Efficient Data Structures for Massive N-Gram Datasets*

Giulio Ermanno Pibiri and Rossano Venturini

Department of Computer Science, University of Pisa, Italy
giulio.pibiri@di.unipi.it, rossano.venturini@unipi.it

Abstract. In this paper we summarize the experimental results of [7], whose main proposal is a compressed trie index for N-grams in which each word is encoded as an integer value proportional to the number of words that follow a context of fixed length. Since the number of words following a given context is typically very small in natural languages, we are able to lower the space of representation to compression levels that were never achieved before. Despite the significant savings in space, we show that our technique introduces a negligible penalty at query time.

1 Elias-Fano Tries with Context-based ID Remapping

The problem we consider in this paper is the one of indexing massive N-gram datasets in compressed space, i.e., how to efficiently represent the map from the N-gram strings to their corresponding frequency counts, so that we can retrieve a count by means of a `Lookup` operation. In the remaining of the section, we sketch the main design of the *trie* data structure proposed in [7].

Uni-grams are indexed using a minimal and perfect hash data structure that stores for each string its integer ID in order to retrieve it in $\mathcal{O}(1)$ worst-case. We sort all k-grams, for $1 \leq k \leq N$, following the token-ID order: the k-th level of the trie is formed by the sequence of IDs corresponding to the k-th tokens of the k-grams. Now, the k-grams sharing the same prefix of length $k-1$ form a strictly increasing *range* of (sibling) IDs. We transform each level of the trie into a *monotone* sequence by taking *range-wise* prefix sums. In order to distinguish the successors of a gram from others, we also maintain where each range begins in another monotone integer sequence of pointers. We compress these two sequences with Elias-Fano in order to support efficient searches.

To further reduce the space taken by the token-ID sequences, we map a token ID to the position it occupies within its siblings. We call this technique *context-based remapping* as each ID is mapped to the position it takes relatively to a context. As the range of a gram is much smaller than the whole vocabulary size (number of uni-grams), we shrink the magnitude of the integers that form the levels of the trie. In our case, a token-ID sequence of length n and universe u is encoded with Elias-Fano: this encoding spends $\lceil \log \frac{u}{n} \rceil + 2$ bits per integer, thus a number of bits proportional to the *average gap* (u/n) between the integers. Remapping them as explained reduces the universe u of representation, *hence* lowering the average gap and space of the sequence.

* The full version of this paper will be presented at the 40-th ACM SIGIR [7].

		Europarl		YahooV2		GoogleV2	
		bpg	μs × query	bpg	μs × query	bpg	μs × query
	EF	1.97	1.28	2.17	1.60	2.13	2.09
	PEF	1.87 (−4.99%)	1.35 (+5.93%)	1.91 (−12.03%)	1.73 (+8.00%)	1.52 (−28.60%)	1.91 (−8.79%)
CONTEXT-BASED ID REMAPPING $k=1$	EF	1.67 (−15.30%)	1.58 (+23.86%)	1.89 (−12.92%)	2.05 (+28.07%)	1.91 (−10.24%)	3.03 (+44.61%)
	PEF	1.53 (−22.36%)	1.61 (+25.89%)	1.63 (−24.91%)	2.16 (+35.22%)	1.31 (−38.71%)	2.30 (+9.88%)
$k=2$	EF	1.46 (−25.62%)	1.60 (+25.17%)	1.68 (−22.32%)	2.08 (+30.23%)	—	—
	PEF	1.28 (−34.87%)	1.64 (+28.12%)	1.38 (−36.15%)	2.15 (+34.81%)	—	—

Table 1. Average bytes per gram (bpg) and average Lookup time per query in micro seconds.

2 Experiments

We performed the experiments on the following standard datasets: Europarl [4], YahooV2 [1] and GoogleV2 [2] that include, respectively, 101 428 257, 828 223 677 and 11 131 242 087 N-grams. We compare the performance of our data structures against the following software packages: BerkeleyLM [6]; Expgram [9]; KenLM [3]; Marisa [10]; RandLM [8].

All experiments have been performed on a machine with 16 Intel Xeon E5-2630 v3 cores (32 threads) clocked at 2.4 Ghz, with 193 GBs of RAM, running Linux 3.13.0, 64 bits. Our implementation is in standard C++11 and freely available at `https://github.com/jermp/tongrams`. The code was compiled with gcc 5.4.1, using the highest optimization settings. To test the speed of lookup queries, we use a query set of 5 million random N-grams for YahooV2 and GoogleV2 and of 0.5 million for Europarl. In order to smooth the effect of fluctuations during measurements, we repeat each experiment five times and consider the mean.

2.1 Elias-Fano Tries

In this subsection we analyze our proposed Elias-Fano trie data structure, when the levels of the trie are encoded with plain Elias-Fano (EF) or partitioned Elias-Fano (PEF) [5]. Refer to Table 1. In particular, for GoogleV2 we use a context of length $k = 1$, as the tri-grams alone take 66% of the whole dataset and, therefore, should be kept remapped. As we can see by the second row of Table 1, partitioning the gram sequences using PEF yields a better space occupancy with respect to EF (up to 29% on GoogleV2) and brings only a negligible overhead in query processing speed (less than 8% on Europarl and YahooV2).

Concerning the efficacy of the context-based remapping, we have that remapping the gram IDs with a context of length $k = 1$ is already able of reducing the space of the sequences by ≈13% on average when sequences are encoded with Elias-Fano, with respect to the EF cost. If we consider a context of length $k = 2$ we *double* the gain, allowing for more than 28% of space reduction *without* affecting the lookup time with respect to the case $k = 1$. As a first conclusion, when space efficiency is the main concern, it is always convenient to apply the

remapping strategy with a context of length 2. The gain of the strategy is even more evident with PEF (36% on average and up to 39% on GoogleV2): this is no surprise as the encoder can better exploit the reduced IDs by encoding all the integers belonging to a block with a universe relative to the block and not to the whole sequence. However, the remapping strategy comes with a penalty at query time as we have to map an ID before it can be searched in the proper gram sequence. On average, we found that 30% more time is spent with respect to the EF baseline. Notice that PEF does *not* introduce any time degradation with respect to EF with context-based remapping: it is actually faster on GoogleV2.

In the following subsection, we call PEF-Trie and PEF-RTrie the partitioned Elias-Fano trie without remapping and with remapping of order 2 respectively.

	Europarl		YahooV2		GoogleV2	
	bpg	μs × query	bpg	μs × query	bpg	μs × query
PEF-Trie	1.87	1.35	1.91	1.73	1.52	1.91
PEF-RTrie	1.28	1.64	1.38	2.15	1.31	2.30
BerkeleyLM C.	1.70 (−8.89%) (+32.90%)	2.83 (+108.88%) (+72.70%)	1.69 (−11.41%) (+22.04%)	3.48 (+101.84%) (+61.70%)	1.45 (−4.87%) (+10.83%)	4.13 (+116.57%) (+79.76%)
BerkeleyLM H.3	6.70 (+258.81%) (+423.40%)	0.97 (−28.46%) (−40.85%)	7.82 (+310.38%) (+465.36%)	1.13 (−34.35%) (−47.41%)	9.24 (+507.79%) (+608.07%)	2.18 (+13.95%) (−5.42%)
BerkeleyLM H.50	7.96 (+326.03%) (+521.45%)	0.97 (−28.49%) (−40.88%)	9.37 (+391.32%) (+576.87%)	0.96 (−44.27%) (−55.35%)	—	—
Expgram	2.06 (+10.18%) (+60.73%)	2.80 (+106.61%) (+70.82%)	2.24 (+17.36%) (+61.68%)	9.23 (+435.33%) (+328.87%)	—	—
KenLM T.	2.99 (+60.11%) (+133.56%)	1.28 (−5.47%) (−21.84%)	3.44 (+80.39%) (+148.52%)	1.94 (+12.32%) (−10.01%)	—	—
Marisa	3.61 (+93.09%) (+181.66%)	2.06 (+52.00%) (+25.67%)	3.81 (+99.60%) (+174.98%)	3.24 (+87.96%) (+50.58%)	—	—
RandLM	1.81 (−3.06%) (+41.41%)	4.39 (+224.20%) (+168.04%)	2.02 (+6.18%) (+46.29%)	5.08 (+194.35%) (+135.82%)	2.60 (+70.73%) (+98.90%)	9.25 (+384.54%) (+302.19%)

Table 2. Average bytes per gram (bpg) and average Lookup time per query in micro seconds. An empty entry means that was not possible to build the data structure due to memory constraints

2.2 Overall Comparison

In this subsection we compare the performance of our selected trie data structures, PEF-RTrie and PEF-Trie, against the competitors mentioned at the beginning of the section. The results of the comparison are shown in Table 2. For each competitor, we report two percentages indicating its score against our PEF-Trie and PEF-RTrie respectively.

BerkeleyLM compressed (C.) variant results ≈21% larger than the PEF-RTrie and slower by more than 70%. It gains, instead, an advantage of roughly 9% over the PEF-Trie data structure, but it is also more than 2 times slower. The HASH variant uses hash tables with linear probing to represent the nodes of the trie. Therefore, we test it with a small extra space factor of 3% for table allocation (H.3) and with 50% (H.50). Clearly the space occupancy of both hash variants do not compete with the ones of our proposals as these are from 3 to 7 times larger, but the $\mathcal{O}(1)$-lookup capabilities of hashing makes it faster than a

trie implementation. Expgram is ≈13.5% larger than PEF-Trie and also 2 and 5 times slower on Europarl and YahooV2 respectively. Our PEF-RTrie data structure retains an advantage in space of 60% and it is still significantly faster: of about 72% on Europarl and 4.3 times on YahooV2. KenLM is the fastest trie language model implementation in the literature. As we can see, our PEF-Trie variant retains 70% of its space with a negligible penalty at query time. Compared to PEF-RTrie, it results a little faster, i.e., ≈15%, but also 2.3 and 2.5 times larger on Europarl and YahooV2 respectively. We also tested the performance of Marisa even though it is not a trie optimized for language models as to understand how our proposals compare against a general-purpose string dictionary implementation. We outperform Marisa in both space and time: compared to PEF-RTrie, it is 2.7 times larger and 38% slower; with respect to PEF-Trie it is more than 90% larger and 70% slower. RandLM is designed for small memory footprint and returns approximated frequency counts when queried. While being from 2.3 to 5 times slower than our exact and lossless approach, it is quite compact because the quantized frequency counts are recomputed on the fly [8]. Therefore, while its space occupancy results even larger with respect to our grams representation by 61%, it is still no better than the whole space of the PEF-RTrie data structure. With respect to the whole space of PEF-Trie, it retains instead an advantage of ≈15.6%.

References

[1] Yahoo! n-grams, version 2.0. 2006. URL: `http://webscope.sandbox.yahoo.com/catalog.php?datatype=l`.

[2] Thorsten Brantz and Alex Franz. The Google Web 1T 5-Gram Corpus. In *Technical Report LDC2006T13*, 2006. URL: `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`.

[3] Kenneth Heafield. Kenlm: Faster and smaller language model queries. In *WMT*, pages 187–197, 2011. URL: `http://kheafield.com/code/kenlm`.

[4] Philipp Koehn. Europarl: A parallel corpus for statistical machine translation. In *MT summit*, pages 79–86, 2005. URL: `http://www.statmt.org/europarl`.

[5] Giuseppe Ottaviano and Rossano Venturini. Partitioned Elias-Fano Indexes. In *SIGIR*, pages 273–282, 2014.

[6] Adam Pauls and Dan Klein. Faster and smaller n-gram language models. In *ACL*, pages 258–267, 2011. URL: `https://github.com/adampauls/berkeleylm`.

[7] Giulio Ermanno Pibiri and Rossano Venturini. Efficient Data Structures for Massive N-Gram Datasets. In *SIGIR*, pages –, 2017.

[8] David Talbot and Miles Osborne. Randomised language modelling for statistical machine translation. In *ACL*, pages 512–519, 2007. URL: `https://sourceforge.net/projects/randlm`.

[9] Taro Watanabe, Hajime Tsukada, and Hideki Isozaki. A succinct n-gram language model. In *IJCNLP*, pages 341–344, 2009. URL: `https://github.com/tarowatanabe/expgram`.

[10] Susumu Yata. Prefix/patricia trie dictionary compression by nesting prefix/patricia tries. In *NLP*, 2011. URL: `https://github.com/s-yata/marisa-trie`.