A Phrase Table without Phrases: Rank Encoding for Better Phrase Table Compression

Marcin Junczys-Dowmunt
Faculty of Mathematics and Computer Science
Adam Mickiewicz University, Poznań, Poland
junczys@amu.edu.pl

Abstract

This paper describes the first steps towards a minimum-size phrase table implementation to be used for phrase-based statistical machine translation. The focus lies on the size reduction of target language data in a phrase table. Rank Encoding (R-Enc), a novel method for the compression of word-aligned target language in phrase tables is presented. Combined with Huffman coding a relative size reduction of 56 percent for target phrase words and alignment data is achieved when compared to bare Huffman coding without R-Enc. In the context of the complete phrase table the size reduction is 22 percent.

1 Introduction

As the size of available parallel corpora increases, the size of translation phrase tables used for statistical machine translation extracted from these corpora increases even faster. Although phrase table filtering methods (Johnson et al., 2007) have been described and physical memory as well as disk space are cheap, even current high-end systems can be pushed to their limits. The current in-memory representation of a phrase-table in Moses (Koehn et al., 2007), a widely used open-source statistical machine 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.[1]

The goal of this paper is to describe the first steps towards a compact phrase table implementation that can be used as a drop-in replacement for both, the binary phrase table implementation and the in-memory phrase table available in Moses. An important requirement is the faithful production of translations identical to translations generated from the original phrase table implementation if the same settings are provided.

The general idea is to trade in processor time for disk and memory space. Instead of keeping fully constructed target phrases in the phrase table, they are stored as Huffman compressed sequences of bytes. On demand, they are decompressed, decoded, and constructed as objects during run-time. As we show later, this does not necessarily mean that performance is negatively affected.

Even better compression can be achieved with a dedicated encoding method of target words developed for translation phrase tables. Rank Encoding (R-Enc) exploits the fact that target phrase words can be reduced to abstract symbols that describe properties of source phrase words rather than target words. The statistical distribution of these abstract symbols in the phrase table allows for a much better choice of Huffman codes.

2 Related Work

Zens and Ney (2007) 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 are substantial.

Promising alternatives to the concept of fixed phrase tables are suffix-array based implementation of phrase tables (Callison-burch and Bannard, 2005; Zhang and Vogel, 2005; Lopez, 2008; Levenberg et al., 2010) that can create phrase pairs on-demand more or less directly from a parallel corpus. However, we do not compare this approach with ours, as we are not concerned with on-demand phrase table creation.

[1] The need for a more compact phrase-table implementation arose during the author's collaboration with the MT team at WIPO. The space requirements of the binary representations of the phrase table and the reordering table for a single language pair exceeded the space available on a single SSD hard drive.

Other approaches, based on phrase table filtering (Johnson et al., 2007) 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. An important advantage of this approach is that can be easily combined with any fixed phrase table, including ours.

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 (Talbot and Brants, 2008; Guthrie et al., 2010). Guthrie et. al (2010) 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. The idea to use the CMPH[2] library (Belazzougui et al., 2009) and MurmurHash3[3] for our phrase table implementation originates from that paper.

The problem of parallel text compression has been addressed by only few works (Nevill-Manning and Bell, 1992; Conley and Klein, 2008; Sanchez-Martinez et al., 2012), most other works are earlier variants of Sanchez-Martinez et al. (2012). Conley and Klein (2008) propose to use an encoding scheme based on word alignment and source words. They require the existence of lemmatizers and other knowledge-heavy language related data. Also, compression results are reported without taking into account the additionally needed data. Conley and Klein claim to use phrase pairs for compression, but in our opinion their method is essentially word based, since pointers to all inflected words of a phrase need to be stored. The most recent work in the field is Sanchez-Martinez et al. (2012) who propose to use *generalized biwords* to compress running parallel data. A generalized biword consists of a source word, a sequence of target words aligned with the source word and a corresponding sequence of offsets. Their *Translation Relationship-based Encoder* (TRE) encodes a biword as a pair consisting of a source language word and a position information in a dictionary of generalized biwords. Rank-Encoding, though developed independently, is a combination of the methods presented by Conley and Klein and the TRE introduced by Sanchez-Martinez et al.

[2]http://cmph.sourceforge.net/

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

Phrase pairs:	3.36×10^8
Distinct source phrases:	2.15×10^8
Distinct target language words:	550,446
Distinct phrase scores:	1.36×10^7
Distinct alignment points:	49
Running source language words:	1.06×10^9
Running target language words:	1.62×10^9
Running phrase scores:	1.68×10^9
Running alignment points:	1.52×10^9
Total running target symbols:	4.84×10^9
Total running symbols:	5.89×10^9

Table 1: Coppa phrase table statistics

3 Experimental Data

The presegmented version of Coppa, the Corpus Of Parallel Patent Applications (Pouliquen and Mazenc, 2011), 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 Coppa phrase table that is used throughout this paper has been created using the standard training procedure of Moses with included word alignment information. Table 1 gives a set of figures for the phrase table.

The file size of the Moses binary phrase table is given in Table 3 (Section 5.3) along with the space and memory requirements of the variants of our phrase table implementation.

4 Compact phrase table implementation

Figure 1 illustrates the architecture of the discussed phrase table implementation. Its main modules are described in more detail in the following subsections.

4.1 Source Phrase Index

The structure of the source phrase index is inspired by Guthrie et al. (2010) 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 CMPH library (Belazzougui et al., 2009). The CHD algorithm generates very small MPH (in this case 109 Mbytes) in linear time.

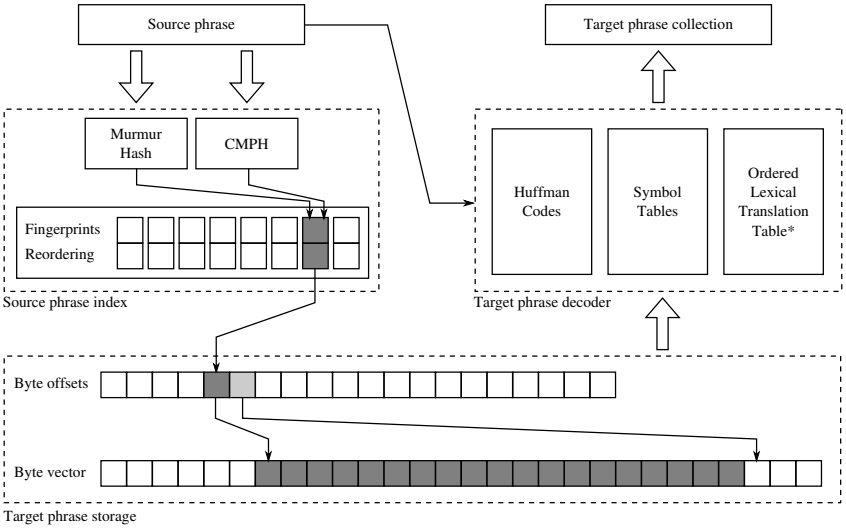

Figure 1: Simplified phrase table implementation schema

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. Guthrie et. al (2010) propose to use a random hash algorithm (MurmurHash3) 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. During our experiments it never happened for such false assignments to surface to a translation.

The MPH generated by the CHD algorithm is not order-preserving, hence the original position of the source phrase in an ordered set S is stored together with each fingerprint. Order-preservation is crucial if any kind of disk IO is involved. In Moses, source phrases are queried by moving the start point of a phrase to each word of a sentence and increasing the phrase length until the length limit or the end of the sentence is reached. Therefore for each start word the querying order is a lexicographical order. In a phrase table representation that preserves the lexicographical source phrase order for corresponding target phrase collections, the results for lexicographically ordered queries will 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.

This phrase table does not contain any representation of source phrases besides the MPH function. Source phrases can be checked for inclusion, but not recovered.

4.2 Target Phrase Storage

The target phrase storage 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, again stored consecutively. 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.

Size reduction is achieved by compressing the symbol sequence of a target phrase collection using symbol-wise Huffman coding (Huffman, 1952; Moffat, 1989). Target phrase words, scores, and alignment points are encoded with different sets of Huffman codes which are switched during coding and decoding.

a bacillus strain | une souche de bacille | 0-0 2-1 1-3

(a) Example phrase pair with alignment

Source	Target	Rank
a	un	0
a	une	1
a	de	2
a	la	3
bacillus	bacillus	0
bacillus	bacille	1
bacillus	bacilles	2
strain	souche	0
strain	contrainte	1
strain	déformation	2
of	de	0
of	d'	1
of	du	2

(b) Bilingual dictionary

Step	Result	Alignment
0.	une souche de bacille	0-0 2-1 1-3
1.	une[0] souche[2] de bacille[1]	∅
2.	une[0,1] souche[2,0] de bacille[1,1]	∅
3.	[0,1] [2,0] de [1,1]	∅
4.	[1] [2,0] de [1,1]	∅

(c) Target phrase encoding procedure

Step	Result	Alignment
0.	[1] [2,0] de [1,1]	∅
1.	[0,1] [2,0] de [1,1]	∅
2.	(a)[1] (strain)[0] de (bacillus)[1]	0-0 2-1 1-3
3.	une souche de bacille	0-0 2-1 1-3

(d) Target phrase decoding procedure

Figure 2: An encoding/decoding example

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[4] differences between the offsets are stored. A synchronization point with the full offset value is inserted and tracked for every 32 values.[5]

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 (Anh and Moffat, 2004) for offset differences and Variable Byte Length encoding (Scholer et al., 2002) for synchronization points. As both techniques use less space if smaller numbers are compressed, the size of the structure keeps decreasing with decreasing offset differences. Therefore a better compression method for the byte array results automatically in a smaller byte offset vector. For the baseline phrase table, the roughly 215 million offsets use 260 Mbytes, but only 220 Mbytes for the rank-encoded variant.

4.3 The Phrase Decoder

The target phrase decoder contains the data that is needed to decode the compressed byte streams. It includes source and target word lists with indexes, the Huffman code sets, and if Rank Encoding is used, a sorted lexical translation table. The word lists and the translation table do not account for more than 30 Mbytes. Huffman codes are stored as canonical Huffman codes, a memory efficient representation. The size of the target phrase decoder is treated as part of the size required to represent target phrases.

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

[5] This an arbitrarily set step size.

4.4 Baseline implementation

In the baseline implementation three different sets of Huffman codes are used to encode target words, scores, and alignments; encoding and decoding relies on switching between the three types of Huffman codes. For target phrase words and alignment points one special stop symbol has to be added. Scores and alignment points are encoded directly, target phrase words have an intermediate representation as integer identifiers which are looked up in a target word table. This implementation is referred to as "Baseline". See Table 3 for the size characteristics of this implementation.

5 Rank Encoding

In this section Rank Encoding (R-Enc), a method for the compression of parallel texts, is presented. Strictly speaking, it is not a compression method by itself, but prepares the data in such a way that traditional compression is more efficient.

5.1 Outline of the Method

The main idea is to modify the probability distribution of symbols in the target data in such a way that the average length of the Huffman codes decreases. Bilingual data (a phrase table is nothing

else) has a property that helps with this problem.

Given a source phrase, a target phrase, and a bilingual dictionary of source and target words, it can be told for most target words which source words they are translations of. This information can be encoded into the target phrase and the surface forms of the target words can be dropped.

Figure 2 illustrates the procedure in more detail for an example phrase pair (2a) consisting of a source phrase, a target phrase, and alignment data. The available alignment information simplifies the process of finding correspondences between source and target words. Also a sample bilingual dictionary (2b) is included. The encoding procedure (2c) can be performed in four steps:

1. Alignments are moved to target words.
2. The source word is looked up in the dictionary and the position (rank) of the target word among the translations is recorded.
3. Aligned target words are dropped.
4. If positions of target and source word are equal, the alignment is dropped.

The target phrase consists now of three different types of symbols: ranks, ranks with alignment information, and target words.

The decoding procedure (2d) is as simple:

1. Alignment information is added to rank-only symbols based on their position in the pattern.
2. The original alignment data is reconstructed and source words are determined by their position in the source phrase.
3. Based on source words and ranks of their translation the target words are re-inserted.

In this example the alignment been reduced to the empty set, as it happens in most cases.

The counterparts of a source word in the dictionary are ordered by their decreasing translation probability $p(t|s)$. The lexical translation table generated during Moses training can be used for this. The lower the rank of a translation the higher is its probability. Symbols with low ranks are therefore more likely to occur, for a probability-based compression algorithm as Huffman coding this is highly desirable.

The described encoding scheme removes the actual target phrases from the phrase table, similarly as the MPH in the source index removes source phrases, hence, the title of the paper. Source and target phrases can only be recovered during querying. Compression is thus achieved by moving information to the query.

5.2 Formal Description of Algorithms

Two functions for dictionary querying are defined: the rank $r(s,t)$ of a target word t relative to a source word s is the position of t within the list of translations of s. Conversely, given a source word s and a rank r the target word $t = t(s,r)$ is obtained from the lexical table.

For each of the three symbols types an encoding function is defined. Plain target words are encoded with e_1, symbols that contain implicit alignments with e_2, and e_3 encodes pairs of source position and target rank. The inverse functions e_1^{-1}, e_2^{-1}, and e_3^{-1} decode numerical values to symbols.

The codomains of e_1, e_2, and e_3 are required to be pair-wise distinct. Then the decision function d can determine the type of a given symbol based on its encoded numerical value. Based on that, the correct decoding function is applied.[6]

Algorithm 1 formalizes the encoding procedure. First, source word positions are partitioned into n sets J_i of positions of source words aligned with t_i. That way, worst-time complexity is $O(mn)$ if the given alignment contains all possible $m \times n$ alignments points. Average time complexity is $O(n)$ for alignments with about n alignment points.[7]

The algorithm processes the target phrase $\mathbf{t}$ of length n from left to right. For each target word t_i all source words that are aligned with t_i are examined. If t_i is not aligned with any source word, it is encoded as a plain symbol of type 1.

For an aligned target word t_i, the minimal rank r of t_i relative to any of these source words is determined. If for the minimal rank there is more than one source word aligned with t_i, the left-most source word position k is chosen. This two-fold selection of minimal values is crucial for the size-reduction effect of the later compression. Lower values of rank and position appear more often and are assigned shorter Huffman codes. If $k = i$, i.e. source and target word occupy the same position,

[6] The implementation of the decision, encoding, decoding functions relies on bitwise operations on integer values, but in fact any representation can be used if it fulfills the previous requirements.

[7] According to the statistics for the Coppa phrase table there are actually less alignment points than target words.

Function EncodePhrase($\mathbf{s}, \mathbf{t}, A$)
begin
  $\hat{\mathbf{t}} \leftarrow \langle\,\rangle$
  $\hat{A} \leftarrow A$
  foreach $\langle j, i\rangle \in A$ **do**
    $J_i \leftarrow J_i \cup \{j\}$
  end
  foreach $i \in \{1, \ldots, |\mathbf{t}|\}$ **do**
    if $J_i = \emptyset$ **then**
      $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{t}} \cdot \langle e_1(t_i)\rangle$
    else
      $r \leftarrow \min\{r(s_j, t_i) : j \in J_i\}$
      $k \leftarrow \min\{j : j \in J_i \wedge r(s_j, t_i) = r\}$
      if $k = i$ **then**
        $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{t}} \cdot \langle e_2(t_i)\rangle$
      else
        $\hat{\mathbf{t}} \leftarrow \hat{\mathbf{t}} \cdot \langle e_3(t_i)\rangle$
      end
      $\hat{A} \leftarrow \hat{A} \setminus \{\langle k, i\rangle\}$
    end
  end
  return $\langle \hat{\mathbf{t}}, \hat{A}\rangle$
end

Algorithm 1: Rank encoding

only the rank r is encoded (symbol type 2). Otherwise, k and r are encoded together as one symbol (symbol type 3). Alignment points used during encoding are dropped from the input alignment. Only the unused alignment points in $\hat{A}$ are saved in the alignment of the encoded target phrase.

Decoding (algorithm 2) is straightforward. The encoded target phrase pattern $\hat{\mathbf{t}}$ is processed from left to right. Each symbol is decoded using the appropriate decoding function based on the symbol type. For symbols of type 1 no alignment point is restored. Symbols of type 2 are decoded to a rank value and looked up using the source phrase word that is located at same position i as the current target phrase word t_i, an alignment point $\langle i, i\rangle$ is restored. For symbols of type 3, the source word position j is recovered from the symbol and the target word is looked-up, a point $\langle j, i\rangle$ is added to the alignment. Average and worst-case time complexity is equal to $O(n)$. Phrase tables without explicit alignment data can be encoded by providing a Cartesian product of source and target word positions instead. Average complexity for encoding is then equal to the worst case complexity $O(nm)$. Decoding time complexity is unchanged.

Function DecodePhrase($\mathbf{s}, \hat{\mathbf{t}}, \hat{A}$)
begin
  $\mathbf{t} \leftarrow \langle\,\rangle$
  $A \leftarrow \hat{A}$
  foreach $i \in \{1, \ldots, |\hat{\mathbf{t}}|\}$ **do**
    switch $d(\hat{t}_i)$ **do**
      case 1
        $\mathbf{t} \leftarrow \mathbf{t} \cdot \langle e_1^{-1}(\hat{t}_i)\rangle$
      case 2
        $r \leftarrow e_2^{-1}(\hat{t}_i)$
        $\mathbf{t} \leftarrow \mathbf{t} \cdot \langle t(s_i, r)\rangle$
        $A \leftarrow A \cup \{\langle i, i\rangle\}$
      case 3
        $\langle j, r\rangle \leftarrow e_3^{-1}(\hat{t}_i)$
        $\mathbf{t} \leftarrow \mathbf{t} \cdot \langle t(s_j, r)\rangle$
        $A \leftarrow A \cup \{\langle j, i\rangle\}$
    endsw
  end
  return $\langle \mathbf{t}, A\rangle$
end

Algorithm 2: Rank decoding

5.3 Results

Table 2 summarizes the results for Rank Encoding applied to target phrases of the Coppa phrase table. Here, only figures for target words and alignment points are compiled as we want to evaluate the performance of Rank Encoding alone.

Rank Encoding reduces the number of distinct target words from 550,446 to 86,367. In the baseline phrase table the first 100 most frequent symbols account for 52 percent of the running target words, but for 91 percent if Rank Encoding is used. These different distributions of symbols and frequencies affect the later applied Huffman coding significantly. The number of bits per running target words decreases from 10.8 to 6.5. The size reduction is even more substantial for alignment points as the number of bits per running alignment point drops from 5.4 to 0.5. This is the effect of the majority of alignment points being encoded into target words symbols. Of ca. 1.5 billion alignment points only 55 million (3.7 percent) are compressed explicitly. Bit numbers include the overhead introduced by stop symbols, for R-Enc the bilingual dictionary is added as well. The total size of target phrases with alignments is reduced by 56 percent from 3,096 Mbytes to 1,351 Mbytes. In the context of the complete phrase table (in Table 3) the size reduction is 22 percent.

	Baseline	R-Enc
Distinct target words:	550,446	86,367
Bytes per target phrase (without scores):	9.7	4.2
Bits per target word:	10.8	6.5
Bits per alignment point:	5.4	0.5
Bits per symbol (words & alignment):	8.2	3.6
Total space (Mbytes):	3,096	1,351

Table 2: Results for rank-encoded target words and alignments

	Moses	Baseline	R-Enc
Total size in Mbytes (ordered) :	29,418	7,681	5,967
Ordered source phrase index (Mbytes):	5,953	1,750	
Target phrase storage (Mbytes):	23,441	5,873	4,127
Target phrase decoder (Mbytes):	—	59	90
Bytes per target phrase:	73.1	18.5	13.2
Bits per symbol (words & score & alignment):	40.6	10.3	7.2
Translation time (1st run):	1606 s	1322 s	1450 s
Translation time (2nd run):	1051 s	940 s	957 s
Memory usage peak:	1.6 G	2.7 G	2.8 G

Table 3: Comparison of phrase table implementations

We measured the speed of our phrase table variants and the Moses phrase table on the first unique thousand sentence pairs from test set provided by WIPO[8]. Two scenarios are considered: During the "1st run" operation system IO caches are dropped before translation. During the "2nd run" the translation process is started with the same parameters, but IO caches of the previous run are available. Caching as provided by Moses is enabled.

Concerning speed, our phrase table implementations outperforms the Moses binary phrase table. The difference is more noticeable for first runs. One has to keep in mind, that the search for translation options occupies only a small percentage of time during the translation. The decoding process itself is much more time consuming. Improved performance for first runs can be explained by greatly reduced disk access which levels out increased processing requirements due to decompression. Speed is more similar for second runs, where all phrase table variants can take advantage of the IO caching mechanism of the operation system. The Moses phrase table fares well when peak memory consumption is compared. The Baseline and the rank encoded variant use over 1 GB more memory than the binary Moses phrase table. This due to the source phrase index which at the moment is fully kept in memory. [9]

6 Conclusions and Future Work

Rank Encoding if combined with Huffman coding reduces the size of a phrase tables substantially when compared to bare Huffman coding. Translation speed is faster or comparable to the binary phrase table in Moses. Memory requirements are currently higher. In the presented phrase table implementation compression has been achieved by removing actual representation of source and target phrases from the phrase table. Both can only be recovered when the phrase table is being queried with potential source phrases.

There is still much potential for further size reductions. In this work the focus lay mainly on tar-

[8]http://www.wipo.int/patentscope/translate/coppa/testset2011.tmx.tgz

[9]After submission of this paper, we managed to reduce the space requirements of the index and to implement a lazy loading procedure. Instead of 1.7 GBytes only 300 MBytes are consumed for the translation of the test set. The methods used to achieve this reduction will be described in a forthcoming paper.

get words and alignment information. Now scores take up a majority of space in a target phrase and can surely be reduced by mathematically grounded smoothing methods without a noticeable impact on translation quality. Also, since a translation table is already used for R-Enc on-line calculation of lexical probabilities can be considered an option. The source phrase index needs to be optimized. Other order preserving or monotonous hash functions or indexing methods are to be reviewed and tested. The impact of fingerprint bit length on translation quality should be examined.

Concerning R-Enc, other similar encoding techniques need to be investigated, especially extending the described approach to full bilingual phrase pairs instead of word pairs. Due to the highly repetitive nature of phrase tables this might be a promising course of research.

Acknowledgments

This research is funded by the Polish Ministry of Science and Higher Education (grant no. N N516 480540). The idea for this work was conceived during a stay of the author's at WIPO in Geneva, while working on the in-house MT system.

References

Anh, Vo Ngoc and Alistair Moffat. 2004. Index compression using fixed binary codewords. In *Proceedings of the 15th Australasian database conference*, pages 61–67.

Belazzougui, Djamal, Fabiano C. Botelho, and Martin Dietzfelbinger. 2009. Hash, displace, and compress. In *Proceedings of the 17th European Symposium on Algorithms*. Springer LNCS.

Callison-burch, Chris and Colin Bannard. 2005. Scaling phrase-based statistical machine translation to larger corpora and longer phrases. In *In Proceedings of ACL*, pages 255–262.

Conley, Ehud S. and Shmuel T. Klein. 2008. Using Alignment for Multilingual Text Compression. *Int. J. Found. Comput. Sci.*, 19(1):89–101.

Guthrie, David, Mark Hepple, and Wei Liu. 2010. Efficient Minimal Perfect Hash Language Models. In *Proceedings of the Seventh Language Resources and Evaluation Conference*.

Huffman, David. 1952. A Method for the Construction of Minimum-Redundancy Codes. *Proceedings of the IRE*, 40(9):1098–1101.

Johnson, J. Howard, Joel Martin, George Fost, and Roland Kuhn. 2007. Improving translation quality by discarding most of the phrasetable. In *In Proceedings of EMNLP-CoNLL07*, pages 967–975.

Koehn, Philipp, Hieu Hoang, Alexandra Birch, Chris Callison-Burch, Marcello Federico, Nicola Bertoldi, Brooke Cowan, Wade Shen, Christine Moran, Richard Zens, Chris Dyer, Ondrej Bojar, Alexandra Constantin, and Evan Herbst. 2007. Moses: Open source toolkit for statistical machine translation. In *Annual Meeting of the ACL, Prague*.

Levenberg, Abby, Chris Callison-Burch, and Miles Osborne. 2010. Stream-based translation models for statistical machine translation. In *Proceedings of NAACL-HLT 2010, Los Angeles*, pages 394–402.

Lopez, Adam. 2008. Tera-scale translation models via pattern matching. In *Proceedings of the 22nd International Conference on Computational Linguistics*, COLING '08, pages 505–512, Stroudsburg, PA, USA. Association for Computational Linguistics.

Moffat, A. 1989. Word-based Text Compression. *Softw. Pract. Exper.*, 19(2):185–198.

Nevill-Manning, Craig G. and Timothy C. Bell. 1992. Compression of Parallel Texts. *Inf. Process. Manage.*, 28(6):781–794.

Pouliquen, Bruno and Christophe Mazenc. 2011. COPPA, CLIR and TAPTA: three tools to assist in overcoming the language barrier at WIPO. In *MT-Summit 2011*.

Sanchez-Martinez, Felipe, Rafael C. Carrasco, Miguel A. Martinez-Prieto, and Joaquin Adiego. 2012. Generalized Biwords for Bitext Compression and Translation Spotting. *Journal of Artificial Intelligence Research*, 43:389–418.

Scholer, Falk, Hugh E. Williams, John Yiannis, and Justin Zobel. 2002. Compression of inverted indexes for fast query evaluation. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '02, pages 222–229.

Talbot, David and Thorsten Brants. 2008. Randomized Language Models via Perfect Hash Functions. In *Proceedings of ACL-08: HLT*, pages 505–513, Columbus, Ohio, June. Association for Computational Linguistics.

Zens, Richard and Hermann Ney. 2007. Efficient Phrase-table Representation for Machine Translation with Applications to Online MT and Speech Translation. In *Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics (HLT-NAACL), Rochester, NY.*

Zhang, Ying and Stephan Vogel. 2005. An efficient phrase-to-phrase alignment model for arbitrarily long phrase and large corpora. In *Proceedings of the 10th Conference of the European Association for Machine Translation (EAMT-05*, pages 30–31.