# Building a 70 billion word corpus of English from ClueWeb

**Jan Pomikálek, Miloš Jakubíček, Pavel Rychlý**

NLP Centre, Faculty of Informatics, Masaryk University, Brno, Czech Republic
Lexical Computing Ltd., Brighton, United Kingdom

**Abstract**

This work describes the process of creation of a 70 billion word text corpus of English. We used an existing language resource, namely the ClueWeb09 dataset, as source for the corpus data. Processing such a vast amount of data presented several challenges, mainly associated with pre-processing (boilerplate cleaning, text de-duplication) and post-processing (indexing for efficient corpus querying using the CQL – Corpus Query Language) steps. In this paper we explain how we tackled them: we describe the tools used for boilerplate cleaning (jusText) and for de-duplication (onion) that was performed not only on full (document-level) duplicates but also on the level of near-duplicate texts. Moreover we show the impact of each of the performed pre-processing steps on the final corpus size. Furthermore we show how effective parallelization of the corpus indexation procedure was employed within the Manatee corpus management system and during computation of word sketches (one-page, automatic, corpus-derived summaries of a word's grammatical and collocational behaviour) from the resulting corpus.

**Keywords:** corpus, clueweb, English, encoding, word sketch

## 1. Introduction

Text corpora–large collections of electronic texts–are one of the essential resources for linguistics and computational linguistics in particular. They have applications in a wide range of fields, such as machine translation, speech recognition, lexicography, language learning and teaching, etc.

A distribution of words in a natural language follows the Zipf law (Zipf, 1949). Its simple interpretation is that there are only few words which are used frequently and many words which are used rarely. The same applies to other language phenomena, such as collocations, phrases, patterns, etc. In order to get evidence about the rare phenomena a lot of text is required. In general, the more text is available, the more distinct language phenomena can be found in it. Thus, everything else being the same, a larger corpus is always better than a smaller corpus.

In the recent years, Web corpora (text corpora created from the Web) became very popular due to the availability of a vast amount of electronic texts on the Web. With relatively low costs, Web corpora can be built in sizes which would be virtually impossible to achieve using traditional corpus creation methods. Nevertheless, building useful Web corpora is not straightforward and requires a lot of expertise.

## 2. Building the corpus

A common Web corpus building procedure is as follows. First, a set of Web pages has to be retrieved. This is typically done using a web crawler. Alternative approaches include BootCaT (Baroni and Bernardini, 2004) or Corpus Factory (Kilgarriff et al., 2010). Next, a language filter should be applied in order to weed out documents not in the language of interest. Web pages need to be converted to plain text along with removing boilerplate parts, such as navigation links, advertisements, copyright notices, etc. Duplicate and near-duplicate texts have to be detected and preserved only in one instance. It is also useful if the corpus undergoes a linguistic annotation, such as part-of-speech tagging. If the corpus data is to be used directly by humans (researchers, lexicographers, language learners), as the last step, the corpus has to be indexed using a specialised software called corpus manager.

Despite being principally straightforward, web crawling is a technically difficult task, especially when large scale crawls are considered. Though the number of freely available web crawling software is substantial, the truly robust solutions suitable for multi-terabyte crawls are rare. Also, when crawling for corpus data, it is typically required that pages not in the language of interest are avoided. This may be difficult to achieve using a third-party software. Last but not least, even if all other problems can be resolved, doing large scale crawls in an academic environment takes a lot of time since the available technology (both software and hardware) is limited.

ClueWeb09[1] is a collection of ca 1 billion Web pages in 10 languages collected by Carnegie Mellon University in January and February 2009. The dataset is freely available for research purposes. This collection is a valuable resource, since (i) unlike constantly changing Web it is a fixed dataset which makes a reproducible research possible and (ii) it saves researchers from struggling at doing their own web crawls.

For building our corpus, we used the English part of ClueWeb which comprises roughly one half of the whole collection, i.e. ca 500 million Web pages. The first stage of the processing included removing boilerplate and language filtering. For removing boilerplate, we used jusText (Pomikálek, 2011)[2] – our heuristic based boilerplate removal tool. Since we wanted to be sure that all texts included in the corpus are indeed in English, we applied a language filter as the next step. We employed the trigram.py module,[3] which classifies input text based on the frequencies of triples of characters. It compares this profile with a profile of a model text and outputs a similarity score between 0 and 1. We used a filtering thresholds of 0.4 which we found to give good

[1] http://lemurproject.org/clueweb09.php

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

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

| Value | Size |
|---|---|
| word forms | 68,845,137,110 |
| numbers | 1,485,033,080 |
| alphanumeric | 70,330,170,190 |
| punctuation | 9,849,840,275 |
| others | 1,810,813,890 |
| total tokens | 81,990,824,355 |
| documents | 138,988,120 |

Table 1: Overall corpus characteristics

results in our previous experiments.
In the next step, we removed duplicate and near-duplicate data from the corpus. This is described in the next section. As the last step, we performed part-of-speech tagging using TreeTagger (Schmid, 1994).

## 3. Deduplication

Removing duplicate and near-duplicate texts was one of the two challenging parts in the processing pipeline. The enormous size of the processed dataset obviously prohibited the naive approach – examining each pair of documents for duplicate content. Rather than that, we used onion[4], a tool we developed specifically for de-duping corpus data. Most other existing de-duplication methods, such as Broder's shingling algorithm (Broder, 2000) or Charikar's algorithm (Charikar, 2002), can only identify highly similar pairs of documents and fail to detect near-duplicates at intermediate level (Pomikálek, 2011). This is a problem for text corpora where any artificial duplicates (those which are not a result of a natural or coincidental independent use of the same sequence of words) cause problems. Onion, on the other hand, can identify and remove near-duplicates at any level of similarity.
Onion is an n-gram based one pass deduplication algorithm. It processed the input documents (or paragraphs, depending on user's preferences) one by one. For each document, all n-grams of words are extracted (10-grams by default) and compared with the set of previously seen n-grams (union of the n-grams of previously seen documents). This identifies the parts of the document which already exist in the currently processed part of the corpus. If the proportion of text within these duplicated parts is above a predefined threshold, the document is discarded. Otherwise, the document is preserved and its n-grams are added to the set of previously seen n-grams.
A major drawback of this simplified version of the algorithm is that it has to hold in memory the set of n-grams of all processed documents, except for the discarded ones. Obviously, the size of the set grows as the algorithm progresses and by the end of processing the number of contained n-grams is usually close to the total number of words in the corpus. This is because in a typical corpus, most n-grams (more than 90 %) have a single (unique) occurrence. For instance, for processing a corpus of 10 billion words, we may expect that more than 9 billion of n-grams will have to be held in memory. Assuming 8 bytes per n-gram[5] and no memory overhead of the used data structure, more than 67 GB RAM would be required.
In order to reduce memory requirements, onion can precompute the set of all duplicate n-grams (n-grams with two or more occurrences) in the whole corpus. Using this set, any n-gram unique in the corpus can be identified (it is not in the set). In the stage of the deduplication algorithm where new n-grams are added to the set of previously seen n-grams, all unique n-grams can be pruned (since it is guaranteed that they will not be seen again, there is no point in storing them). This reduces the size of the set and thus the memory requirements significantly, since, as we already mentioned, the unique n-grams typically constitute the vast majority (more than 90 %) of all n-grams in the corpus.
When precomputing the set of duplicate n-grams, or more precisely the set of hashes of duplicate n-grams, the algorithm divides the hash space into non-overlapping intervals (or buckets, 10 by default). Each interval is associated with a file. In a single pass over all n-grams in the corpus, a hash of each n-gram is computed and stored in the file correspondent to the value of the hash. Duplicate hashes are then found in each file individually by sorting it in memory. The number of intervals (buckets) must be such that sufficiently small files are created and in-memory sorting is possible.
In Table 2 we summarize the influence of each step on the resulting corpus size during preparation of a subpart of ClueWeb that consisted of about 7 % of the whole data set (for efficiency reasons intermediate results were not stored when the whole corpus was processed).

## 4. Parallelization of processing

All processing has been done on a single powerful server with eight 8-core Intel Xeon X7560 2.27 GHz CPUs and 440 GB RAM. As a disk storage we used a 70 TB RAID-6 disk array with 2 TB 7200 rpm SAS 6 Gbps hard drives. Despite using a single server, the availability of 64 CPU cores allowed for a massive parallelization.
For the initial stage (removing boilerplate, language filtering) and the last stage (POS-tagging), the parallelization was completely straightforward. We simply split the corpus into X equally sized parts and processed them with X concurrently running processes of the same type. The X was 50 for removing boilerplate and language filtering, and 20 for POS-tagging.
Parallelizing the de-duplication stage was more difficult and only possible to a certain extent. We first reduced the amount of duplicated data by identifying identical documents using their MD5 hashes and keeping only one instance of each. This has been done by splitting the corpus into 10 parts and processing them independently and in parallel. Apparently, this procedure did not account for identical documents in different parts, but it still reduced the amount of duplicated data significantly. The purpose of this step was mainly to reduce the CPU and RAM resources required for the next step.
We then ran 10 instances of onion's n-gram hash generation program (hashgen) on the 10 corpus parts using 100 buckets.

[4] http://code.google.com/p/onion

[5] The algorithm stores 64-bit hashes rather than raw n-grams.

| stage | word count | % | token count |
|---|---|---|---|
| original subcorpus | 24,633,016,767 | n/a | n/a |
| after removing boilerplate | 11,363,624,711 | 46.1 % | n/a |
| after language filtering | 9,586,878,336 | 84.4 % | n/a |
| after removing exact duplicates | 8,983,831,725 | 93.7 % | 10,359,277,678 |
| after block level de-duplication | 7,279,959,176 | 81.0 % | 8,390,930,801 |

Table 2: Overview of subcorpus (7 %) size throughout processing. The percentage values are relative to the previous row. Word count refers to the number of space separated tokens.

| Step | CPU cores | Time | RAM |
|---|---|---|---|
| removing boilerplate + language filtering | 50 | 108 hours | < 1GB |
| removing identical documents | 10 | 9 hours | 5 GB (10 x 0.5 GB) |
| generating hashes of n-grams | 10 | 1 hour | < 1GB |
| finding hashes of duplicate n-grams | 1 | 9 hours | 10 GB |
| de-duplication | 1 | 65 hours | 148 GB |
| POS-tagging | 20 | 44 hours | < 1GB |

Table 3: Processing times, RAM requirements

This yielded 1,000 files. We then concatenated the files corresponding to the buckets of the same type ending up with 100 files containing distinct lists of hashes. Though the duplicate hashes could be found in each of these files independently, it was not possible to do this in parallel due to memory constraints. Thus, the complete set of all duplicate hashes has been obtained by sorting these 100 files one by one. Also, the de-duplication itself only used a single CPU. The overview of the parallelization process in terms of used CPU time and memory is provided in Table 3.

## 5. Corpus Encoding

For manual linguistic introspection, text corpora are usually subject to specific encoding (indexation) by specialized tools (corpus managers, corpus database management systems). Appropriate indexation makes it possible to perform fast and complex search even for billion word corpora (Jakubíček et al., 2010). For this purpose we used the Manatee/Bonito corpus management system (Rychlý, 2000; Rychlý, 2007).

For obvious reasons the indexation procedure was of less interest what regards its time and space complexity – what mattered was the efficiency of resulting queries to the corpus while the time required to perform the whole encoding was bearable even if it lasted several dozens of hours.

This was all true up to and including billion word corpora. However, advancing by another order of magnitude revealed that this step of corpus finalization needs to be revisited: sequential algorithms for corpus indexation would even with high-tech equipment require not hours or days, but rather weeks and months to finish. For the first time in the 20 years history of text corpora encoding, the amount of processed data has outrun the computer performance that is commonly available.

### 5.1. Parallelization of Corpus Encoding

To speed up the corpus encoding, the whole process has been parallelized so as to make it possible to deploy the encoding procedure in a single-host multi-process or even multi-host cluster or cloud environment.

Unfortunately, the encoding process cannot be parallelized in an entirely trivial way, i.e. simply by separating the corpus into several datasets that could be processed independently and their results would be just merged in the end – and we believe that this holds not only for Manatee, but for any corpus management system, since the very basic thing such a system needs to perform is a string (type) to id (index number) mapping of the underlying lexicon and these values are likely to be included in multiple subsequent indexing –and of course must be consistent across the whole corpus which would not be guaranteed by trivial parallelization.

One obvious solution to this issue would consist of reindexing the whole lexicon and dependent indices at the end of the encoding process but considering the target size of the corpus (let's emphasize it again: we are talking about corpora of 10 to 100 billion words) this might slow down the parallelization to the extent that it might not even outperform its simple sequential predecessor.

In Figure 1 we outline the workflow of a much more efficient approach that has been implemented in the Manatee corpus management system and exploits the Zipf probability distribution of the lexicon that can be for this purpose reformulated as follows: when processing the corpus, the size of lexicon increment is inversely proportional to the number of tokens processed. In other words, the lexicon grows vastly only at the beginning of the processing and tends to keep almost the same size at its end.

Therefore we can process first 10 (20, 50, ...) million words sequentially at the very beginning of the encoding process and hereby prepare a solid lexicon basis that will require

| sequential encoding | parallel encoding, unique attributes handling | speedup |
|---|---|---|
| 76 days | 7 days | > 10 |
| **sequential sketch computation** | **parallel sketch computation** | **speedup** |
| 64 days | 8 days | > 8 |

Table 4: Comparison of sequential and parallel encoding of the 70 billion EnClueWeb corpus

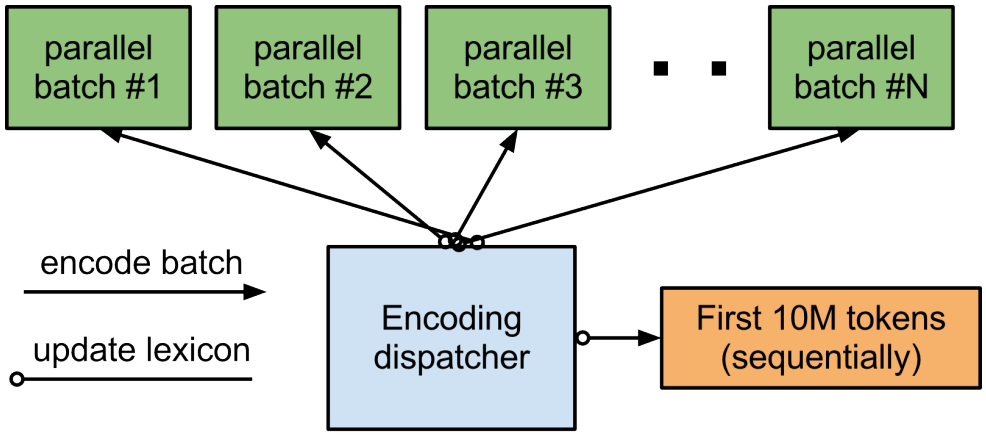


Figure 1: Overview of the parallel encoding workflow

only infrequent updates in further parallel processing that will gain a significant speedup, as given in Table 4.
Besides the parallelization, special handling of unique structure attributes (like document's URL) has been implemented in order to speed up the encoding since such attributes do not require building most of the indexation structures (like inverted index), need just a basic lexicon (string to attribute id mapping).

### 5.2. Parallel Computation of Word Sketches

Besides basic encoding of the corpus for the Manatee system, the word sketch relations (Kilgarriff et al., 2004) have been computed on the resulting corpus. The computation of word sketches consists in evaluation of a series of complex CQL queries (see e.g. (Jakubíček et al., 2010) for an overview of CQL – Corpus Query Language). Fortunately, at least for some word sketch relations this procedure can be easily parallelized by computing separate relations (or even queries) in parallel and merging their results. By applying this approach we managed to speed up the evaluation of word sketches on the corpus by factor of 8 down to about 8 days.

## 6. Conclusion

In this paper we presented a newly created 70 billion word corpus of English that originates from the ClueWeb data set. We described the process of converting the collection of Web pages into a linguistically useful text corpus. Our main focus was on de-duplication, which presented the major challenge. Finally, a parallelization procedure for corpus encoding within the Manatee corpus management system has been presented that speeds up the whole indexation process by more than factor of 5.

## 7. Acknowledgements

This work has been partly supported by the Ministry of Education of CR within the LINDAT-Clarin project LM2010013, by EC FP7 project ICT-248307 and by the Czech Science Foundation under the project P401/10/0792.

## 8. References

Marco Baroni and Silvia Bernardini. 2004. BootCaT: Bootstrapping Corpora and Terms from the Web. In *Proceedings of LREC*, volume 4.

Andrei Broder. 2000. Identifying and Filtering Near-Duplicate Documents. In *Combinatorial Pattern Matching*, pages 1–10. Springer.

Moses S. Charikar. 2002. Similarity Estimation Techniques from Rounding Algorithms. In *Proceedings of the Thirty-Fourth Annual ACM Symposium on Theory of Computing*, pages 380–388. ACM.

Miloš Jakubíček, Pavel Rychlý, Adam Kilgarriff, and Diana McCarthy. 2010. Fast Syntactic Searching in Very Large Corpora for Many Languages. In *PACLIC 24 Proceedings of the 24th Pacific Asia Conference on Language, Information and Computation*, pages 741–747, Tokyo.

A. Kilgarriff, P. Rychlý, P. Smrž, and D. Tugwell. 2004. The Sketch Engine. In *Proceedings of the Eleventh EURALEX International Congress*, pages 105–116, Lorient, France. Universite de Bretagne-Sud.

Adam Kilgarriff, Siva Reddy, Jan Pomikálek, and Avinesh PVS. 2010. A corpus factory for many languages. In *LREC Workshop on Web Services and Processing Pipelines*. Valetta, Malta: ELRA.

Jan Pomikálek. 2011. *Removing Boilerplate and Dupli-*

*cate Content from Web Corpora*. Phd thesis, Masaryk University.

Pavel Rychlý. 2000. *Korpusové manažery a jejich efektivní implementace*. PhD Thesis, Masaryk University, Brno.

Pavel Rychlý. 2007. Manatee/Bonito - A Modular Corpus Manager. In *1st Workshop on Recent Advances in Slavonic Natural Language Processing*, pages 65–70, Brno.

Helmut Schmid. 1994. Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, volume 12, pages 44–49. Manchester, UK.

George Kingsley Zipf. 1949. *Human Behavior and the Principle of Least Effort*. Addison-Wesley Press.