FOUNDATIONS OF COMPUTING AND DECISION SCIENCES
Vol. 37 (2012) No. 4

DOI: 10.2478/v10209-011-0015-8

GOOGLE BOOKS NGRAMS RECOMPRESSED AND SEARCHABLE

Szymon GRABOWSKI*, Jakub SWACHA**

Abstract. One of the research fields significantly affected by the emergence of "big data" is computational linguistics. A prominent example of a large dataset targeting this domain is the collection of Google Books Ngrams, made freely available, for several languages, in July 2009. There are two problems with Google Books Ngrams; the textual format (compressed with Deflate) in which they are distributed is highly inefficient; we are not aware of any tool facilitating search over those data, apart from the Google viewer, which, as a Web tool, has seriously limited use. In this paper we present a simple preprocessing scheme for Google Books Ngrams, enabling also search for an arbitrary n-gram (i.e., its associated statistics) in average time below 0.2 ms. The obtained compression ratio, with Deflate (zip) left as the backend coder, is over 3 times higher than in the original distribution.

Keywords: data compression, random access, n-gram language model

1. Introduction

The emergence of big data is about to change everything, in research domains and industries like genomics, e-commerce, Internet search, automated translation, social networking and more. One particular field getting a boost in recent years from massive datasets is linguistics. Among those corpora a notable example is the collection of Google Books Ngrams, made freely available in July 2009 [13].

The Google Books Ngrams contain n-word phrase statistics, for n from 1 to 5 (inclusive), for a few popular languages. Those statistics are obtained from over 5 million digitized books, corresponding to about 4% of all books ever published [5]. In fact, over 15 million books were digitized in the project, but only those with high enough OCR quality and available metadata (data and place of publication) were retained.

* Lodz University of Technology, Institute of Applied Computer Science, al. Politechniki 11, 90-924 Łódź, Poland; email: sgrabow@kis.p.lodz.pl.

** University of Szczecin, Institute of Information Technology in Management, Mickiewicza 64, 71-101 Szczecin, Poland; e-mail: jakubs@uoo.univ.szczecin.pl.

The oldest books are from the 16th century, but of course only the last 60 years or so are responsible for most of its volume; in particular, the number of words in publications from 2000 in the collection is about 8 billion, while it is only 1.4 billion for the year 1900. Many more curiosities and statistics can be found in reference [5].

It can be argued that analysis of the corpus can impact fields as diverse as linguistics, sociology, cultural history, shedding light on lexicography, the evolution of grammar (e.g., which irregular English verbs tend to become regular, over time), censorship in past decades, adoption of technology, surging and declining trends and fashions in society. Some interesting findings: (i) people become celebrities faster than in the past, but they are also more quickly forgotten, (ii) one can track banned persons, like Marc Chagall, whose full name appeared in the German corpus in years 1936–44 only once, to be followed with almost 100 occurrences in 1946–54 in the same corpus; (iii) peaks in "influenza" correlate with dates of known pandemics [5]. It is a truism to say that what we speak (and write) reflects what and how we think. Or, stressing slightly different aspects of the same thing, we could repeat after Wittgenstein: *Die Grenzen meiner Sprache bedeuten die Grenzen meiner Welt* ("The limits of my language mean the limits of my world").

If Google Books Ngrams are a great gift for many communities of researchers (and they are), what is bothering us? Two things. One is that the collection is distributed in highly inefficient textual form, compressed with a general-purpose compressor, zip. The other (and more important) is that the raw form does not support any efficient way for queries. The n-grams in files are lexicographically sorted, but there is no index, let alone a possibility to keep a significant fraction of the corpus in the RAM memory even of a high-end PC.

The contributions is this work are thus two-fold. We present a simple specialized compression scheme for the n-gram data, resulting in almost 3.4 times higher compression ratio than their original distribution. Moreover, our format supports efficient n-gram search (for relatively small penalty in compression ratio), which is experimentally validated.

2. Related work

We are not aware of any compression attempts for the Google Books Ngrams data. Still, other large linguistic collections, in particular the (commercial) Google 1T 5-gram corpus with Web data [19], Microsoft Web N-gram Language Models [17] and English Gigaword corpus of newswire [20] have been objects of interest. Large language models have been shown to be beneficial e.g. for statistical machine translation [1], search query spelling correction [6] and search query reformulation [2].

It is possible to search for trends in phraseology via Google viewer [13], but such a tool (as any Web-based tool) cannot scale to thousands of queries from a single researcher, i.e., serves mostly as a demonstration. For instance, Figure 1 illustrates how the numbers of occurrences for two phrases (in this example, 2-grams) change over 100 years.

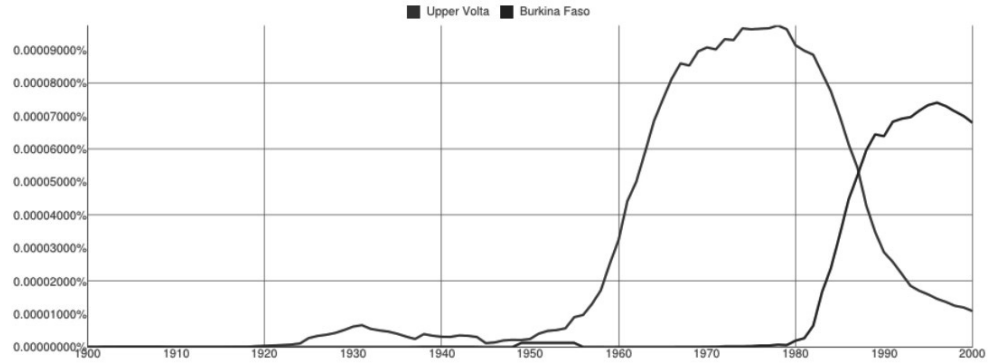

Figure 1. Occurrences of 'Upper Volta' vs. 'Burkina Faso' between 1900 and 2000

Huge language models are precious, to mention only the finding of Brants et al. [1], that machine translation quality continues to improve even when the text over which the language model is built grows beyond 1 trillion tokens. Talbot and Brants [10] compress the Google Web 1T 5-gram corpus to slightly over 3 bytes per *n*-gram, with a randomized solution (i.e., allowing errors with a small probability). Guthrie et al. [4] improve this result to 2.5 bytes per *n*-gram, also with a probabilistic solution, based on perfect hashing; even less is needed if the *n*-gram frequency counts are quantized. It is notable however the best "error-free" algorithm from [10] requires about 60% more space (i.e., above 5 bytes per *n*-gram). Pauls and Klein [7] store the same dataset, containing 4 billion *n*-grams, with associated frequency counts, in 23 bits per *n*-gram, in a lossless scheme. About twice larger representation allows them to handle queries (without caching) in about half a microsecond on average.

3. The characteristics of Google n-gram dataset structure

The Google *n*-gram datasets consist of lines with tab-separated textual fields. The first field in every line is the *n*-gram itself, that is a phrase of *n* successive words from a reference text. Punctuations in the collections are generally preserved, i.e., the phrase 'run,' is treated as a bigram: 'run' + ',' (comma). More on this issue can be found in [15].

Each *n*-gram (let us denote it with f_1) is followed by four fields that describe it:

- f_2 – the year in which the *n*-gram f_1 occurred in print (in the book collection scanned by Google, of course),
- f_3 – the total number of occurrences of *n*-gram f_1 in the year f_2,
- f_4 – the number of distinct book pages having occurrences of *n*-gram f_1 in books published in the year f_2,
- f_5 – the number of distinct books having occurrences of *n*-gram f_1 and published in the year f_2.

Observing the dataset contents [14], it is easy to notice the following properties:

- the order of lines is that for any interval of lines with f_1 unchanged, the values of f_2 are usually increasing;

- for any line, the total number of occurrences is larger or equal than the number of distinct book pages, which in turn is larger or equal than the number of distinct books, $f_3 \geq f_4 \geq f_5$.

4. The algorithm

4.1. The general concept

There are two key requirements for the algorithm: effective compression and random access to specified n-gram. Following our experience with XML compression [9] we expect the former to be achievable by compressing different types of data separately, whereas following our experience with URL collection compression [3] we assume the latter can be accomplished by dividing n-gram datasets into small blocks and compressing them separately.

4.2. Compression algorithm

The compression algorithm can be described as follows:
1) Open the dataset containing *n*-grams
2) Read *n*-gram *x*
3) If *x* is the first *n*-gram of a group,
   a) store it in the index file
4) Otherwise,
   a) encode and store it in the *n*-grams buffer
5) For every data element accompanying the *n*-gram:
   a) Encode it into respective buffer
6) If the number of entries in the current group surpasses defined group size
   a) For every buffer:
      i) Compress the data it contains using a LZ77-based [12] algorithm
      ii) Flush the compressed data to the main file
      iii) Append the current main file position to the index file
7) Go back to step (2) unless the end of file is encountered.

There are eight buffers in use during compression: *n-gram*, *entry count*, *first year*, *subsequent year*, *match count*, *page count*, *volume count* and *low bits*.

The first buffer is for the actual *n*-grams (phrases of words). They are encoded as strings of characters (no word substitution applied) with front-coding [11]. In case there are multiple occurrences of the same *n*-gram (note that they all must be next to each other, as the input file is sorted), the *n*-gram is encoded only once, whereas the number of repetitions is encoded in the *entry count* buffer as a decimal (see Table 1 for detailed information on the encoding scheme used for each data type).

Table 1. Used encoding schemes

Data type	Value (x)	Approximately encoded as (bytes)	Into buffer
n-gram	(group begin)	string of characters	*index file*
	(not group begin)	string of characters with front-coding	*n-gram*
entry count	$x \le 127$	x	*entry count*
	$128 \le x \le 16383$	$128+x$(*bits 7..13*),x(*bits 0..6*)	*entry count*
	$x \ge 16384$	$(x/16384)^{x}128$,$128+x$(*bits 7..13*),x(*bits 0..6*)	*entry count*
year (first for n-gram)	$x \le 1820$	$2(1885-x)-1$ (*bits 8..14*) $2(1885-x)-1$ (*bits 0..7*)	*first year* *low bits*
	$1821 \le x \le 1884$	$2(1885-x)-1$ (*bits 0..7*)	*first year*
	$1885 \le x \le 1948$	$2(x-1885)$ (*bits 0..7*)	*first year*
	$x \ge 1949$	$2(x-1885)$ (*bits 8..14*) $2(x-1885)$ (*bits 0..7*)	*first year* *low bits*
year (not first for n-gram)	$x-x_{-1} \le 63$	$x-x_{-1}$	*subs. year*
	$x-x_{-1} \ge 64$	$64\cdot\lceil \log_{256}(x-x_{-1})\rceil+(x-x_{-1})$(*upper 6 bits*) $\lceil \log_{256}(x-x_{-1})\rceil^{x}$ $(x-x_{-1})$(*lower 8 to 24 bits*)	*subs. year* *low bits*
match count (mc)	$x \le 64$	$x-1$	*match cnt.*
	$x \ge 65$	$64\cdot\lceil \log_{256}(x-1)\rceil+(x-1)$(*upper 6 bits*) $\lceil \log_{256}(x-1)\rceil^{x}$ $(x-1)$(*lower 8 to 24 bits*)	*match cnt.* *low bits*
page count (pc)	$x=1$	0	*page cnt.*
	$mc-x \le 62$	$1+mc-x$	*page cnt.*
	$mc-x \ge 63$	$64\cdot\lceil \log_{256}(1+mc-x)\rceil+(1+mc-x)$(*upper 6 bits*) $\lceil \log_{256}(1+mc-x)\rceil^{x}$ $(1+mc-x)$(*lower 8 to 24 bits*)	*page cnt.* *low bits*
volume count	$pc-x \le 63$	$pc-x$	*volume cnt.*
	$pc-x \ge 64$	$64\cdot\lceil \log_{256}(pc-x)\rceil+(pc-x)$(*upper 6 bits*) $\lceil \log_{256}(pc-x)\rceil^{x}$ $(pc-x)$(*lower 8 to 24 bits*)	*volume cnt.* *low bits*

Remark: x_{-1} denotes previously encoded value of x.

4.3. Search algorithm

The search algorithm for n-gram x can be described as follows:

1) Open the index file
2) Search the index for the group g that possibly contains x (using binary search)
3) Position the main file pointer at the group g beginning
4) Read and decompress n-gram buffer of the group g
5) Sequentially decode n-grams and check for x
6) If x has been found at position p:
   a) Read and decompress remaining buffers of the group g
   b) Decode $p-1$ data of n-grams preceding x in the group g
   c) Decode and return x data
7) Otherwise, return search failure

Note that only a small part of the compressed file has to be read and decompressed to return data related to a specified n-gram, and even smaller part of the compressed file has to be read and decompressed if the n-gram being searched for does not exist in the dataset.

5. Empirical evaluation

5.1. Proof-of-concept implementation

The algorithm described in Section 4 has been implemented by the second author in Turbo Delphi. The Deflate compression algorithm has been used for compression (as implemented in Delphi Zlib library [18]). The executables used in the tests can be downloaded from the following web address: http://iiwz.wneiz.pl/jakubs/progs/ngram_compressor.zip [16].

5.2. Test data sets

The Google Books N-grams datasets were downloaded from http://books.google.com/ngrams/datasets (July 2009 version) [14]. Although their content looks ordered, it is not guaranteed to be so, therefore all files used in the tests were sorted prior to compression using the sort tool included in [16].

We have to note that the Ngram corpus is unfortunately contaminated with errors, such as missing characters and unusual repetitions of the same n-grams. For the sake of achieving comparable results we decided not to fix any of its defects, and compressed the data as they were.

For the tests, datasets in English ('all' version), French and German were selected. Each of these sets consists of ten to hundreds of files for each phrase length from 1 to 5 words. We have observed that although compression ratio depends on the language and the number of words in phrases, the ratios measured for files of the same language and number of words were very highly consistent. For instance, we have calculated the coefficients of variation for groups of several hundred German datasets and they all were below 0.1%. It means that a single file from a given group is representative for the entire group. In our tests we have used the following test files for each of the three languages: #5 for 1-grams, #42 for 2-grams, #83 for 3-grams, #164 for 4-grams, and #325 for 5-grams. The numbers approxi-mate the 3/5 of the entire collection, whereas the last digit (for 2…5-grams) reminds the phrase length.

5.3. Test environment

The test platform was a desktop PC featuring Intel Core i3-2120 (3.3 GHz) processor, a motherboard based on Intel B75M chipset, 8 GB of DDR3-1600 RAM, and Intel 330 solid state drive (120 GB). The operating system was 64-bit Windows 7. Notice that the algorithm was implemented in an environment that can only produce 32-bit code, which could negatively affect measured query times.

5.4. Compression test results

Table 2 shows bit rates attained by applying the proposed algorithm (with the default compression settings: Deflate mode 6) to the chosen test files, grouping *n*-grams for random access decompression by respectively: 1000, 2000, and 4000 entries. For a comparison, we also list the bit rate of the individual data streams compressed as a whole (no groups for random access decompression; the *Streams* column), and the bit rate of the original Zip file (as they are currently distributed; the last column).

Table 2. Attained bit rates in output bits per input character

File name	1000	2000	4000	Streams	Zip
eng-1gram	0.709	0.670	0.648	0.618	1.700
eng-2gram	0.480	0.450	0.433	0.406	1.320
eng-3gram	0.380	0.354	0.339	0.314	1.136
eng-4gram	0.330	0.307	0.294	0.270	1.030
eng-5gram	0.306	0.285	0.272	0.248	0.960
fre-1gram	0.677	0.641	0.621	0.593	1.657
fre-2gram	0.448	0.420	0.404	0.379	1.274
fre-3gram	0.362	0.337	0.323	0.300	1.102
fre-4gram	0.326	0.302	0.289	0.266	1.016
fre-5gram	0.308	0.285	0.273	0.249	0.959
ger-1gram	0.594	0.560	0.542	0.514	1.491
ger-2gram	0.412	0.384	0.368	0.342	1.185
ger-3gram	0.349	0.324	0.309	0.284	1.064
ger-4gram	0.314	0.291	0.277	0.252	0.994
ger-5gram	0.304	0.282	0.269	0.243	0.960
Average	**0.420**	**0.393**	**0.377**	**0.352**	**1.190**

As the reader may observe, the proposed algorithm allows to store *n*-gram data in memory of capacity equal to about 5% of their initial volume. Looking at the *Streams* column, it can be observed that the feature of retrieving specified *n*-gram data without decompressing the whole file costs, in terms of degraded compression effectiveness, on average, from 7% (in case of the largest group size tested) to 19% (in case of the smallest group size tested). Even so, a significant improvement has still been achieved for each of the tested datasets compared the original Zip format. Figure 2 shows the actual improvement measured.

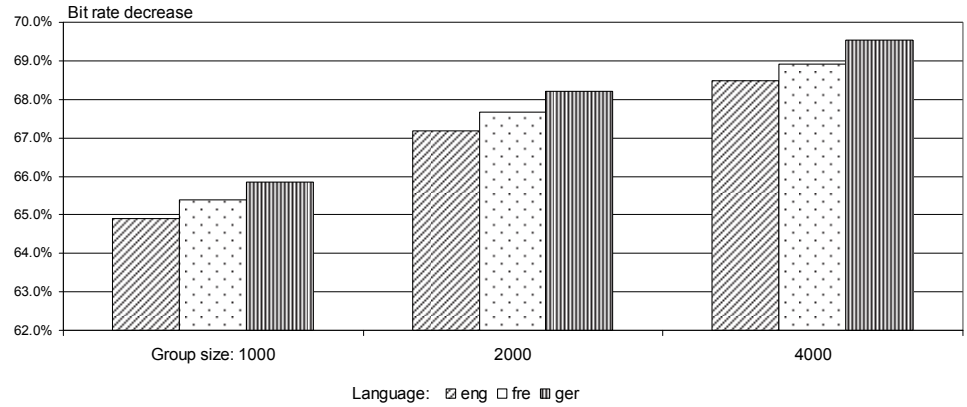

Figure 2. Measured improvement in the bit rate in relation to the original Zip format

5.5. Search test results

Table 3 shows average query execution times measured on the test data sets for around 200 phrases (chosen randomly using –rg option of the software used in the tests [16]). The measurements were done internally (–rd option), therefore they include only the sole query execution time (including saving the query results to a file, –f option), not the program start-up time.

Table 3. Extract time and extracted data size measurements

File name	Extract time (ms)			Extracted data (kB)		
	1000	**2000**	**4000**	**1000**	**2000**	**4000**
eng-1gram	0.160	0.193	0.269	3.86	7.54	14.86
eng-2gram	0.148	0.192	0.275	3.81	7.42	14.66
eng-3gram	0.180	0.228	0.310	3.84	7.40	14.42
eng-4gram	0.190	0.227	0.292	3.84	7.17	14.20
eng-5gram	0.139	0.190	0.286	3.66	7.26	14.38
fre-1gram	0.155	0.198	0.278	3.85	7.47	14.79
fre-2gram	0.147	0.189	0.274	3.68	7.16	14.17
fre-3gram	0.148	0.193	0.270	3.60	6.99	13.85
fre-4gram	0.142	0.187	0.273	3.54	6.91	13.70
fre-5gram	0.148	0.209	0.293	3.56	12.33	19.10
ger-1gram	0.150	0.195	0.285	3.86	7.50	14.85
ger-2gram	0.143	0.189	0.266	3.65	7.18	14.15
ger-3gram	0.141	0.188	0.270	3.57	7.06	13.86
ger-4gram	0.146	0.191	0.303	3.73	7.19	24.01
ger-5gram	0.154	0.207	0.304	7.03	10.50	17.53
Average	**0.152**	**0.198**	**0.283**	**3.94**	**7.81**	**15.50**

With the default grouping of 2000 entries, the computer used in the tests would be able to handle about 5000 different queries per second. It seems to be a satisfactory value for typical real-world scenarios.

The extraction time obviously depends on the amount of data that need to be decompressed. Comparing left and right sides of Table 2, one can notice that the average 300% growth in extracted data volume between the 4000-entry and 1000-entry groups is accompanied by less than 100% increase in query execution time. It means that unless rigid query time limits are to be kept, it is reasonable to increase the group size for the sake of improved compression effectiveness (see Figure 3).

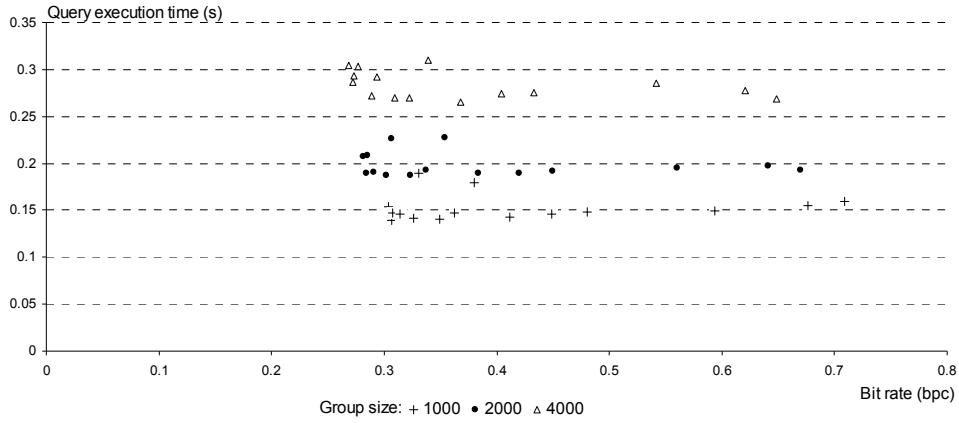

Figure 3. Measured query execution time versus attained bit rate

6. Conclusions

Gathering large amounts of data is not everything. Cleaning process is tedious and non-trivial task. For example, Procházka and Pollák [8] report issues with Google CzechWeb 1T n-Grams, e.g., a lot of problematic tokens (foreign words, or Czech words but without diacritical marks, or even semi-random character strings). We have also noticed some problems with Google Books Ngrams statistics, e.g. missing characters. Such issues are likely to hamper applications of huge corpora in machine translation, speech recognition etc.

Our task, however, was to deal with data at hand. "At hand" is spoken lightly, but the collections are not quite at hand, because of the multiple large volumes that must be downloaded to keep complete statistics for at least one language. This was a primary motivation for our research: show that the textual format used by Google is highly inefficient from the compression viewpoint, and even leaving Deflate (zip) as the backend compressor, we can squeeze those data much better (more precisely, about 3.4 times tighter).

Moreover, we added a simple index enabling to search quickly for a given n-gram, to obtain all its statistics (occurrences per year). Extraction times are usually within 0.2 ms. Future works should focus on handling more general queries, lossy compression (the exact

occurrence values are not necessary for all applications) and tests with storing a complete language n-gram database in main memory of a powerful PC, with 16–32 GB of RAM.

References

[1] Brants T., Popat A. C., Xu P., Och F. J., Dean J., Large language models in machine translation, in: *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, Prague, ACL 2007, 858–867.

[2] Gao J., Nguyen P., Li X., Thrasher C., Li M., Wang K., A Comparative Study of Bing Web N-gram Language Models for Web Search and Natural Language Processing, in: *Workshop of the 33rd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Geneva 2010.

[3] Grabowski Sz., Swacha J., Compact Representation of URL Collections with Fast Access, *Automatyka*, **15**, 3, 2011, 349–355.

[4] Guthrie D., Hepple M., Liu W., Efficient Minimal Perfect Hash Language Models, in: N. Calzolari, K. Choukri, B. Maegaard, J. Mariani, J. Odijk, S. Piperidis, M. Rosner, D. Tapias (eds.), *Proceeding of the Seventh International Conference on Language Resources and Evaluation*, Valetta, ELRA 2010.

[5] Michel J.-B. B., Kui Y., Presser A., Veres A., Gray M. K., Google Books Team, Picket J. P., Hoiberg D., Clancy D., Norvig P., Orwant J., Pinker S., Nowak M. A., Lieberman Aider E., Quantitative Analysis of Culture Using Millions of Digitized Books, *Science*, **331**, 6014, 2011, 176–182.

[6] Microsoft Research, Spelling Alteration for Web Search Workshop, City Center – Bellevue, WA, July 19, 2011. Materials available at http://web-ngram.research.microsoft.com/Spellerchallenge/Docs/Spelling_Alteration_Workshop.pdf (last checked: June 2012).

[7] Pauls A., Klein D., Faster and Smaller N-Gram Language Models, in: Y. Matsumoto, R. Mihalcea (eds.), *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies – Volume 1*, Stroudsburg, ACL 2011, 258–267.

[8] Procházka V., Pollák P., Analysis of Czech Web 1T 5-Gram Corpus and Its Comparison with Czech National Corpus Data, in: P. Sojka, A. Horák, I. Kopecek, K. Pala (eds.), *Proceedings of the 13th International Conference Text, Speech and Dialog*, Brno, Springer 2010, 181–188.

[9] Skibiński P., Grabowski Sz., Swacha J., Effective asymmetric XML compression, *Software–Practice and Experience*, **38**, 10, 2008, 1027-1047.

[10] Talbot D., Brants T., Randomized Language Models via Perfect Hash Functions, in: *Proceedings of the 46th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, Columbus, ACL 2008, 505–513.

[11] Witten I. H., Moffat A., Bell T. C., *Managing Gigabytes: Compressing and Indexing Documents and Images*, Morgan Kaufmann Publishers, Los Altos, 1999.

[12] Ziv, J., Lempel, A., A Universal Algorithm for Sequential Data Compression, *IEEE Transactions on Information Theory*, **23**, 3, 1977, 337–343.

[13] http://books.google.com/ngrams (last checked: June 2012).

[14] http://books.google.com/ngrams/datasets (last checked: June 2012).

[15] http://books.google.com/ngrams/info (last checked: June 2012).

[16] http://iiwz.wneiz.pl/jakubs/progs/ngram_compressor.zip (last checked: June 2012).

[17] http://research.microsoft.com/en-us/collaboration/focus/cs/web-ngram.aspx (last checked: June 2012).

[18] http://www.base2ti.com (last checked: June 2012).

[19] http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC2006T13 (last checked: June 2012).

[20] http://www.ldc.upenn.edu/Catalog/catalogEntry.jsp?catalogId=LDC2011T07 (last checked: June 2012).

Presented at the Congress of Young IT Scientists, Międzyzdroje, Poland, 20-22.09.2012