Practical Web Crawling for Text Corpora
Work in Progress

Vít Suchomel[1] and Jan Pomikálek[1,2]

[1] Natural Language Processing Centre
Faculty of Informatics, Masaryk University, Brno
{xsuchom2, xpomikal}@fi.muni.cz
nlp.fi.muni.cz

[2] Lexical Computing Ltd.
jan.pomikalek@sketchengine.co.uk

Abstract. SpiderLing—a web spider for linguistics—is new software for creating text corpora from the web, which we present in this article. Many documents on the web only contain material which is not useful for text corpora, such as lists of links, lists of products, and other kind of text not comprised of full sentences. In fact such pages represent the vast majority of the web. Therefore, by doing unrestricted web crawls, we typically download a lot of data which gets filtered out during post-processing. This makes the process of web corpus collection inefficient. The aim of our work is to focus the crawling on the text rich parts of the web and maximize the number of words in the final corpus per downloaded megabyte. We present our preliminary results from creating Web corpora of texts in Czech and Tajik.

Key words: Crawler, web crawling, corpus, web corpus, text corpus

1 Introduction

Text corpora have a wide range of applications in linguistics. The source of data for corpora which has become very popular in the recent years is the web. A web crawler is a computer program traversing web pages and downloading documents. Due to the enormous size of the web an important measure of the quality of a crawler is its speed – the number of bytes downloaded per time unit. However, often it also matters which bytes we are downloading. In the context of web corpora, an even more important measure of quality is the number of words in the final corpus retrieved per time unit. Therefore, in this context the quality of a crawler depends on the implemented traversing algorithm.

We have experimented with using third party software for obtaining text documents from the web. Following the example of other researchers [1], we have used Heritrix crawler[1] and downloaded documents for the language in

[1] http://crawler.archive.org/

Aleš Horák, Pavel Rychlý (Eds.): Proceedings of Recent Advances in Slavonic Natural Language Processing, RASLAN 2011, pp. 97–108, 2011.

interest by restricting the crawl to national web domains of the countries where the language is widely used (e.g. `.cz` for Czech). Though we managed to compile corpora of up to 3 billion words in this way, we were not satisfied with the fact that we need to keep the crawler running for several weeks and download terabytes of data in order to retrieve a reasonable amount of text. It turned out that most downloaded documents are discarded during post-processing since they contain only material with little or no running text.

In order to reduce the amount of unwanted downloaded content, we decided to create a custom web crawler which actively looks for text rich resources and avoids websites containing only material not suitable for text corpora. Our hope was that by avoiding the unwanted content we can not only save bandwidth but also shorten the time required for building a web corpus of a given size.

2 Analysis of previous work

We were interested to know how much data we download in vain when using Heritrix and if the sources which should be avoided can be easily identified. In order to get that information we analyzed the data of a billion word corpus of European Portuguese downloaded from the `.pt` domain with Heritrix. For each downloaded web page we computed its yield rate as

$$yield\ rate = \frac{final\ data}{downloaded\ data}$$

where *final data* is the number of bytes in the text which the page contributed to the final corpus and *downloaded data* is simply the size of the page in bytes (i.e. the number of bytes which had to be downloaded). Many web pages have a zero yield rate, mostly because they get rejected by a language classifier or they only contain junk or they only contain text duplicate to previously retrieved text.

We grouped the data by web domains and computed a yield rate for each domain as the average yield rate of the contained web pages. We visualized this on a scatterplot which is displayed in Fig. 1. Each domain is represented by a single point in the graph.

It can be seen that the differences among domains are enormous. For example, each of the points in the lower right corner of the graph represents a domain from which we downloaded more than 1 GB of data, but it only yielded around 1 kB of text. At the same time, there are domains which yielded more than 100 MB of text (an amount higher by 5 orders of magnitude) from a similar amount of downloaded data. These domains are positioned in the upper right corner of the graph.

Next, we selected a set of yield rate thresholds and computed for each threshold the number of domains with a higher yield rate and the sum of downloaded and final data in these domains. The results can be found in Table 1.

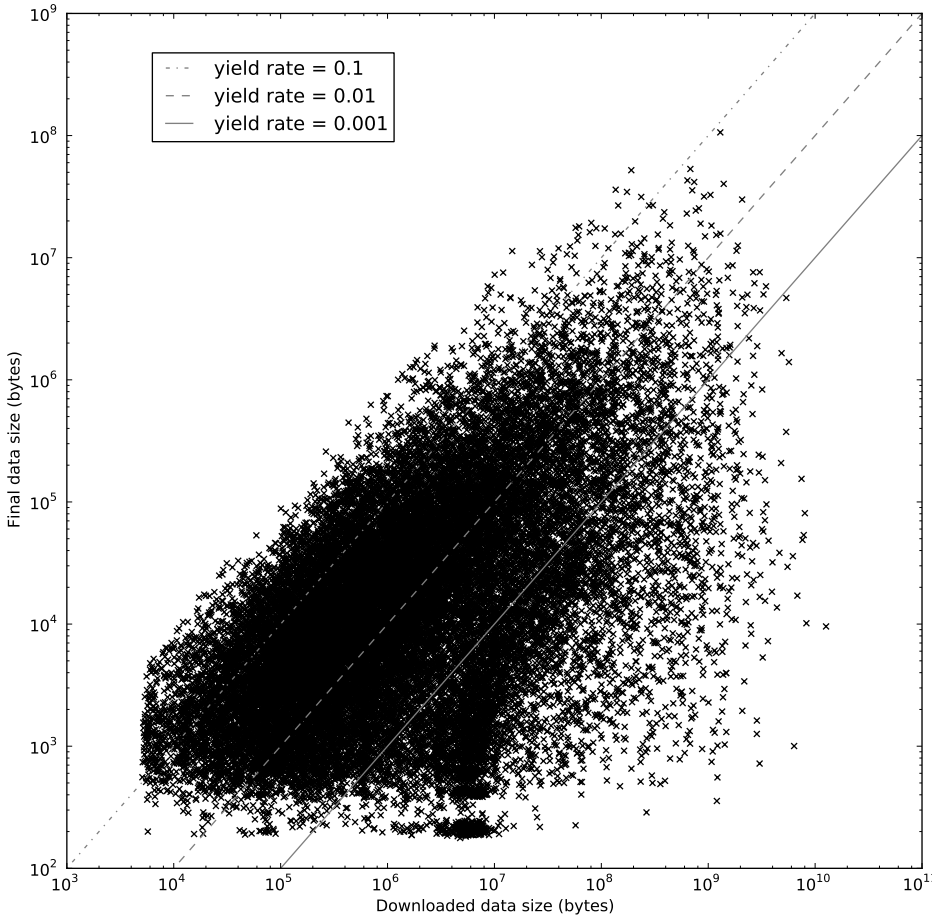

Fig. 1. Web domains yield rate for a Heritrix crawl on `.pt`.

Table 1. Sums of downloaded and final data size for all domains above given the yield rate threshold.

yield rate threshold	domains	total downl data	total final data
none	51645	1288.87 GB	4.91 GB
0	31024	1181.56 GB	4.91 GB
0.0001	29580	705.07 GB	4.90 GB
0.0002	28710	619.44 GB	4.89 GB
0.0004	27460	513.86 GB	4.86 GB
0.0008	25956	407.30 GB	4.80 GB
0.0016	24380	307.27 GB	4.68 GB
0.0032	22325	214.18 GB	4.47 GB
0.0064	19463	142.38 GB	4.13 GB
0.0128	15624	85.69 GB	3.62 GB
0.0256	11277	45.05 GB	2.91 GB
0.0512	7003	18.61 GB	1.98 GB
0.1024	3577	5.45 GB	1.06 GB
0.2048	1346	1.76 GB	0.54 GB
0.4096	313	0.21 GB	0.1 GB

It is easy to see that as the yield rate threshold increases the size of the downloaded data drops quickly whereas there is only a fairly small loss in the final data. This suggests that by avoiding the domains with low yield rate a web crawler could save a lot of bandwidth (and time) without making the final corpus significantly smaller. For instance if only domains with a yield rate above 0.0128 were crawled, the amount of downloaded data would be reduced from 1289 GB to 87 GB (to less than 7%) while the size of the final data would only drop from 4.81 GB to 3.62 GB (73.7%). This is of course only a hypothetical situation, since in practice one would need to download at least several pages from each domain in order to estimate its yield rate. Nevertheless, it is clear that there is a lot of room for making the crawling for web corpora much more efficient.

One could argue that a segmentation by domains is too coarse-grained since a single domain may contain multiple websites with both high and low yield rates. While we fully agree, we believe that identifying more fine-grained sets of web pages, such as websites, introduces further complications and we leave that for future work.

3 SpiderLing

We came to the conclusion that the easiest way of implementing our very specific requirements on web crawling is to create a custom crawler from scratch. We selected Python as our programming language to support rapid development. In order to make debugging easier we avoided a multi-threaded design. Instead, we use asynchronous communication for downloading data

from multiple servers at the same time – a simple solution which scales up well (we can keep up to 5000 simultaneously open connections without any problems).

3.1 Improving yield rate

Our primary aim is to identify high-yielding domains and to avoid low-yielding ones. At the same time we want to make sure that we do not download all the data only from a few top-yielding domains so that we achieve a reasonable diversity of the obtained texts.

We collect information about the current yield rate of each domain as we are crawling the web. If the yield rate drops below a certain threshold we blacklist the domain and do not download any further data from it. We define a minimum amount of data which must be retrieved from each domain before it can be blacklisted. Currently the used limit is 8 web pages or 512 kB, whichever is a higher amount of data. The yield rate threshold is dynamic and increases as more pages are downloaded from the domain. This ensures that sooner or later all domains get blacklisted, which prevents overrepresentation of data from a single domain. Nevertheless, low-yielding domains are blacklisted sooner and thus the average yield rate increases.

The yield rate threshold for a domain is computed using the following function:

$$t(n) = 0.01 \cdot \left(\log_{10}(n) - 1\right)$$

where n is the number of documents downloaded from the domain. The function is based partly on the authors' intuition and partly on the results of initial experiments. Table 2 contains a list of thresholds for various numbers of downloaded documents.

Table 2. The yield rate threshold as a function of the number of downloaded documents.

# of documents	yr threshold
10	0.00
100	0.01
1000	0.02
10000	0.03

We experimented with various parameters of the yield rate threshold function. Fig. 2 shows how the average yield rate changes in time with different yield rate threshold functions. All these experiments have been performed with Czech as the target language. It can be seen that stricter threshold functions result in higher average yield rate. However, too high thresholds have a negative impact on the crawling speed (see section 3.5). It is therefore necessary to make a reasonable compromise.

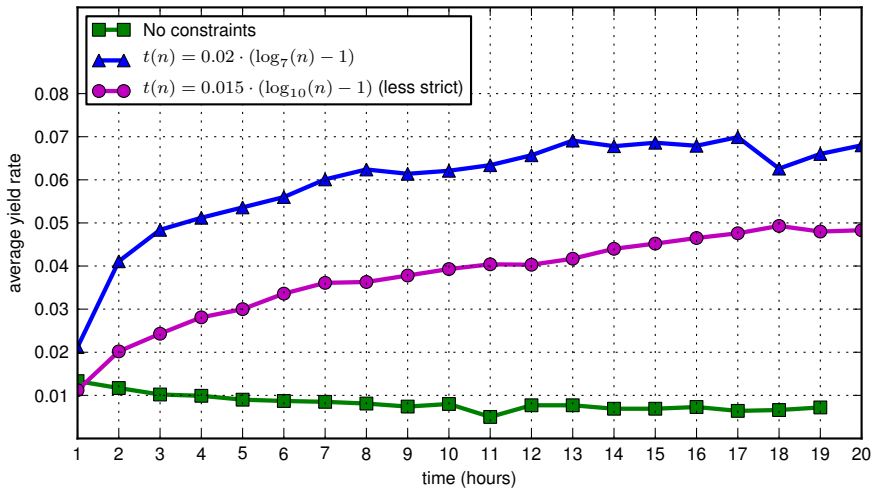

Fig. 2. Average yield rate in time for various yield rate threshold functions.

3.2 Removing junk and duplicates

We use jusText[2] [3]—a heuristic based boilerplate removal tool—to remove content such as navigation links, advertisements, headers and footers from downloaded web pages. Only paragraphs containing full sentences are preserved.

Duplicate documents are removed at two levels: (i) original form (text + HTML), and (ii) clean text as produced by jusText. Two correspondent checksums are computed for each web page and stored in memory. Documents with previously seen checksums are discarded. As a post-processing step, we also remove near-duplicates using onion[3].

We currently do not filter unwanted web content such as link farms and machine generated texts. This may be a subject to further research. Note though that some of such content (e.g. excerpts of Wikipedia articles on link farms) is already reduced in our current processing pipeline as a positive side effect of de-duplication.

3.3 Character encoding and language detection

We need to detect the character encoding of each downloaded document in order to be able to display its text correctly and/or to unify the encoding of all documents, e.g. by converting to UTF-8. Though for most web pages the character encoding may be determined from the meta tags or from HTTP headers, the information is not always available and not always correct and

[2] `http://code.google.com/p/justext/`
[3] `http://code.google.com/p/onion/`

in general cannot be relied upon. Therefore, we detect the encoding from the contained text by using chared[4].

Language detection is another problem which has to be addressed since we are typically building a web corpus for a particular language and want to avoid any texts in other languages. Unfortunately, it is difficult to detect a language of a text without identifying its character encoding first and vice versa. This is a typical chicken and egg problem. We use a simple trick here. We perform the character encoding detection first, assuming the input is in our target language. If the assumption is not correct it is very likely that the input gets rejected by the language filter in the next step anyway and thus it does not matter if the encoding detections fails.

Language filtering is performed at two levels. (i) jusText—our boilerplate removal tool—uses a list of the most frequent words in the language for identifying paragraphs containing grammatical text. The rationale is that the most frequent words are typically function words and a grammatical text should contain a certain proportion of these words. This filtering has a positive side effect of rejecting texts in other languages. (ii) We build a histogram of triples of characters on a sample text in the target language and compare the histogram of the text of each downloaded document with the model. This is done using the Trigram class created by Douglas Bagnall[5]. We use a similarity threshold of 0.5.

Both applied language filtering methods tend to accept texts in similar languages (e.g. Slovak texts when the target language is Czech). Nevertheless, this has not been a major problem so far. When creating a Czech corpus, only a small amount of Slovak has been included and we managed to identify and remove these texts during post-processing.

3.4 Starting URLs

A large set of starting URLs is needed for the crawler to quickly start retrieving the data from many domains in parallel. We use Corpus Factory [2] for getting a list of starting URLs for the target language. The tool compiles a list of medium frequency words in the language by using texts from Wikipedia. These words are then randomly combined into tuples of 3 to 5 and each tuple is used as a query to a search engine (we currently use bing[6]). As a result we get a list of URLs which are likely to contain documents in our target language.

3.5 Crawling speed

The maximum crawling speed we have achieved in our experiment was ca. 12 MB/s. However, we observe that the speed tends to decrease as the crawling progresses. Since we typically start from a large set of seed URLs, we have

[4] `http://code.google.com/p/chared/`

[5] `http://code.activestate.com/recipes/326576-language-detection-using-character-trigrams/`

[6] `http://www.bing.com/`

enough distinct domains to download from in parallel at the beginning and thus the initial crawling speed is good. However, as the crawling continues currently processed domains get blacklisted faster than new high-yielding domains are discovered. This reduces the number of domains available for download and thus limits the crawling speed.

Fig. 3 and Fig. 4 show how the crawling speed changes in time. The speed is measured as the amount of raw HTML and the amount of clean text retrieved per time unit. The data originates from two web crawls – for Czech and Tajik. For Tajik, the available online resources are very scarce which affects the crawling speed significantly.

3.6 Crawling constraints

A web crawler should abide by the Robots Exclusion Standard[7]. SpiderLing uses a third party Robot Exclusion Rules Parser[8] which implements the up-to-date (2008) version of the standard better than the Python built-in library. The used parser also supports several non-standard but frequently used robots.txt directives.

A crawler should not overuse web servers by querying too often. Our crawler implements both per web domain and per IP address limits and by default makes a maximum of 12 queries per minute and 10 queries per second respectively.

3.7 Checkpoints

SpiderLing supports periodical saving of all important in-memory data (visited domains, queued URLs, document checksums) to file system. It is possible to resume crawling from a saved state in case of a failure (e.g. due to a bug in the code or a server dropout). The state is saved in a human readable form and allows manual inspection for debugging purposes.

4 Results

4.1 Yield rate

By applying yield rate thresholds on domains we managed to reduce downloading data which is of no use for text corpora and increased the overall average yield rate. Fig. 5 contains the same kind of scatterplot as displayed in Fig. 1, this time on the data downloaded by SpiderLing with Czech as a target language. This is a significant improvement over the previous graph. For low-yielding domains only up to 1 MB of data is downloaded and high amounts of data are only retrieved from high-yielding sources. Table 3 contains a summary of the results of this web crawler run.

[7] `http://www.robotstxt.org/`

[8] `http://nikitathespider.com/python/rerp/`

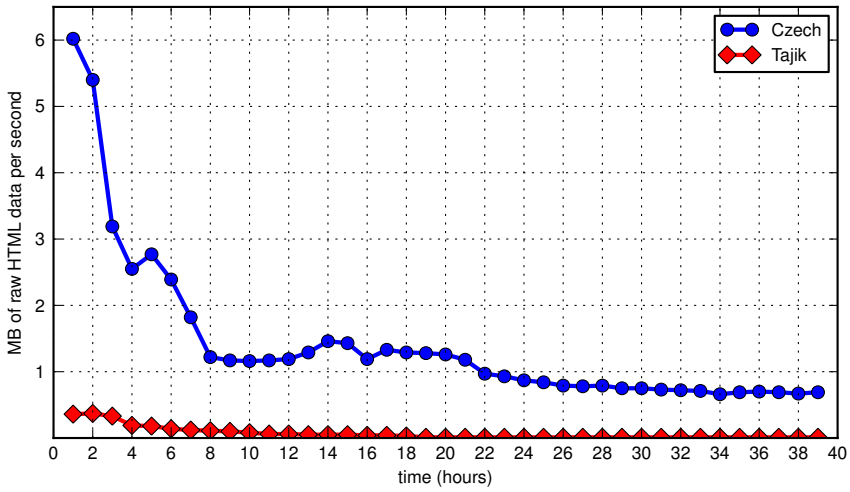

Fig. 3. Download speed in time in terms of downloaded raw HTML data.

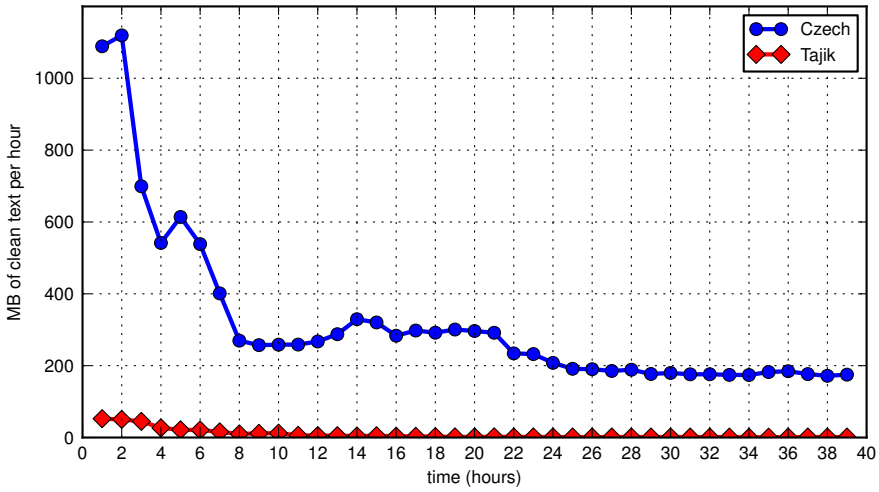

Fig. 4. Download speed in time in terms of downloaded clean texts.

Table 3. Results of crawling Czech web with SpiderLing.

downloaded documents	15,525,554
downloaded data size	515,580 MB
final data size	30,522 MB
yield rate	5.92 %

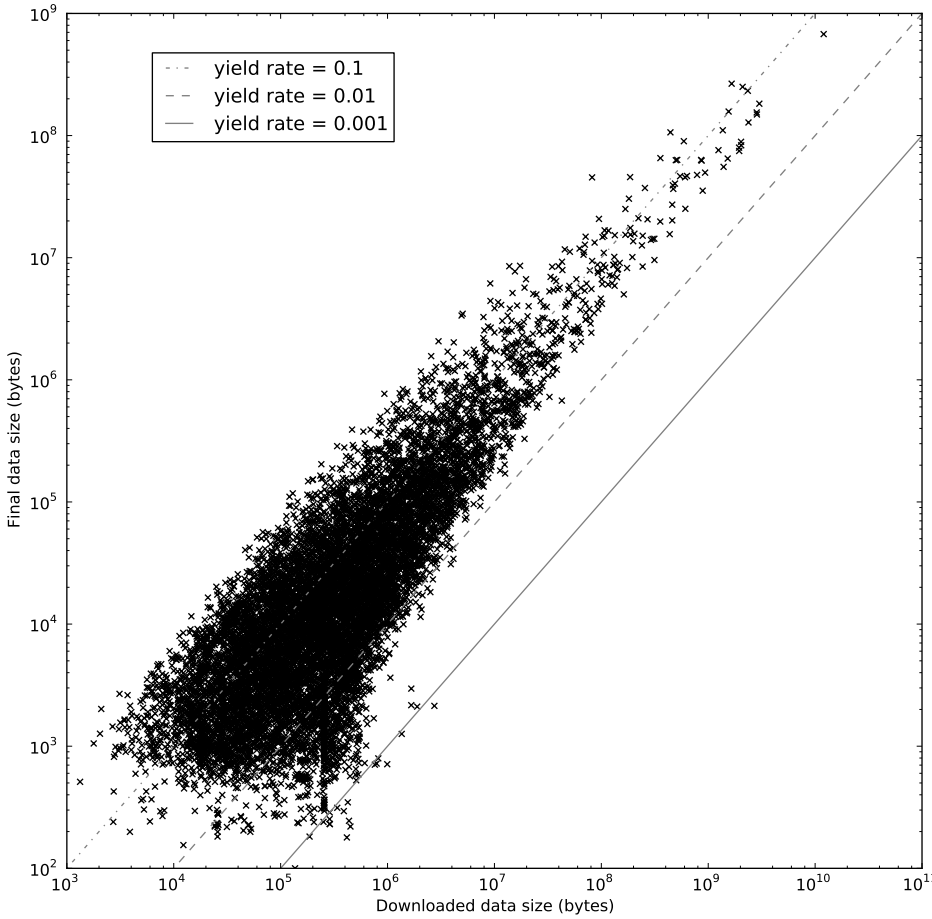

Fig. 5. Web domains yield rate for a SpiderLing crawl on Czech web.

4.2 Created corpora

So far we have used SpiderLing to create two corpora. During the development of the crawler we downloaded a total of ca. 4 TB Czech web pages in a number of web crawler runs. This amounts to ca. 5 billion tokens after all post-processing steps, including de-duplication with onion. We merged the corpus with a ca. 2 billion word Czech web corpus we have collected previously by using Heritrix. Since the two corpora overlapped to a high extent, the size of the final Czech web corpus after de-duplication is 5.8 billion tokens.

As a next exercise we ran SpiderLing on Tajik, partly to support the work of a visiting fellow researcher and partly to find out how the crawler will deal with scarce online resources. We started the crawl from 2570 seed URLs (from 475 distinct domains) collected with Corpus Factory. Over 3 days the crawler downloaded 9.5 GB of HTML data which yielded a 35 million tokens corpus after all post-processing.

5 Future work

The primary goal of our future work is to test the crawler on other languages and create further large web corpora. We believe that crawling speed might be less of a problem for languages where vast online text resources are available (e.g. English or Spanish). Nevertheless, we also want to invest more effort into optimizing the crawling constraints so that a higher crawling speed can be achieved even for scarcer resources.

Other plans for the future include analyzing the topics and genres of the downloaded texts and eventually balancing the downloaded content in this respect.

6 Conclusion

We presented SpiderLing, a web crawler for text corpora. We have shown that the crawler can effectively avoid web data not suitable for text corpora and significantly improve the yield rate of the downloaded content. The crawler has already been successfully applied for creating a major part of a large (5.8 billion tokens) Czech web corpus. We also managed to create a 35 million tokens web corpus of Tajik by using SpiderLing. Though this is only a smallish corpus, we consider it a promising achievement since online Tajik texts are scarce.

Acknowledgements The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. 248307 (PRESEMT project), from the Ministry of Education of CR within the Center of basic research LC536 and from the Czech Science Foundation under the project P401/10/0792.

References

1. M. Baroni, S. Bernardini, A. Ferraresi, and E. Zanchetta. The wacky wide web: A collection of very large linguistically processed web-crawled corpora. *Language Resources and Evaluation*, 43(3):209–226, 2009.
2. A. Kilgarriff, S. Reddy, J. Pomikálek, and A. PVS. A corpus factory for many languages. *Proc. LREC, Malta*, 2010.
3. J. Pomikálek. *Removing Boilerplate and Duplicate Content from Web Corpora*. PhD thesis, Masaryk University, Brno, 2011.