A Space-Efficient Phrase Table Implementation Using Minimal Perfect Hash Functions

Marcin Junczys-Dowmunt

Faculty of Mathematics and Computer Science
Adam Mickiewicz University
ul. Umultowska 87, 61-614 Poznań, Poland
junczys@amu.edu.pl

Abstract. We describe the structure of a space-efficient phrase table for phrase-based statistical machine translation with the Moses decoder. The new phrase table can be used in-memory or be partially mapped on-disk. Compared to the standard Moses on-disk phrase table implementation a size reduction by a factor of 6 is achieved.

The focus of this work lies on the source phrase index which is implemented using minimal perfect hash functions. Two methods are discussed that reduce the memory consumption of a baseline implementation.

Keywords: Statistical machine translation, compact phrase table, minimal perfect hash function, Moses.

1 Introduction

As the size of parallel corpora increases, the size of translation phrase tables used for statistical machine translation extracted from these corpora increases even faster. The current in-memory representation of a phrase table in Moses [1], a widely used open-source statistical machine translation toolkit, is unusable for anything else but toy-size translation models or prefiltered test set data. A binary on-disk implementation of a phrase table is generally used, but its on-disk size requirements are significant.

This paper continues the research of [2] towards a compact phrase table that can be used as a drop-in replacement for both, the binary phrase table implementation and the in-memory phrase table of Moses. An important requirement is the faithful production of translations identical to translations generated from the original phrase table implementation if the same phrases, scores, and settings are provided. The phrase table design is inspired by the architecture of large-scale n-gram language models which employ minimal perfect hash functions (MPH) as their main indexing structure. In this paper we investigate two methods to reduce the size of a baseline source phrase index implementation.

2 Related Work

Zens and Ney [3] describe a phrase table architecture on which the binary phrase table of Moses is based. The source phrase index consists of a prefix tree. Memory requirements are low due to on-demand loading. Disk space requirements however can become substantial. A suffix-array based implementation of a phrase table for Moses that can create phrase pairs on-demand is introduced by Levenberg et. al [4]. While it surely is a promising alternative, we do not compare this approach with ours, as they differ in assumptions.

Other approaches, based on phrase table filtering [5], can be seen as a type of compression. They reduce the number of phrases in the phrase table by significance filtering and thus reduce space usage and improve translation quality at one stroke.

The architecture of the source phrase index of the discussed phrase table has been inspired by the efforts concerned with language model compression and randomized language models [6,7]. Guthrie et. al [7] who describe a language model implementation based on a minimal perfected hash function and fingerprints generated with a random hash function is the greatest influence.

3 Experimental Data

The presegmented version of Coppa, the Corpus Of Parallel Patent Applications [8], a parallel English-French corpus of WIPO's patent applications published between 1990 and 2010, is chosen for phrase table generation. It comprises more than 8.7 million parallel segments with 198.8 million English tokens and 232.3 million French tokens. The phrase table that is used throughout this paper has been created with the standard training procedure of Moses. Word alignment information is included. There are ca. 2.15×10^8 distinct source phrases and 3.36×10^8 phrase pairs.

4 Compact Phrase Table Implementation

Figure 1 illustrates the architecture of the discussed phrase table implementation. One important design guideline is the representation of most of the data in plain array structures that can be either fully read into memory or directly mapped from disk to memory. The phrase table consists of three main modules that are described in more detail in the following subsections.

4.1 Source Phrase Index

The source phrase index assigns integer positions to source phrases. As mentioned above, its structure is inspired by [7] who use a similar implementation for huge n-gram language models. The most important part of the index is a minimal perfect hash function (MPH) that maps a set S of n source phrases to n consecutive integers. This hash function has been generated with the CHD algorithm included in the freely available CMPH[1] library [9]. The CHD algorithm generates very small MPH (in this case 109 Mbytes) in linear time.

[1] `http://cmph.sourceforge.net`

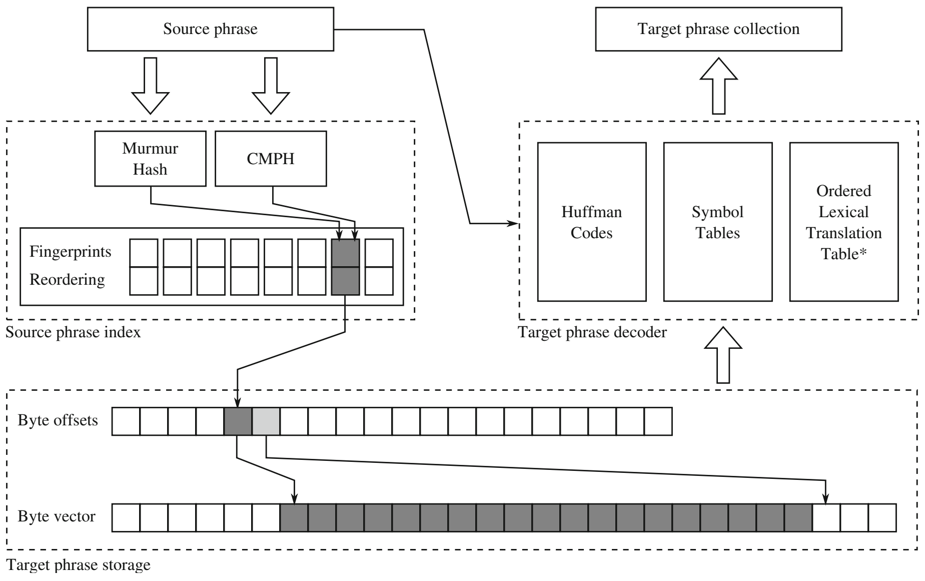

Fig. 1. Simplified phrase table implementation schema

The MPH is only guaranteed to map known elements from S to their correct integer identifier. If a source phrase is given that has not been seen in S during the construction of the MPH, a random integer will be assigned to it. This can lead to false assignments of target phrase collections to unseen source phrases, so-called *false positives*. Guthrie et. al [7] propose to use a random hash algorithm (MurmurHash3[2]) during construction and store its values as fingerprints for each phrase from S. For querying, it suffices to generate the fingerprint for the input phrase and compare it with the fingerprint stored at the position returned by the MPH function. If it matches, the phrase has been seen and can be further processed. For 32 bit fingerprints there is a probability of 2^{-32} of an unseen source phrase slipping through. Fingerprints are stored in an array at the position assigned to the phrase by the MPH. Since for each source phrase one fingerprint has to be stored, the array of fingerprints uses around 820 Mbytes.

MPHs generated using the CHD algorithm are not order-preserving, hence the original position of the source phrase is stored together with each fingerprint. Results for lexicographically ordered queries lie close or next to each other. If the data is stored on disk, this translates directly to physical proximity of the data chunks on the drive and less movement of the magnetic head. Without order-preservation the positions assigned by the MPH are random which can render a memory-mapped version of the phrase table near unusable. For each source phrase a 32 bit integer reordering position is stored, this consumes another 820 MBytes. In total the source phrase index consumes 1,750 MBytes. In this paper we examine two methods that allow to shrink memory consumption in comparison to the described baseline.

[2] `http://code.google.com/p/smhasher/wiki/MurmurHash3`

4.2 Target Phrase Storage

The target phrase storage contains collections of target phrases at position assigned by the index. It consists of a byte vector that stores target phrase collections consecutively according to the order of their corresponding source phrases. A target phrase collection consists of one or more target phrases. A target phrase is a sequence of target word symbols followed by a special stop symbol, a fixed-length sequence of scores, and a sequence of alignment points followed again by a special stop symbol.

Random access capability is added by the byte offset vector. For every target phrase collection, it stores the byte offset at which this collection starts. By inspecting the next offset the end position of a target phrase collection can be determined. While the byte vector is just a large array of bytes, the byte offset vector is a more sophisticated structure. Instead of keeping offsets as 8-byte integers[3], differences between the offsets are stored. A synchronization point with the full offset value is inserted and tracked for every 32 values.[4] This turns the byte offset vector into a list of rather small numbers, even more so when the byte array is compressed. Techniques from inverted list compression for search engine indexes are used to reduce the size further: Simple-9 encoding for offset differences and Variable Byte Length encoding for synchronization points.

Size reduction is achieved by compressing the symbol sequence of a target phrase collection, for instance using symbol-wise Huffman coding. The number of Huffman codes can become large for a particular type of symbols, e.g. there are nearly 13.6 million distinct scores (32 bit floats). Three different sets of Huffman codes are used to encode target phrase words, scores, and alignment points. In every set are as many codes as distinct symbols of the encoded type. This variant of the phrase table is denoted as “Baseline”. Another encoding schema — Rank-Encoding (denoted “R-Enc”) — has been proposed by [2]. Target phrase words are encoded using the positions of source phrase words they are aligned with. This reduces the entropy of target words and results in shorter Huffman codes. See Table 2 for the size characteristics of both these variants.

4.3 The Phrase Decoder

The target phrase decoder contains the data that is required to decode the compressed byte streams. It includes source and target word lists with indexes, the Huffman codes, and if Rank Encoding is used a sorted lexical translation table. Huffman codes are stored as canonical Huffman codes, a memory efficient representation.

5 Memory Requirements for Order Preservation

Order preservation comes at a cost of 4 bytes (32 bits) per source phrase. This can be significantly reduced if the order itself is exploited. For lexicographically sorted phrases, the cost for a chunk of order information can be reduced to n bits if every 2^n-th source phrase is stored explicitly. We call these phrases *landmarks*. That way

[3] 4-byte integers could hold byte offsets up to 4 Gbytes only.

[4] This is an arbitrarily set step size.

a partition of 2^n equally[5] sized ranges on the set of sorted source phrases S is created. For each range an MPH is generated together with corresponding fingerprint arrays and reordering arrays. Landmarks are stored in a sorted structure and retrieved by binary search.

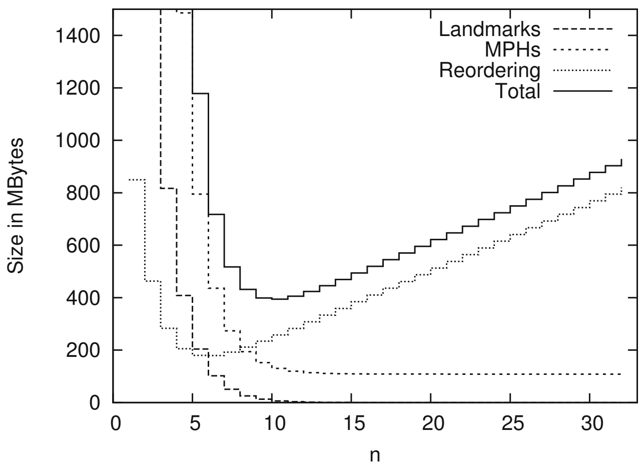

Fig. 2. Memory-requirement for chosen n (Coppa phrase-table)

Given a source phrase s located in the sorted phrase table at position p, the value $\hat{p}$ in the reordering array corresponding to s is calculated as $\hat{p} = p \bmod 2^n$. This value is always smaller than 2^n and can be stored in a n-bit sized storage.

Querying is done as follows: For a phrase s, the largest i is determined among the sorted landmark phrases, for which lexicographically $s_i \leq s$. The phrase s is hashed using the i-th MPH which assigns the precalculated fingerprint and the position $\hat{p}$ to s. Using MurmurHash3 a fingerprint is calculated for s. If fingerprints match, the original position $p = 2^n i + \hat{p}$ is returned.

Fig. 2 depicts the relation between n and memory requirements for landmark phrases, reordering information, and MPHs using the example of the Coppa phrase table. Fingerprints are omitted since their memory usage is fixed at 820 MBytes. The minimum of 393 MBytes is reached for $n = 10$ with 1024 phrases per range. For smaller n the space usage of the MPHs and the landmark phrases is growing rapidly. This is due to the exponential increase in number of landmarks, MPHs, and reordering arrays for small n. Overhead per MPH is ca. 100 bytes, per reordering array 8 bytes. For $n = 8$ memory consumption (425 MBytes) is slightly increased in comparison to $n = 10$, but full byte values are easier stored. This variant is denoted as "Ord-8". It can be safely assumed that a choice of n between 8 and 12 results in general in a significant space reduction.

[5] The last range may be smaller.

6 Memory-Requirements for Fingerprints

6.1 Estimating an Optimal Fingerprint Size

Fingerprint bit length is a certain candidate for optimization, but this may affect translation quality. In this section a method is described that allows to estimate a safe and space-optimal bit number *before* the construction of the compact phrase table using the text version of the phrase table.

When creating an SMT system, typically in-domain test sets or development sets distinct from the training data are available. Such a sample text can be used to create a set P of all phrases up to a fixed length[6] occurring in that text. During the translation of the sample, the phrase table is being queried with all phrases from P. As introduced above, S is the set of all source phrases present in the phrase table. The intersection $P \cap S$ is the set of phrases from the sample for which translations exist. Conversely, $P \setminus S$ is the set of unseen phrases that have no translation. The following sums are defined:

$$C_P = \sum_{p \in P} c(p), \quad C_S = \sum_{p \in P \cap S} c(p), \quad C_U = \sum_{p \in P \setminus S} c(p) = C_P - C_S \tag{1}$$

where $c(p)$ is the number of occurrences of a phrase p in the sample. C_P is the sum of occurrences of all phrases in the sample, C_S counts occurrences of all seen sample phrases, C_U is the number of occurrences of all unseen sample phrases.

$C_{\mathrm{fp}}(b)$ is the observed number of false positives that occur during the translation of the sample text if a fingerprint size of b is chosen. Dividing $C_{\mathrm{fp}}(b)$ by N — the number of sentences in the sample text — a more comparable observed per-sentence false positive ratio denotated as ofpr(b) is obtained. This value is not known yet, but the expected value can be calculated as follows:

$$\mathrm{efpr}(b) = E\left[\mathrm{ofpr}(b)\right] = E\left[\frac{C_{\mathrm{fp}}(b)}{N}\right] = \frac{C_U}{2^b N}. \tag{2}$$

By fixing efpr(b) at a desired value f, the number of fingerprint bits b is given as:

$$\mathrm{bits}(f) = \left\lceil \mathrm{efpr}^{-1}(f) \right\rceil = \left\lceil \log_2 \frac{C_U}{f \cdot N} \right\rceil. \tag{3}$$

WIPO released a distinct test set[7] for the COPPA corpus, for which the following values are obtained: $N = 7{,}122$, $C_P = 919{,}755$, $C_S = 487{,}137$, and $C_U = 432{,}618$. For this test set and a bit size of $b = 32$, a per-sentence false positive rate $\mathrm{efpr}(b) = 1.414 \times 10^{-8}$ is estimated, i.e. a false positive is expected to occur once every 71 million sentences. It seems reasonable to choose a smaller b, for instance, by fixing $\mathrm{efpr}(b) = 0.001$ (one false positive occurring every 1,000 sentences) which results in $\mathrm{bits}(0.001) = 16$. A 16 bit fingerprint reduces the size requirements by half.

[6] Standard settings in the Moses training procedure limit source and target phrases to 7 symbols. The decoder should be configured to use the same phrase length limit.

[7] `http://www.wipo.int/patentscope/translate/coppa/testSet2011.tmx.gz`

6.2 Fingerprint Size and Translation Quality

Table 1 compares the epfr(b) for chosen fingerprint sizes with observed false positive rates for the WIPO test set. Additionally, the obtained BLEU scores are given. The column "Used" contains the total number of sentences in which false positives actually surfaced causing an unwanted translation result.

Table 1. BLEU score in relation to expected and observed false positive rates

b	efpr(b)	ofpr(b)	Used	BLEU
32	$1.41 \cdot 10^{-8}$	0	0	46.86
24	$3.62 \cdot 10^{-6}$	0	0	46.86
20	$5.79 \cdot 10^{-5}$	$1.42 \cdot 10^{-4}$	1	46.86
16	$9.27 \cdot 10^{-4}$	$5.68 \cdot 10^{-4}$	3	46.86
12	$1.48 \cdot 10^{-2}$	$1.42 \cdot 10^{-2}$	36	46.79
8	0.24	0.25	494	46.02
4	3.80	3.78	3438	37.35

As expected, 32 bit down to 24 bit fingerprints are very successful in prohibiting false positives. For 20 bits a single false positive appears as a translation option and also surfaces in one translation (out of over 7,000). Similarly, 16 bit fingerprints offer an acceptable protection, only 3 sentences have been translated differently. BLEU scores remain unchanged for 20 bit and 16 bit fingerprints. For 12 bits there is a small decrease in BLEU that might still be acceptable for some applications. Bit lengths of 8 and lower have a serious effect on BLEU and should not be used. We choose 16 bit fingerprints for the Coppa phrase table. The translation quality seems acceptable and again full byte values can be used. This variant is denoted as "Fprt-16".

7 Summary and Future Work

Table 2 summarizes the memory consumption for variants of the presented phrase table. Variants to the right include the memory reduction methods of the previous variants. Figures for the Moses binary phrase table are given for comparison. The memory usage of the source phrase index was reduced by 52 percent without sacrificing translation quality in terms of BLEU. If target phrase compression techniques are applied (described in more depth in [2]), the whole phrase table is reduced by 34 percent compared to the baseline. The final variant of the described phrase table implementation uses only 17 percent disk space of the Moses binary phrase table.

Future work will address the problem of scores in the phrase table which currently consume the most space. For information on performance see [2]. First experiments suggest that the described phrase table allows much faster decoding than the Moses binary phrase table, especially on machines where disk I/O is a bottle neck. Also, the described source index implementation seems to be a good starting point for a distributed implementation.

Table 2. Comparison of phrase table implementations

	Moses	Baseline	R-Enc	Ord-8	Fprt-16
Total size in Mbytes :	29,418	7,681	5,967	5,463	5,052
Ordered source phrase index (Mbytes):	5,953	1,750	1,750	1,246	835
Bytes per source phrase:	29.1	8.5	8.5	6.1	4.1
Target phrase storage (Mbytes):	23,441	5,873	4,127	4,127	4,127
Target phrase decoder (Mbytes):	—	59	90	90	90

Acknowledgements. This paper is based on research funded by the Polish Ministry of Science and Higher Education (Grant No. N N516 480540).

References

1. Koehn, P., Hoang, H., Birch, A., Callison-Burch, C., Federico, M., Bertoldi, N., Cowan, B., Shen, W., Moran, C., Zens, R., Dyer, C., Bojar, O., Constantin, A., Herbst, E.: Moses: Open Source Toolkit for Statistical Machine Translation. In: Annual Meeting of the Association for Computational Linguistics (ACL). The Association for Computer Linguistics, Prague (2007)
2. Junczys-Dowmunt, M.: A Phrase Table without Phrases: Rank Encoding for Better Phrase Table Compression. In: Proc. of the 16th Annual Conference of the European Association for Machine Translation (EAMT), pp. 241–252 (2012)
3. Zens, R., Ney, H.: Efficient Phrase-table Representation for Machine Translation with Applications to Online MT and Speech Translation. In: Proc. of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (2007)
4. Levenberg, A., Callison-Burch, C., Osborne, M.: Stream-based Translation Models for Statistical Machine Translation. In: Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics, pp. 394–402 (2010)
5. Johnson, J.H., Martin, J., Fost, G., Kuhn, R.: Improving translation quality by discarding most of the phrasetable. In: Proc. of EMNLP-CoNLL 2007, pp. 967–975 (2007)
6. Talbot, D., Brants, T.: Randomized Language Models via Perfect Hash Functions. In: Proc. of ACL 2008: HLT, pp. 505–513. Association for Computational Linguistics, Columbus (2008)
7. Guthrie, D., Hepple, M., Liu, W.: Efficient Minimal Perfect Hash Language Models. In: Proc. of the 7th Language Resources and Evaluation Conference (2010)
8. Pouliquen, B., Mazenc, C.: COPPA, CLIR and TAPTA: three tools to assist in overcoming the language barrier at WIPO. In: MT-Summit 2011 (2011)
9. Belazzougui, D., Botelho, F.C., Dietzfelbinger, M.: Hash, Displace, and Compress. In: Fiat, A., Sanders, P. (eds.) ESA 2009. LNCS, vol. 5757, pp. 682–693. Springer, Heidelberg (2009)