 1994.

[8] DILIGENTI, M., COETZEE, F., LAWRENCE, S., GILES, C. L., GORI, M., ET AL. Focused crawling using context graphs. In *VLDB*, pages 527–534. 2000.

[9] HERSOVICI, M., JACOVI, M., MAAREK, Y. S., PELLEG, D., SHTALHAIM, M., AND UR, S. The shark-search algorithm. an application: tailored web site mapping. *Computer Networks and ISDN Systems*, 30(1):317–326, 1998.

[10] LIU, H. AND MILIOS, E. Probabilistic models for focused web crawling. *Computational Intelligence*, 28(3):289–328, 2012.

[11] MEUSEL, R., VIGNA, S., LEHMBERG, O., AND BIZER, C. Graph structure in the web—revisited: a trick of the heavy tail. In *Proceedings of the companion publication of the 23rd international conference on World wide web companion*, pages 427–432. International World Wide Web Conferences Steering Committee, 2014.

[12] SRINIVASAN, P., MENCZER, F., AND PANT, G. A general evaluation framework for topical crawlers. *Information Retrieval*, 8(3):417–447, 2005.

About Authors...

Tomáš GOGÁR was born in Jablonec nad Nisou, Czech Republic. Tomáš received Masters degree in Artificial Intelligence from the Czech Technical University in Prague. He now continues his studies as PhD student focusing on NLP, mainly on Information extraction for specific domains.