International Journal of Foundations of Computer Science

USING ALIGNMENT FOR MULTILINGUAL TEXT COMPRESSION

EHUD S. CONLEY
Department of Computer Science, Bar-Ilan University
52100 Ramat-Gan, Israel
konli@cs.biu.ac.il

and

SHMUEL T. KLEIN
Department of Computer Science, Bar-Ilan University
52100 Ramat-Gan, Israel
tomi@cs.biu.ac.il

Received (received date)
Revised (revised date)
Communicated by Editor's name

ABSTRACT

Multilingual text compression exploits the existence of the same text in several languages to compress the second and subsequent copies by reference to the first. We explore the details of this framework and present experimental results for parallel English and French texts.

Keywords: Compression, multilingual texts, text alignment, coding

1. Introduction

In countries like Canada, Belgium and Switzerland, where speakers of two or more languages live side-by-side, all official texts have to be published in multilingual form. The current legislation of the ever expanding European Union obliges the translation of all official texts into the languages of all member states. As a result, there is a growing corpus of important texts, large parts of which are highly redundant, since they do not have any information content of their own, and are just transformed copies of some other parts of the text collection.

We wish to exploit this redundancy to improve compression efficiency in such situations, and introduce the notion of *Multilingual Text Compression*: one is given two or more texts, which are supposed to be translations of each other and are referred to as *parallel* texts. One of the texts will be stored on its own (or compressed by means of pointers referencing only the text itself), the other texts can be compressed by referring to the translation, using appropriate dictionaries.

Data compression in general, and text compression in particular, have for long been prominent topics in the Information Retrieval literature, as full text IR systems are voracious consumers of storage space, both for the underlying textual database itself, but also for the auxiliary overhead, such as indices, dictionaries, thesauri, etc., see, for example, [14, 20, 13]. This work concentrates on multilingual information retrieval systems and how their data could be compressed.

In a certain sense, multilingual text compression is an extension of *delta-coding*, in which source and target files S and T are given, with the assumption that T is very similar to S, for example in the case of several versions of the same software package. Highly efficient compression schemes have been designed for that case, and the compressibility is obviously a function of the similarity of the input files. Our problem extends the delta-coding paradigm to the case where similarity is not based on the appearance of identical strings, but allow the use of some transformation to pass from a given text fragment to its matching part.

The basis for enabling multilingual text compression is first the ability to match the corresponding parts of related texts by identifying semantic correspondences across the various sub-texts, a task generally referred to as *alignment*. As the methods for detailed alignment are quite sensitive to noise, they usually use a rough alignment of the text as an auxiliary input. They might also use an existing multilingual glossary, but they always generate their own probabilistic glossary, which corresponds to the processed text.

The current work extends the use of alignment to the question of whether and how the property of parallelism can be exploited to store those texts in a more space-efficient way. In other words, we wish to find a way to compress the constituent parallel sub-texts so that the result will demand less space than would be required if they were compressed without exploiting their parallelism.

In the next section, we review some related work. Section 3 brings the suggested algorithm and reports on preliminary experimental results. The last section suggests future work.

2. Related work

Multilingual texts have been considered in the Information Retrieval literature, where the challenge is to access information in one language while the query might be given in another, see, e.g. [5]. Alignment of parallel texts has been used mainly for machine translation, machine-aided translation and bilingual term extraction [17]. Most algorithms for alignment are designed for bilingual texts only [9], but some work has been done already for three languages as well [16]. However, the state of the art for detailed alignment, even for two languages, is still far from perfect. It is thus not surprising that works on more than two languages do not exist, but a reasonable mapping for (A, B, C) can be synthesized given alignment outputs for (A, B) and (A, C).

Most current detailed alignment techniques are based on one of the following models: (a) IBM's Model 2 [3], from which the *word_align* algorithm [7] has been derived; and (b) Hiemstra's model [12], used both by Xerox' system [10] and the

Linköping Word Aligner [1].

All these methods use some monolingual tools such as part-of-speech taggers, lemmatizers and possibly parsers for phrase detection. Determining the lemma (= base form) of each word is critical for the success of the alignment process, especially when performed across languages from different groups [6]. When the lemmatized versions of the texts are processed instead of the original versions, the words within the induced bilingual glossary will naturally be all lemmata rather than morphological variants.

The compression of similar texts has been considered in the vast research area dealing with delta coding, see [4, 2]. The popular ZLIB tool is optimized to take advantage of the similarity across the files, and some of its features are used also in our algorithm. The compression of parallel texts is treated in [15], but without using text alignment tools.

3. Compression of a text using its translation

3.1. Compression modeling

The following compression algorithm tries to take advantage of the fact that the text being compressed is divisible into two parallel parts which are translations of each other. Dictionary based compression algorithms use pointers to occurrences of the same substrings either along the text, as in LZ77 [21] or within an auxiliary dictionary, as in LZW [19]. The current algorithm, however, uses pointers to the translations of the substring appearing in the parallel section of the text. The original substrings may be easily retrieved through these pointers using a bilingual glossary along with some other linguistic resources.

Pointing to another occurrence of a given substring within the same text sometimes requires a relatively large number of bits. That is because the closest occurrence of that substring can happen far back in history, which is why most implementations limit the size of the window in which a previous occurrence is to be searched for. In contrast, translations of words or phrases within a parallel text, if such exist, must appear in the corresponding translation unit, namely a sentence or paragraph. Moreover, if no large omissions or insertions occur, the translation is expected to be found within a very narrow text window, whose middle position is computable using the given alignment. The encoding pointers can store the offset of the translation from that alignment; these offsets are always very small and thus may be encoded using only a few bits.

It is important to emphasize that the quality of the alignment does not have any effect on the correctness of the compression algorithm. That is because the missing words or word sequences are restored according to the same glossary by which the alignment has been determined. It is expected that the compression rate would not be affected either, as alignment algorithms make mistakes due to the consistent appearance of the wrong translations in the corresponding text windows, even in more probable positions. This means that the same sequence can be compressed at

least the same number of times using the erroneous translation and perhaps even at a better cost.

The suggested algorithm assumes the following resources:

1. S, T: The source- and target-language texts, respectively, where T is a translation of S.

2. $A_{S,T}$: A word- and phrase-level alignment of the text pair (S,T). Let $s_{i,l}$ denote the word sequence of length l within S beginning at the ith word. Similarly, let $t_{j,m}$ denote the word sequence of length m within T beginning at the jth word. $A_{S,T}$ consists of a set of connections of the form $\langle i,l,j,m\rangle$, each of which indicating the fact that $s_{i,l}$ and $t_{j,m}$ have been determined as matching phrases. We assume that for any pair (j,m) there is at most one connection of the form $\langle i,l,j,m\rangle$ within $A_{S,T}$. From here and below, s_i and t_j stand for $s_{i,1}$ (the ith word of S) and $t_{j,1}$ (the jth word of T), correspondingly.

3. S^{lem}, T^{lem}: Lemmatized forms of S and T. Let $s^{lem}_{i,l}$ and $t^{lem}_{j,m}$ denote the lemma sequences corresponding to $s_{i,l}$ and $t_{j,m}$, respectively. That is the concatenations of the lemmata of s_i, s_{i+1}, $\ldots$, s_{i+l-1} and t_j, t_{j+1}, $\ldots$, t_{j+l-1}, correspondingly.

4. L_S: A lemmata dictionary. The entries of this dictionary are the words appearing in S. Each entry stores a list of all possible lemmata of the keyword, sorted in descending order of frequency. Let $L_S(s)$ denote the lemma list for the word s. For instance, if S is an English text, then $L_S(\texttt{working}) = (\texttt{work}, \texttt{working})$.

5. V_T: A variant dictionary. The entries of this dictionary are the lemmata of all words appearing in T. Each entry stores a list of all possible morphological variants of the key lemma, sorted in descending order of frequency. Let $V_T(t)$ denote the variant list for the lemma t. For example, if T is a French text, then $V_T(\texttt{normal}) = (\texttt{normal}, \texttt{normale}, \texttt{normaux}, \texttt{normales})$.

6. $G_{S,T}$: A bilingual glossary corresponding to the text pair (S,T). The entries of this glossary are source language lemma sequences. Each entry includes a list of possible translations of the key sequence into target language sequences, sorted in descending order of probability. The translations also appear in lemmatized form. Let $G_{S,T}(s)$ denote the translation list of the source language sequence s into the target language. For instance, if S and T are English and French texts, correspondingly, then $G_{S,T}(\texttt{mineral water}) = (\texttt{eau mineral})$. Note that the word `eau` (water) in French is feminine, which requires a feminine-form adjective, namely `minerale`, whereas the adjective `mineral` is the masculine singular form, which is the corresponding lemma.

Let $al(j)$ denote the expected position within S of the term corresponding to t_j in T, that is,

$$al(j) = \left\lfloor \frac{|S|}{|T|} j + \frac{1}{2} \right\rfloor .$$

COMPRESS_TARGET

```
j ⟵ 1
while j ≤ |T| do
        found ⟵ false
        for m ⟵ m_max downto 1 do
                if ∃i, l such that ⟨i, l, j, m⟩ ∈ A_{S,T}    //⟨i, l, j, m⟩ is unique
                        diff ⟵ i − al(j)
                        if diff ≥ 0          then   sign ⟵ 0
                                             else   sign ⟵ 1
                        offset ⟵ B(|diff|)
                        length ⟵ B(l − 1)
                        for n ⟵ 0 to l − 1 do
                                lemma_n ⟵ I(s^lem_{i+n}, L_S(s_{i+n}))
                        trans ⟵ I(t^lem_{j,m}, G_{S,T}(s^lem_{i,l}))
                        for n ⟵ 0 to m − 1 do
                                variant_n ⟵ I(t_{j+n}, V_T(t^lem_{j+n}))
                        pointer ⟵ concatenation ( 1, offset, sign, length,
                                                  lemma_0, ..., lemma_{l−1}, trans,
                                                  variant_0, ..., variant_{m−1})
                        output pointer
                        j ⟵ j + m
                        found ⟵ true
                        break
                endif
        end for
        if not found
                output concatenation ( 0, code(t_j))
                j ⟵ j + 1
        endif
end while
```

FIGURE 1: *Compression using a translated file*

In other words, $s_{al(j)}$ is the source word parallel to t_j if taking into account only the proportion between the lengths of S and T. The accurate alignment may then be expressed by the signed offset from $s_{al(j)}$. If a paragraph- or sentence-level alignment is available, then S and T can be referred to as the current parallel units, and the indices i and j are then relative to the beginnings of these units.

Token number	S (English)	T (French)	Encoding
1	`Subject`	`Objet`	1(0,ϵ,0,ϵ,6,0)
2	`:`	`:`	0(c(:))
3	`Supplies`	`Livraisons`	0(c(`livraison`),2)
4	`of`	`de`	1(2,0,0,ϵ,1,0,0)
5	`military`	`matériel`	
6	`equipment`	`militaire`	1(0,ϵ,0,ϵ,0,1)
7	`to`	`à`	0(c(`à`),0)
8	`Iraq`	`l'`	0(c(`le`),2)
9		`Irak`	1(0,ϵ,0,ϵ,0,ϵ)

FIGURE 2: *Example of compression of French text using its English parallel*

The algorithm works as follows: beginning at the first position $j = 1$ within T, use $A_{S,T}$ to find the longest sequence $t_{j,m}$ having a corresponding sequence $s_{i,l}$ in S. If found, create a pointer to $s_{i,l}$ by concatenating some binary encodings of the following details:

1. $i - al(j)$: Offset of s_i from $al(j)$, including sign bit.
2. $l - 1$: Length of the source sequence minus 1. As l is always greater than 0, $l - 1$ can be encoded.
3. Indices of $s_i^{lem} \dots s_{i+l-1}^{lem}$ within $L_S(s_i) \dots L_S(s_{i+l-1})$, respectively. If a single lemma exists, then the empty string ϵ is used as index (no need for encoding).
4. Index of $t_{j,m}^{lem}$ within $G_{S,T}(s_{i,l}^{lem})$. As above, in the case of a single translation, ϵ will be used.
5. Indices of $t_j \dots t_{j+m-1}$ within $V_T(t_j^{lem}) \dots V_T(t_{j+m-1})$, correspondingly. Again, ϵ is used in the case of singletons.

The pointer is then output with a 1-bit prefix. The next iteration will work for $j = j + m$.

If no m is found such that $\langle i, l, j, m \rangle \in A_{S,T}$, an alternative encoding of t_j is written to the output stream preceded by a 0-bit, and j is incremented by 1. The process continues while $j \leq |T|$. We shall use some UD (Uniquely Decypherable) code, e.g., a Huffman code, for all unaligned words in T. This code may be initially generated for all words in T and then be improved when the counts of unaligned

words are known. Alternatively, the final code can be generated in advance following a preliminary parsing stage.

As to the encoding of the pointer consisting of a sequence of generally very small numbers, many of which are zeros, a simple solution would be to use an Elias γ-code for each component. A more compact encoding can be achieved by devising a Huffman code for the possible numbers, see the section on coding below.

Figure 1 displays the formal pseudo-code. $B(x)$ denotes the variable length binary encoding of x and $I(x, y)$ denotes the variable length binary encoding of the index of x within the dictionary entry y; if y contains only one item, $I(x, y) = \epsilon$.

The decompression algorithm is straightforward. Note that it needs only the dictionary files, as all relevant information included in the other files is encoded within the compressed text itself.

Figure 2 gives an example of the algorithm's output. The second and third columns contain the English and French parallel texts, respectively. The fourth column is a decimal representation of the binary encoding. The 0 to the left of parentheses denotes the encoding of unaligned words, while a 1 indicates a pointer. Numeric values within the parentheses are actually written to the binary output as variable length binary numbers, for example, if a γ-code is used, the 6-tuple $(2, 0, 0, \epsilon, 1, 0, 0)$ would be encoded as 1100|0|0||10|0|0 (10 bits).

As an example, we explain in detail the decoding of the fourth encoded token, which is $(2, 0, 0, \epsilon, 1, 0, 0)$, assuming that the first three items have already been decoded to `Objet : Livraison`. The current position (in terms of tokens) in the file T is therefore 4, and in S, it is $\lfloor (8/9) \times 4 + \frac{1}{2} \rfloor = 4$, corresponding to the word `of`. The first two numbers of the 6-tuple are retrieved: 2, 0 are translated to +2, indicating the fact that the translation sequence is located two words to the right of the current position in S, which brings us to the term `equipment`. Adding 1 to the next value, 0, tells the decoder that it should relate to a 1-word English sequence beginning (and ending) at the word `equipment`. Taking a look at the entry `equipment` in the English lemmata dictionary (L_{en}(`equipment`)) reveals there is only one lemma for that word (the lemma `equipment`). Therefore, no bits are needed in order to lemmatize it.

Now the decoder looks up the entry `equipment` within the bilingual glossary ($G_{en,fr}$(`equipment`)) and finds the list `le équipement, de matériel, équipement`. Since several French translations exist, it reads the next value, 1, and retrieves the corresponding translation (the second option), namely `de matériel`, so the translation sequence is of length 2. Since both words in this sequence have more than one variant, another two values are fetched in order to determine the exact form of each. The variants list of the lemma `de` starts with `de`, `des`, `d'`, `du`... and that of `matériel` starts with `matériel`, `matériaux`, `matériels`, `matérielles`, `matérielle`.... The two last zeros in the sequence to be decoded indicate that the first variant of each list should be taken, yielding finally the terms (not the lemmata) `de matériel` as translation for `equipment`.

Note that this translation, if considered on its own and not within the larger context of a bilingual corpus, is in fact quite wrong, since `de matériel` is a genitive

form rather corresponding to `of equipment`. This is an example for the fact that an erroneous translation can still be useful in our case, if the error appears consistently.

3.2. Choosing the encoding

To understand the rational of the encoding decisions, consider Figure 3, listing the first few output lines of the above algorithm applied to our test data.

```
1    0 0 0 0
0    :
0    organigramme 1
0    de 0
1    11 0 0 1 1 0
1    5 0 0 3 5 0
1    1 0 0 3 0
0    elle 0
0    :
1    9 0 0 0
0    )
1    3 0 0 2 0
1    5 0 0 5 4 0 2
0    agent 0
1    7 0 0 1
1    10 0 0 0 3 0
0    dans 0
```

FIGURE 3: *Output of translation algorithm*

The first column is a flag indicating whether the element is a pointer or one of the non-aligned words. If it is a word, it may be followed by a number, giving the index of the requested variant in the list of alternatives for this lemma. If it is a pointer, it starts with a number k, representing an offset, in number of words, between some term positions as explained above. If k is not zero, it is followed by a sign bit, encoded here by 0 or 1. The rest of the numbers in the pointers are indices within sets of variants.

The encoding tries to take advantage of the fact that the distribution of the elements in the different fields is not the same. In fact, three Huffman codes are used:

1. H_1 — for the different words in the lines labeled 0;
2. H_2 — for the offsets (first numbers in lines labeled 1);
3. H_3 — for all the indices appearing in both types of lines, words and pointers.

The first tree H_1 is quite large, giving a codeword for each of the different non-aligned words. As to H_2, most of the offsets are small, and their distribution is skewed, with a clear bias to the smaller numbers. The numbers encoded by H_3

are usually even smaller, since for most sets, there are generally very few variants. Moreover, since these variants are ordered by decreasing frequency, the first few integers, especially 0, will appear with high probability. The reason for not using the same Huffman tree for the last two classes, in spite of the fact that similar elements are encoded, is that their distributions are different enough to justify two trees, in particular because no ambiguity arises: there is only one element of H_2 for each pointer line, so no special indicator is needed for the fact that the next codeword is from H_3.

There is no need to encode the sign field by some Huffman code. Once we know that a pointer is encoded, the first codeword belongs to H_2, and if it is decoded as representing a number different from 0, we know that it is followed by a sign bit, so the Huffman codeword is just followed by the sign bit itself. On the other hand, the flag bit indicating if the current line is a word or a pointer, needs to be encoded. Instead of wasting one bit for each line, it turned out, on our tests, to be advantageous to adopt the following scheme: every new line is by default assumed to represent a word, and the Huffman tree H_1 is extended to accommodate also an "Escape" word, which will be used at the beginning of every pointer line.

The encoding of H_3 can further be improved by noticing that the probability of the number 0 will be higher than $\frac{1}{2}$, suggesting, as in [8], to build a Huffman code for a set of items consisting of (a) individual numbers appearing in the sequences and (b) of runs of zeros of different lengths. The elements to be encoded by H_3 are therefore $0, 1, 2, \ldots, Z_2, Z_3, \ldots$, where Z_i stands for a run of i zeros.

As example, the first 5 lines of Figure 3 would be encoded by the sequence: H_1(ESC), $H_2(0)$, $H_3(Z_3)$, H_1(:), H_1(`organigramme`), $H_3(1)$, H_1(`de`), $H_3(0)$, H_1(ESC), $H_2(11)$, 0, $H_3(0)$, $H_3(1)$, $H_3(1)$, $H_3(0)$.

3.3. Results

The bilingual text used for evaluating the new algorithm comprises the English and French versions of the European Union's JOC corpus, a collection of pairs of questions and answers on various topics. These texts, used on the ARCADE alignment evaluation project [18], were supplied aligned at the question/answer (paragraph) level. As the translation is rather precise, correct word- and phrase-level alignments reside quite close to the linear alignment of each paragraph pair. The automatic word- and phrase-level alignment as well as the bilingual glossary were obtained using an extended version of the *word_align* algorithm [7].

The English raw text has about 1,050,000 words, whereas the respective French text consists of about 1,162,000 words. Table 1 brings the sizes of the compressed French file (as a fraction of the original) for various compression schemes: GZIP, based on LZ77, BZIP, based on the Burrows-Wheeler transform, HWORD, a Huffman code encoding each of the different words in the text as single items, and finally TRANS, the algorithm suggested in this work, based on the translation from the English parallel.

The numbers do not include the sizes of the auxiliary files for TRANS and HWORD, since in the scenario of a large multilingual information retrieval system,

dictionaries and glossaries are needed anyway and are not stored exclusively as an aid for compression. However, even if those sizes are to be considered, it should be kept in mind that, according to Heaps' Law [11], the size of a dictionary for a text of size n is expected to be αn^{β}, where $0.4 \leq \beta \leq 0.6$. The total size of the auxiliary dictionaries for the current evaluation corpus, compressed using BZIP (rather than a dictionary-oriented compression scheme), is about 9% of the French raw text. Should a 1GB corpus be compressed, then corresponding dictionaries would comprise less than 0.9% of the original text. Obviously, specific dictionary compression can further decrease that rate.

Full size	GZIP	BZIP	HWORD	TRANS
7551550	0.307	0.214	0.225	0.212

TABLE 1: *Comparison of compression efficiency*

As can be seen, TRANS is better than GZIP, HWORD and BZIP, even without attempting to optimize the code further. Additional savings can be achieved by using an improved alignment module, transforming a larger part of the file into pointers rather than words, or by improving the encoding schemes. Consider, for example, again the table in Figure 3. At first sight, having a variant number associated with words like `agent` seems reasonable, as the word could also appear in plural form `agents`, but getting such a number for a preposition like `dans` might be surprising. A closer look however reveals that almost every word appears in at least two forms: all lower case and capitalized (except, obviously, special words like punctuation signs). This suggests the following strategy (not yet implemented).

Only one form of every word will be kept, using capitalization for proper nouns and lower case for the other words. If a word appears at the beginning of a sentence (follows a period or similar mark), it will be assumed to be capitalized. Exceptions, which should be rare, are handled by adding a codeword for an OVERRIDE, which will be encoded as part of the Huffman tree H_1 and will have the interpretation of (a) being followed by another codeword w from H_1; and (b) changing the case of the first letter of the word represented by w. The OVERRIDE will be used in case of lower case proper nouns (like in email addresses) or capitalized other words in the middle of a sentence. The effect of such a change will be to reduce the number of variants, so that smaller numbers will be encoded, and in some cases, if the number of variants is reduced to 1, no encoding at all is needed.

A significant improvement can be achieved if the compression scheme is altered with the following one. The source text will be kept in lemmatized form, that is, each word will be replaced by its lemma along with the suitable variant (inflection) code. This is not expected to worsen the compression rate of the source text, as lemmatization decreases the number of different strings, which results in a shorter code for them. The savings should be much greater than the additional space needed for the variant codes, as their variety is very low. This method has already improved our results when we applied it to the unaligned target words, as described

above. Now, the lemmatization information, currently integrated into target-text pointers, can be simply omitted.

Another optimization could be to compare, for each item, the number of bits required to encode it with reference to its translation with the number of bits needed for the corresponding word using a word based Huffman code, that is, it might sometimes pay to consider a term that could be aligned as if it were unaligned. The resulting hybrid algorithm improves on both the original form of TRANS and on HWORD.

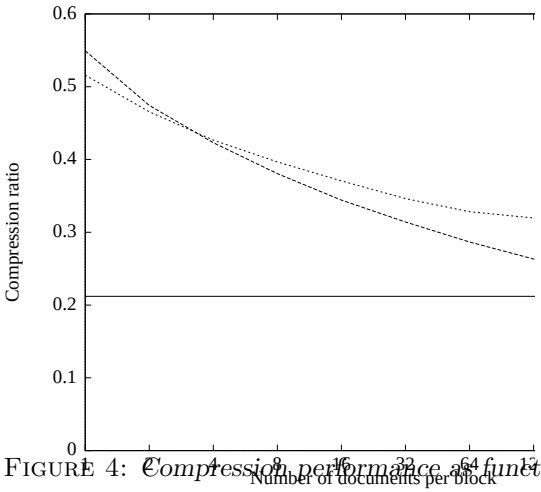

FIGURE 4: *Compression performance as function of basic block size*

GZIP and BZIP are adaptive methods and not really competitors for the applications treated here. The full decoding of the entire corpus is rarely needed, and small sub-parts, such as a single question/answer document, should be accessible individually. This is, however, not the case for adaptive methods, which require a sequential scan from the beginning of the file, while methods like TRANS and static Huffman coding support selective access and decoding. One can, of course, encode smaller parts of the file individually also by GZIP and BZIP, but compression will deteriorate. Figure 4 shows the relative size of the compressed French file for the various methods, as a function of the size of a basic block, which is supposed to be encoded independently from the others. This size is expressed by the number of consecutive question/answer documents in each block. For example, if each document is compressed on its own, compression by GZIP and BZIP reduces the full file only to 0.516 and 0.549, respectively, while TRANS stays at 0.212. With increasing block size, compression by the adaptive methods improves, but approaches the performance of TRANS only for very large blocks of more than 500 documents.

The ability of compressing small blocks without paying for it, is a good reason to think of an efficient search method for compressed parallel texts, based on space-efficient indices. The idea is to store a full index for the source text, while the indices for the other versions will only include unaligned tokens. Keeping partial

indices is extremely beneficial for large IR systems, as the size of an index is linear in the size of the respective text. A search within target texts will be done by extending the search query using the translation dictionary. This extension will be performed by extracting the source equivalents of the search words or phrases. Then, a list of candidate blocks will be built using the source-text index. Knowing the binary representation of the target text, it is possible to invalidate many of those candidates by applying pattern-matching methods to the compressed text. The remaining candidate blocks will be decoded to verify the match. However, as block sizes are very small, this process should be quite rapid. Note also that the processing of various sub-queries as well as candidate elimination and verification can be done in parallel.

4. Conclusion and Future Work

The existence of the same text in several languages can be used to improve the compression of a multilingual system. We have presented preliminary tests for two languages, achieving a good performance. By fine tuning the encoding, the compression results may be improved.

We intend to test our method on much larger parallel corpora of various languages, in order to obtain more reliable and generic results. We plan to explore also the possibility of bidirectional bilingual compression, where pointers can refer both from S to T and vice versa, which could lead to improvements, since phrases may have different lengths in different languages. A further topic to be treated is searching the compressed bilingual text using efficient indexing and pattern matching in the compressed text.

And last, but not least: we would like to seek for a generic model for k languages, which would be not just a trivial extension of the bilingual model, but would rather take advantage of the even greater redundancy existing in a multilingual text.

ACKNOWLEDGEMENTS: This work has been supported in part by Grant 25915 of the Israeli Ministry of Industry and Commerce (Magnet Consortium KITE). The first author also wishes to express his gratitude for the support by a grant from GLOBES, the Israeli business daily.

References

1. Ahrenberg, L., Andersson, M., and Merkel, M. A knowledge-lite approach to word alignment. In J. Véronis, editor, *Parallel Text Processing*, pages 97–116. Kluwer Academic Publishers, Dordrecht, 2000.
2. Ajtai, M., Burns, R. C., Fagin, R., and Long, D. D. E. Compactly encoding unstructured inputs with differential compression. *Journal of the ACM*, 49(3):318–367, 2002.
3. Brown, P. F., Della Pietra, S., Della Pietra, V. J., and Mercer, R. L. The mathematics of statistical machine translation: parameter estimation. *Computational Linguistics*, 19(2):263–311, 1993.
4. Burns, R. C. and Long, D. D. E. Efficient distributed backup and restore with delta compression. In *Workshop on I/O in Parallel and Distributed Systems (IOPADS)*.

ACM, 1997.

5. Carbonell, J. G., Yang, Y., Frederking, R. E., Brown, R. D., Geng, Y., and Lee, D. Translingual information retrieval : A comparative evaluation. In *Proc. IJCAI*, pages 708–715, 1997.
6. Choueka, Y., Conley, E. S., and Dagan, I. A comprehensive bilingual word alignment system : Application to disparate languages: Hebrew and english. In J. Véronis, editor, *Parallel Text Processing*, pages 69–96. Kluwer Academic Publishers, Dordrecht, 2000.
7. Dagan, I., Church, K. W., and Gale, W. A. Robust bilingual word alignment for machine-aided translation. In *Proc. of the Workshop on Very Large Corpora: Academic and Industrial Perspectives*, pages 1–8, Columbus, Ohio, 1993.
8. Fraenkel, A. S. and Klein, S. T. Novel compression of sparse bit-strings. In Apostolico, A. and Galil, Z., editors, *Combinatorial Algorithms on Words*, volume F12 of *NATO ASI Series*, pages 169–183. Springer Verlag, Berlin, 1985.
9. Gale, W. A. and Church, K. W. A program for aligning sentences in bilingual corpora. *Computational Linguistics*, 19(3):75–102, 1993.
10. Gaussier, É., Hull, D., and Aït-Mokhtar, S. Term alignment in use : Machine-aided human translation. In J. Véronis, editor, *Parallel Text Processing*, pages 253–274. Kluwer Academic Publishers, Dordrecht, 2000.
11. J. Heaps. *Information Retrieval : Computational and Theoretical Aspects.* Academic Press, Inc., New York, NY, 1978.
12. D. Hiemstra. Using statistical methods to create a bilingual dictionary. Master's thesis, Universiteit Twente, 1996.
13. S. T. Klein. Techniques and applications of data compression in information retrieval systems. In C. Leondes, editor, *Database and Data Communication Network Systems*, volume 2, chapter 16, pages 573–633. Elsevier Science, San Diego, CA, 2002.
14. Moffat, A. and Zobel, J. Adding compression to a full-text retrieval system. *Software — Practice & Experience*, 25(8):891–903, 1995.
15. Nevill, C. and Bell, T. Compression of parallel texts. *Information Processing & Management*, 28:781–793, 1992.
16. M. Simard. Translation-text alignment: Three languages are better than two. In *Proc. of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, pages 2–11, June 1999.
17. J. Véronis, editor. *Parallel Text Processing.* Kluwer Academic Publishers, Dordrecht, 2000.
18. Véronis, J. and Langlais, P. Evaluation of parallel text alignment systems: The arcade project. In J. Véronis, editor, *Parallel Text Processing*, pages 369–388. Kluwer Academic Publishers, Dordrecht, 2000.
19. T. A. Welch. A technique for high-performance data compression. *IEEE Computer*, 17:8–19, June 1984.
20. Witten, I. H., Moffat, A., and Bell, T. C. *Managing Gigabytes: Compressing and Indexing Documents and Images.* Van Nostrand Reinhold, New York, 1994.
21. Ziv, J. and Lempel, A. A universal algorithm for sequential data compression. *IEEE Trans. on Inf. Th.*, IT–23:337–343, 1977.