Pamukkale Üniversitesi
Mühendislik Bilimleri Dergisi
Cilt 16, Sayı 3, 2010, Sayfa 235-245

Exploiting Redundancy to Achieve Lossy Text Compression

Fazlalıktan Yararlanarak Kayıplı Metin Sıkıştırma Gerçekleştirimi

Ebru CELIKEL CANKAYA[a]*, Venka PALANIAPPAN[b] ve Shahram LATIFI[c]

[a]University of North Texas, Dept. of Computer Science and Engineering, Denton, TX, USA, 76207
[b]University of Nevada Las Vegas, Electrical & Comp. Eng. Dept., Las Vegas, NV, USA, 89154
[c]University of Nevada Las Vegas, Electrical & Comp. Eng. Dept., Las Vegas, NV, USA, 89154

Geliş Tarihi/Received : 11.01.2010, Kabul Tarihi/Accepted : 06.05.2010

ABSTRACT

Regardless of the source language, text documents contain significant amount of redundancy. Data compression exploits this redundancy to improve transmission efficiency and/or save storage space. Conventionally, various lossless text compression algorithms have been introduced for critical applications, where any loss after recovery is intolerable. For non-critical applications, i.e. where data loss to some extent is acceptable, one may employ lossy compression to acquire superior efficiency. We use three recent techniques to achieve character-oriented lossy text compression: Letter Mapping (LM), Dropped Vowels (DV), and Replacement of Characters (RC), and use them as a front end anticipating to improve compression performance of conventional compression algorithms. We implement the scheme on English and Turkish sample texts and compare the results. Additionally, we include performance improvement rates for these models when used as a front end to Huffman and Arithmetic Coding algorithms. As for the future work, we propose several ideas to further improve the current performance of each model.

Keywords: *Lossy text compression, Letter mapping, Dropped vowels, Replacement of characters.*

ÖZET

Kaynak dil her ne olursa olsun metin dosyaları, kayda değer miktarda tekrar (fazlalık) içerebilmektedir. Veri sıkıştırma, bu fazlalığı kullanarak ileti etkinliğini artırmayı ve bilgi depolama masrafını azaltmayı amaçlar. Geleneksel olarak, kodlanan verinin çözülmesi sırasında kaybın tolere edilemeyeceği kritik uygulamalarda kullanılmak üzere, çok çeşitli kayıpsız sıkıştırma algoritması geliştirilmiştir. Belirli bir dereceye kadar veri kaybının tolere edilebileceği kritik olmayan uygulamalar için, daha iyi etkinlik elde etmek adına, kayıplı sıkıştırma algoritmalarından faydalanılabilir. Bu çalışmada, karakter tabanlı kayıplı sıkıştırma sağlamayı hedefleyen üç yeni teknik - Harf eşleme (LM), düşürülen sesliler (DV), ve karakterlerin değiştirilmesi (RC) modelleri – kullanılarak geleneksel sıkıştırma algoritmalarının performansının iyileştirilmesi öngörülmektedir. Adı geçen modeller İngilizce ve Türkçe örnek metinler üzerinde çalıştırılarak sonuçları karşılaştırılmıştır. Buna ek olarak çalışmada, önerilen modeller Huffman Kodlaması ve Aritmetik Kodlama gibi yaygın olarak kullanılan geleneksel sıkıştırma algoritmalarına ön yüz olarak kullanıldığında kaydedilen performans iyileşme değerleri de yer almaktadır. Makale kapsamında, gelecekteki çalışmayla ilgili olarak, herbir modelin mevcut performansını artırmaya yönelik çeşitli öneriler de sunulmuştur.

AnahtarKelimeler: *Kayıplı metin sıkıştırma, Harf eşleme, Düşürülen sesliler, Karakterlerin değiştirilmesi.*

1. INTRODUCTION

Text compression plays a fundamental role in transferring and storing text by providing bandwidth optimization and significant cost savings in storage. Though typically applied to image data, lossy compression is also applicable on text.

In general, many applications benefit from text compression. To improve text compression rates, studies are carried out to determine how much recovery is possible with lossy compression. These studies have proven that human brain is capable of

* Yazışılan yazar/Corresponding author. E-posta adresi/E-mail address : ecelikel@cs.unt.edu (E. Çelikel)

recognizing words correctly, even if some characters are misspelled or missing. The ability to capture and fill in these (un) intentional gaps provides the basis for developing lossy compression tools with better rates.

Depending on the application, compression with a loss to some extent can well be tolerated. Take, for instance, a message archiving application, where even without exact recovery, the intended content can still be restored. In some other applications, such as short hand writing, some abbreviations can be restored without ambiguity. These examples are promising for lossy text compression to be effective.

The objective of this paper is to locate redundancy in text and exploit it to compress better. We particularly concentrate on lossy compression, which inevitably involves error to a certain degree. This error rate is calculated by the difference between original text prior to compression and the text recovered after decompression. In the field of compression, although tolerating errors may sound inacceptable, there are certain applications where reader can still understand the content after recovery, even though it is not a 100% successful one. Such applications can be listed as -but not limited to- instant messaging, chatting, online journals, blogs, message archiving, etc. The human mind is intelligent enough to recover the meaning of a text even though some characters are missing or misspelled. The rate of success for recovery increases with such factors as experience, relevance to the field, age, interest, etc.

2. RELATED WORK

Optimal bandwidth utilization is a critical requirement in network communication, and data storage. This can be achieved with compression, either lossless or lossy. Consequently, compression is an intensively studied field and several studies exist on text compression. For lossless compression, statistical properties of text play a fundamental role in achieving good compression rates: Simply the longest patterns that occur repeatedly in text are replaced with shorter sequences to provide compression. The longer this pattern gets, and the more frequent it occurs in text, the better gets the compression rate. Algorithms such as Arithmetic Coding (Bose and Pathak, 2006), Huffman Coding (Shukla et al., 2009), Prediction by Partial Matching (PPM) (Korodi and Tabus, 2008) all exploit this statistical property: repeatedly occurring text. Burrows Wheeler Transform (BWT) (Gilchrist and Cuhadar, 2007) is a slightly different algorithm, which also utilizes statistical properties of text, and uses permutation to better compress it. In addition to lossless compression, there are also various lossy compression approaches, such as transform coding that converts the input data to a new domain to better represent the content. The examples of transform coding are discrete cosine transform (Zhou and Chen, 2009), fractal compression (Jorgensen and Song, 2009), wavelet compression and its variations (Liu and Zhao, 2007), vector quantization (Kruger et al., 2008), and linear predictive coding (Nagarajan and Sankar, 1998).

In compression literature, several scholar works exists on lossy text compression. Witten et al., (1994a) introduce semantic and generative models to obtain improved text compression rates. (Howard, 1996) takes text as an image file and applies pattern matching to achieve compression. Another interesting approach is to replace longer words with their shorter synonyms (Nevill and Bell, 1992). This method is called thesaurus technique, and performs better especially in morphologically rich (inflected) source languages. The thesaurus method preserves semantics throughout text. A specific type of lossy compression finds itself a vast area of implementation in the field of programming languages: By simply removing multiple wildcards and/or comment lines, one can achieve lossy compression to significantly high levels.

Palit and Garain, (2006) introduce watermarking over lossy compressed text to provide verification of text ownership.

Representing text as image (called textual image) is a common practice in lossy compression. (Broder and Mitzenmacher, 1996) develop a compression scheme called GRID to represent text documents as gray scale images. Some researchers bring together lossless and lossy approaches to utilize the benefits of the two: For example, (Witten et al.,1994b).

implement a two stage compression on image representation of texts. In the first stage, the authors apply lossy compression by matching a group of pixels representing certain characters in textual image against an adaptively built library of patterns seen so far. In the second stage, they use the reconstructed image to encode the original image with a statistical context-based lossless compression. While some of the scholar work considers lossless and lossy compression separately, some others, such as Kaufman and Klein, (2004) generate algorithms as a hybrid implementation and name it as semi-lossy compression. Textual image compression has been addressed by the works of Ye and Cosman in (Ye and Cosman, 2001) and (Ye and Cosman, 2003). A different version of textual image compression relies on OCR based text image compression and is studied by (Shang et al., 2006).

Palaniappan and Latifi, (2007) introduce three simple yet effective ideas to accomplish lossy text compression: Letter Mapping, Dropped Vowels and

Replacement of Characters. The authors present results on English text. Our lossy text compression approach expands these techniques by applying them on Turkish text, and also combining them with Arithmetic Coding other than Huffman Coding. This simple approach gives the opportunity to explore many fields involving text processing. For example, applications such as instant messaging and e-mail transmission will benefit from this technique both in transmission time and storage space. Online diaries or blogs are rapidly growing and these areas can also benefit from the lossy text compression scheme we expand.

Theoretically, our design lends itself as a front end to any lossless compression algorithm. We anticipate our front end to offer compression improvement on lossless compression algorithms. Although this paper presents the execution of our scheme succeeded by Huffman and Arithmetic Coding, in practice, it can be succeeded by any lossless compression algorithm.

3. MATERIAL AND METHODS

In the following three subsections, we revisit the three lossy text compression techniques and propose some expansions.

3. 1. Letter Mapping (LM) Method

The Letter Mapping (LM) method is simply a character replacement process that is applied to the entire text. The block diagram of LM compression is illustrated in Figure 1.

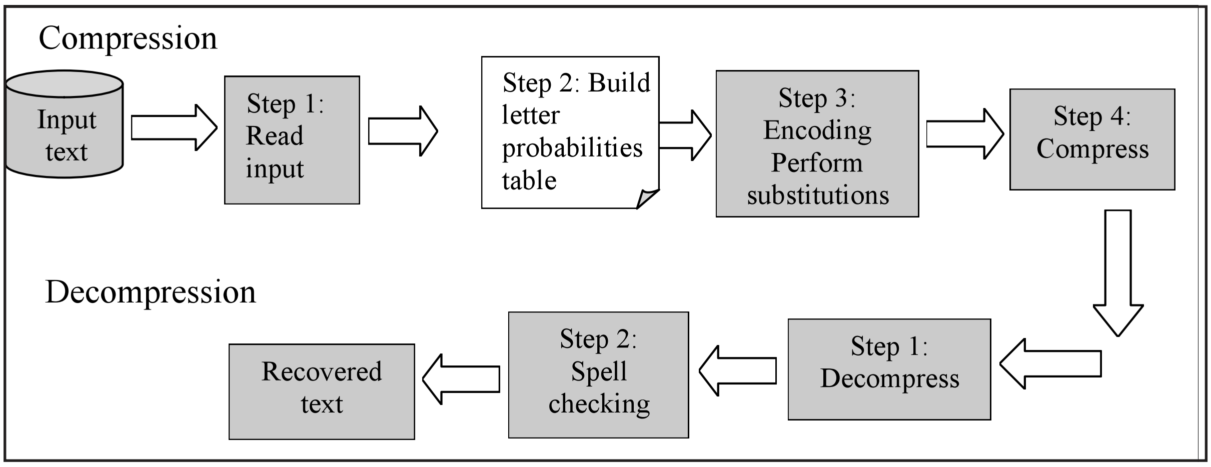

Figure 1. Letter mapping (LM) model flow diagram.

The LM method removes a predetermined number of least frequently occurring characters and replaces them with the most frequently occurring characters. For experimental work, the authors use the ordered letter frequencies of a typical English text in Figure 2 (Lewand, 2000).

Not surprisingly, majority of the most frequent letters in English are vowels such as e, a, o, and i. We deliberately leave the vowels for implementing the next compression technique, i.e. Dropped Vowels (DV), and take only consonants instead. As can be seen from Figure 2, characters such as t, n, s, h, r are likely to occur with higher probability then characters k, j, x, q and z.

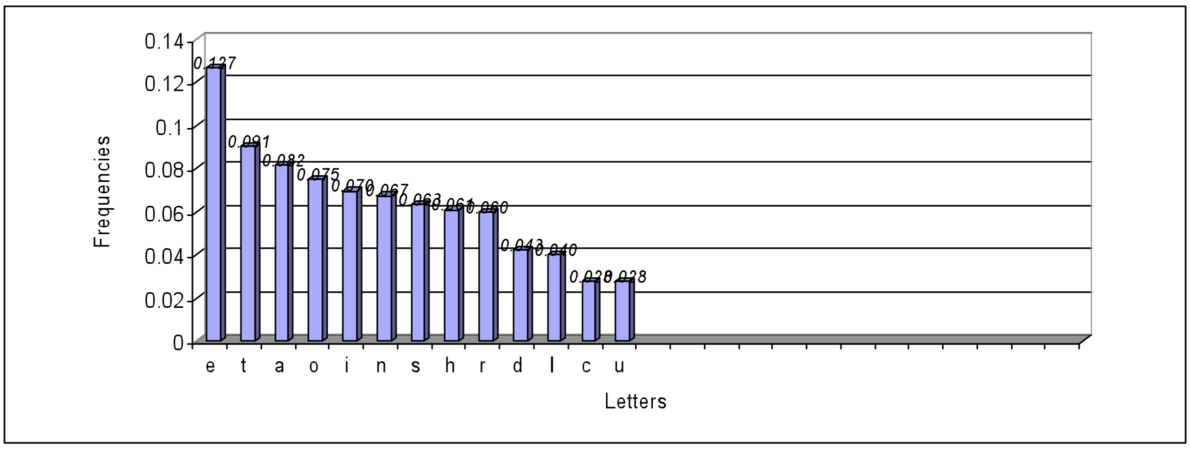

Figure 2. English letter frequencies.

While implementing this technique, our gain becomes two fold: First, we aim at achieving lower number of bits per symbol (to be referred to as bit rate from this point on) during encoding, i.e. compression to some extent. We employ Huffman Coding for this encoding. The overall idea behind Huffman Coding is that more probable characters are assigned shorter bits (Lee et al., 2009). Hence, by replacing the least frequently occurring characters with more frequently occurring ones, a better compression rate is highly likely to be achieved. Second, when we replace the least occurring characters with more frequent characters, the source alphabet size is reduced, in the ongoing example by a rate of 5 (from whole English alphabet set to 26-5 = 21 English letters), which suggests further compression.

At the decoding end, decompression is applied on the encoded text in the hope of recovering the original text as much as possible. Still, the text recovered will most probably be erroneous due to the existence of misspelled words. This problem can easily be solved with an access to comprehensive dictionaries and thesauri. By running the recovered text through a simple spell checker, most of these errors can easily be removed. Deciding which words will be corrected with the spell checker is quite straightforward: Words in error would be the ones containing the replaced least frequent characters of the original text, namely k, j, x, q, and z, in them. Therefore, the spell checker only needs to look for the words containing these characters in the decoding end. The spell checker module does not come free from complications, though. Evidently, spell checker does not guarantee full recovery of the original text, because some recovered words with replaced characters will not be captured by the spell checker. The reason for missing such words is that although did not occur in the original text, some reconstructed words may still be semantically correct. Take the word “try” transformed into word “cry” after implementing LM, for example.

While measuring the performance of LM compression, we have used different text documents to explore which application (for example E-mail messages, blogs, newspaper articles, etc.) is most suitable for LM compression with a tolerable error rate. The process of one-to-one mapping between character pairs is also varied to discover the most efficient text recovery.

3. 1. 1. Compression

The compression module takes the input text, extracts character probabilities with a single pass over the text, and applies encoding, which serves as a front end prior to implementing lossless compression. The front end encoder later on differentiates as either letter mapping (LM), or dropped vowels (DV), or replacement of characters (RC). The compression module is finalized with the conventional compression algorithm. For this work, this algorithm is chosen as Huffman Coding. Following explains the internal details of the compression scheme.

Step 1: Read the input text. This step requires the entire input text to be stored in advance, instead of applying the process on the fly. For this reason, for applications that involve large amount of text processing, the input buffer should be held large.

Step 2: With a single pass through input text, compute the frequency of each character and build a letter probabilities table. For example: Assume that Table 1 is constructed by scanning through the sample English text that is mentioned in Section 3.1. (Lewand, 2000). According to this table, the first five consonants with highest probability of occurrence are t, n, s, h, and r.

Table 1. Ordered English letter frequencies.

	Letter	Frequency
1	e	0.12702
2	t	0.09056
3	a	0.08167
4	o	0.07507
5	i	0.06966
6	n	0.06749
7	s	0.06327
8	h	0.06094
9	r	0.05987
10	d	0.04253
11	l	0.04025
12	c	0.02782
13	u	0.02758
14	m	0.02406
15	w	0.0236
16	f	0.02228
17	g	0.02015
18	y	0.01974
19	p	0.01929
20	b	0.01492
21	v	0.00978
22	k	0.00772
23	j	0.00153
24	x	0.0015
25	q	0.00095
26	z	0.00074

With simple table look up, individual probabilities of these letters are determined as: P(t)= 0.09056, P(n)= 0.06749, P(s)= 0.06327, P(h)= 0.06094, and P(r)= 0.05987, respectively. Similarly, the last five consonants with the lowest probability of occurrence for the same English text are k, j, x, q, and z, and their individual probabilities are retrieved from Table 1 as: P(k)= 0.00772, P(j)= 0.00153, P(x)= 0.0015, P(q)= 0.00095, and P(z)= 0.00074.

Step 3: The maxNumberOfSubstitutions variable will represent the number of letters to be mapped. Set this variable to a value that is less than or equal to |A|/2 -where |A| is the source alphabet size-, because with 1:1 mapping, our scheme can map at most |A|/2 (least frequent letters) to (most frequent) letters. Using the probability values calculated in Step 2, replace the first most frequent letter with the first letter in the least frequent letters list, replace the 2nd most frequent letter with the next letter in the least frequent letter list, until the end of the list is reached. The end of list is determined by the value of numberOfSubstitutions variable, that was initialized earlier in Step 3. As an example: Assume numberOfSubstitutions=5. Then, in the ongoing example our scheme will take the least frequent letters in the original text as (k, j, x, q, z) and replace them with (t , n, s, h, r) in the given order. Therefore, the 1:1 mapping will replace k with t, j with n, x with s, q with h and z with r.

Step 4 : Compress the encoded sequence using a standard compression method such as Huffman Algorithm or Arithmetic Coding.

3. 1. 2. Decompression

The decompression module takes the compressed text from compression module and applies decompression on it to achieve initial recovery. After, a spell checking submodule is employed in the hope of recovering as closer to the original text as possible. Since this is a lossy compression scheme, we do not expect a perfect recovery. Still, using spell checker is beneficial and removes a significant amount of erroneous characters. The following summarizes the tasks of the decompression module.

Step 1 : Decompress the received text and send it to a spell checker.

Step 2 : The spell checker corrects the errors. The authors want to remind at this point the fact that this correction does not guarantee 100% recovery of the original text. The recovered text may contain a significant amount of false positives with incorrectly recovered meaningful words, which did not exist in the original text before LM encoding. This ambiguity arises from the nature of the source language itself: Some words are syntactically very close to each other, although their meanings are completely different, such as words *"far"* and *"car"* in English. Still, this is not of main concern for our scheme, because it targets lossy compression, not a lossless compression. If further correction is required, a human tester or a semantic parser can be used in a second round to clarify such ambiguities.

3. 2. Dropped Vowels (DV) Method

In a typical English text, the ratio of vowels to consonants is approx. 58% (Pollo et al., 2005), while for Turkish it is calculated as 76.46% on average for the 4 sample text from test set. Hence, for both source languages, the vowels occur with a surprisingly high frequency throughout the text. This is not only important in the written text, but also in speech processing studies, because this property helps improve audibility for the hearing impaired (Kewley-Port et al., 2007).

The idea we use for dropped vowels (DV) method is quite simple: If we drop all vowels from text, it will disrupt the readability of the entire text. Still, the original text can be fully or partially recovered by using a spell checker. And, if applied to the entire text, the context help us place best probable characters into unknown slots.

The flow diagram of DV model is very similar to that of LM model (Figure 1). Since vowels occur with significantly high frequency in a typical English text, if the vowels were dropped from the text, a significant increase in the compression is achieved due to their frequency of occurrence. But we cannot drop all the vowels to get the best compression, as with all the vowels dropped, the problem of recovering the word again will be much harder. Under this restriction, we replace all vowels with a special character, e.g. the blank character ' ', or with one single vowel, say the letter *'e'* (Choosing letter "e" is for a twofold gain: First, *'e'* is the most frequent letter in the sample text, so will help yield better compression rates. Second, because some vowel to vowel encodings will be letter *'e'* to letter *'e'* mappings, this will not have to be corrected with spell checker). Although both approaches will give the same compression rate and bit rate, replacing all vowels with the letter *'e'* will yield a better error recovery rate. We then apply Huffman Coding as the compression tool to further compress the DV encoded text. With DV model, we obtain a reduced bit rate because instead of using the full alphabet letters as 26 characters, we use only 22 after encoding. At the decompression side, the first step, i.e. decompression is similar to that of LM method. In step 2, we use the spell checker again, in the hope for full recovery, which is not guaranteed always.

We propose transforming the dropped vowels (DV) model into a lossless compression scheme (Figure 3):

During compression, while performing all vowels to vowel *"e"* (or blank character) replacement, we can use a place holder to remember what the original vowel was. This is required for correct recovery in the decompression end. For this purpose, we record every vowel replacement in an auxiliary file with the extra information on what the original vowel was and send this file with the compressed file to the recipient.

On the receiving end, we no longer need the spell checker. We use auxiliary file for full recovery. So, the scheme becomes lossless. This brings an overhead of preparing and transferring the auxiliary file, which may get too large as the input size gets bigger.

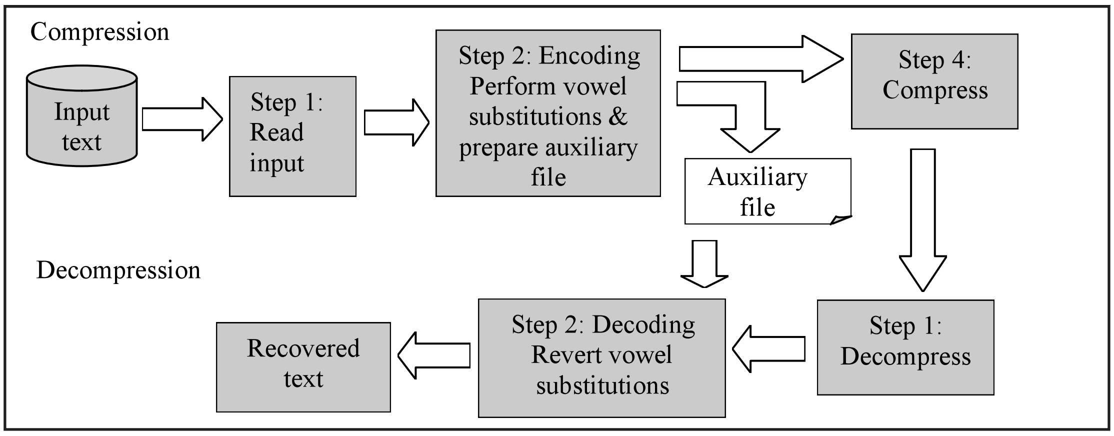

Figure 3. Modified dropped vowel (DV) model flow.

3. 3. Replacement of Characters (RC) Method

Based on previous work by Palaniappan and Latifi, 2007 we use the third compression model as Replacement of Characters (RC). This method exploits the well-known shorthand technique: Simulating a secretary taking notes, we represent a combination of several characters as one character. Determining which characters to combine, and which character to use to represent this combination is quite flexible. This allows us to reduce the number of characters to be processed in advance. To further improve RC encoding, we replace capital letters with lower case letters and thereby subtracting 26 characters from the alphabet size. This provides improvement in bit rate, as well.

As was the case with dropped vowels (DV) model, the RC model also inherits the similar flow diagram from letter mapping (LM) model. The only part that differs is the actual encoding itself.

The recovered text may be highly erroneous; still this could be acceptable as long as one can understand it. RC type of lossy compression is more suitable for applications where ability to read the content is more important than the text itself. Therefore, this technique uses less space to store the same amount of information.

3. 4. Comparison of Three Models

The LM model compares to the DV model in two aspects: One, the former is a lossy compression scheme, while the latter can be easily transformed into a lossless compression scheme with the proposed modification (Figure 3). Still, LM model could be preferable because it does not require the extra preparation and transfer of the auxiliary file, which may grow too large in proportion with the original text size. Second, the performance of the LM model will differ how frequently the most and least frequent letters occur in the given text. Likewise, the performance of the DV model will differ based on the vowel to consonant ratio in the given text, which might differ significantly from one text to another.

The performance of the RC model depends on which character set we encode with a single symbol. Due to the flexibility offered, this model is the most promising one among the three. Because as a theoretical extreme, we can even represent the whole text with one letter, which may never be the case with LM or DV models.

In practice, all three models are prone to fluctuations in terms of compression performance.

In the next section, we present compression rates obtained with three lossy compression techniques on English and Turkish sample texts and their comparisons.

4. EXPERIMENTAL WORK AND RESULTS

To compare compression performances of each technique, we compiled English and Turkish test sets of 10 and 4 texts from a variety of domains, respectively.

4. 1. Compression Performances

Our design is a front end that can precede any lossless compression algorithm. We present implementation results on the two most common conventional lossless compression algorithms: Huffman and Arithmetic Coding. For English, we first run plain Huffman Coding on each text of the test set. After, we run our scheme with each of the

3 models as LM, DV, and RC succeeded by Huffman Coding on the same test set. We then computed the rate of improvement on Huffman compression and plot the chart in Figure 4.

Figure 4 shows that each model behaves differently on individual elements of the test set. This result supports our anticipation: Performance improvement (or degradation) relies basically on the nature of the text: For LM model, it depends on the statistical distribution of the most and least frequent letters, for DV model the ratio of vowels to consonants, and for RC model, the statistical distribution of what we choose as the short hand representation(s). These distributions are obviously different in elements of the test set; which explains differences even in the same model.

Figure 4 also shows that for English, the best model among the three is RC model, which performs better for most of the text files. The results for text1 are very different from the rest of the test results. This implies that text1 is not suitable for RC encoding. DV model is the next best model after RC model, and LM is the least performing model in terms of compression improvement. These results conform to our earlier expectations, as we explained in Section 3.4 as the extreme case, where RC encodes the whole text with one single character.

We repeated experiments by employing three models as a front end to Huffman Coding on Turkish test set. We further expanded the scheme to serve as a front end to Arithmetic Coding, as well. Figure 5 presents the compression rate (bpc) values on Turkish test set for Huffman Coding.

Figure 5 shows that when applied as a front end to Huffman, each model yields better performance than that of plain Huffman compression. DV model performs best for each Turkish text, while RC model performs consistently poorer. This can be explained as the characteristic difference between source languages: For English, we can find more letter sequences that appear multiple times in text, while for Turkish, there is not that many such occurrences to exploit for the RC model. Also, the Turkish test set apparently has less redundancy than English test set. By expanding test sets to include more texts that are representative of more domains, this difference can be better explained. Using each model as the front end and Arithmetic Coding as the compression algorithm, we obtained the *bpc* values in Figure 6 on Turkish texts.

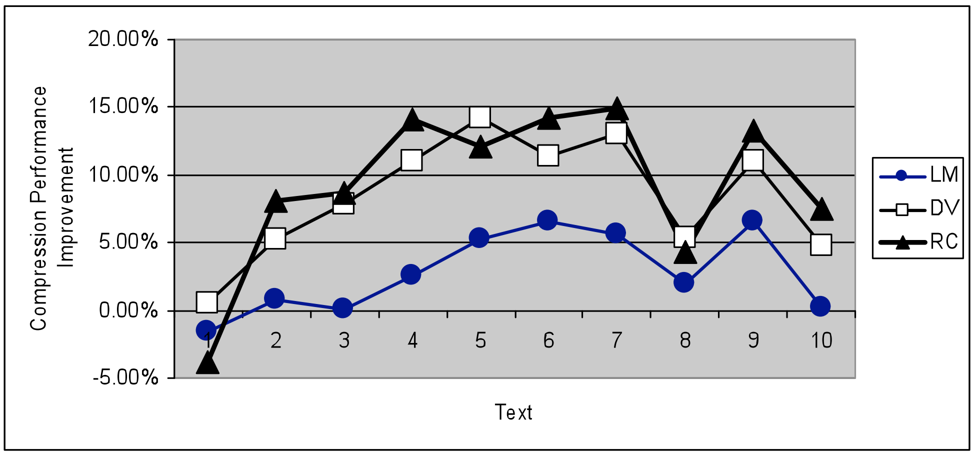

Figure 4. Compression improvement chart for LM, DV and RC models on English.

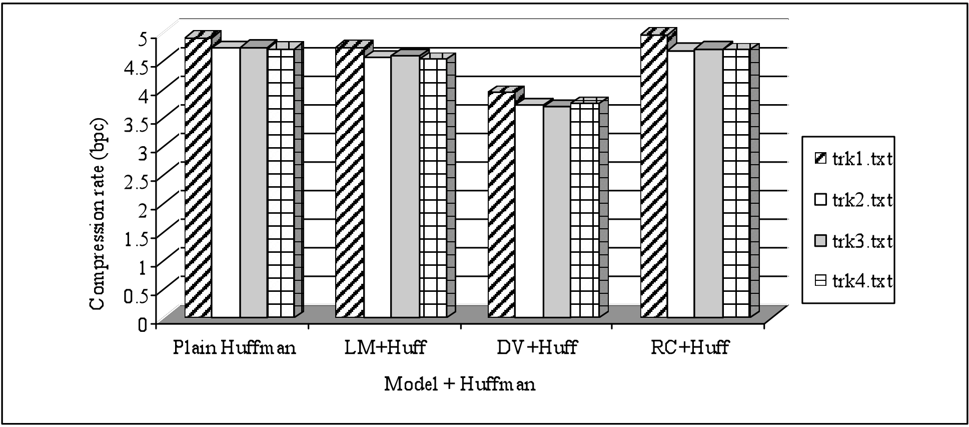

Figure 5. Models + Huffman Algorithm on Turkish.

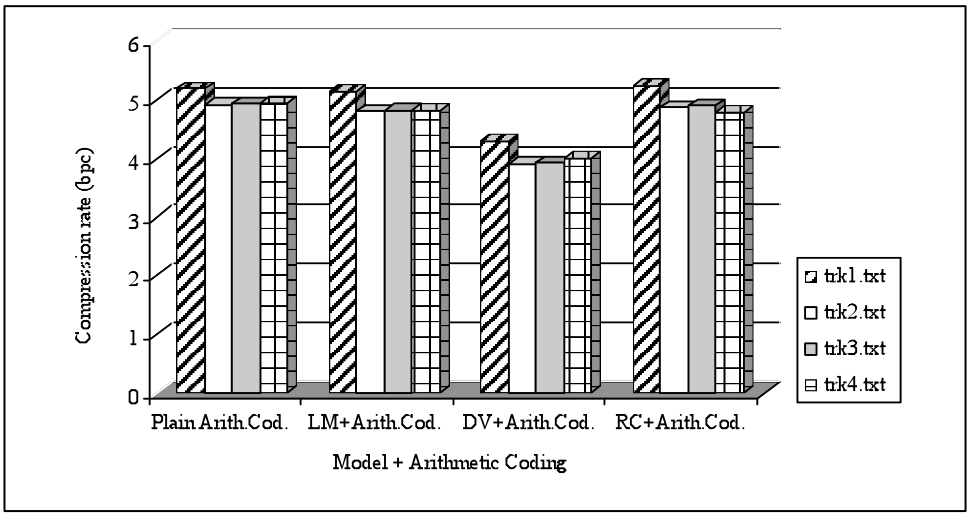

Figure 6. Models + Arithmetic Coding on Turkish.

As seen in Figure 6, although performs poorer (higher *bpc* rates) than Huffman algorithm, results with Arithmetic Coding are similar to that of Huffman algorithm. On average, the percentage of improvement with DV model with Arithmetic Coding is 19.16%, which is very close to the corresponding value (20.76%) with Huffman algorithm.

4. 2. Information Theoretical Comparison

If we do not employ any encoding, the input to the compression algorithm will contain the entire source alphabet, with each alphabet letter having a different probability of occurrence. When we apply Huffman Coding as the compression tool, it assigns a unique binary code to each alphabet letter, based on its frequency of occurrence. To measure the performance of a coding scheme, we use three standard measures as entropy (*H*), bit rate (*l*), and redundancy (*R*), whose formula are given as (Palaniappan and Latifi, 2007):

$$H = -\sum P(letter_i) \times \log_2 P(letter_i) \quad (1)$$

$$l = \sum \left(P(letter_i) \times l_i\right) \quad (2)$$

$$R = l - H \quad (3)$$

Redundancy is a measure to determine how much more a text can still be compressed. If we compute redundancy before and after a certain encoding, we expect it to be lower to conclude that the encoding achieved a good level of redundancy.

Table 2 shows the calculation of entropy and bit rate values for the English sample text, for which letter statistics were given earlier (Figure 2 and Table 1).

Initially, we apply plain Huffman Coding to the sample text to calculate the redundancy value. Then, by employing each of the three encodings as LM, DV, and RC, we compute their redundancy levels as well and compare them with that of plain Huffman.

In Table 2, calculation of redundancy (*R*) involves bit rate (*l*), which involves code length (l_i). The code length calculation requires constructing the Huffman tree with minimal code lengths.

As seen from Table 2, the bit rate for plain Huffman is $l = \sum \left(P(letter_i) \times l_i\right)$ =4.2101 bpc for the sample text. Hence, redundancy for plain Huffman is calculated as R = 4.21008 - 4.1758 = 0.0343 bpc.

Similarly, we compute entropy (*H*), bit rate (*l*), and redundancy (*R*) values for LM and DV models. These values can be seen at Table 3, where all values are measured in bpc.

When LM encoding is used as a front end to Huffman algorithm, the alphabet size is reduced by the number of characters being mapped. This helps us obtain better performance than that of plain Huffman compression. The increased level of redundancy from 0.0343 bpc to 0.351 bpc in Table 3 verifies that justification.

According to Table 3, DV encoding yields even better performance (with highest level of redundancy as 0.0622 bpc) because it removes certain number of (5 for the ongoing experiment) characters with lower probability, and replaces each with one single symbol (letter *'e'* in this experiment). Therefore, the number of bits required to code the alphabet becomes less. Furthermore, since letter *'e'* is also the most frequent letter in the text, it helps improve the compression performance. The calculation of redundancy for RC encoding is deliberately not included here. Because of the flexibility of encoding it provides, one can obtain different compression rates, so there is no fixed compression rate for the RC model. Repeating entropy and bit rate calculations on the sample Turkish text *trk1.txt* (which is a daily newspaper article) we obtained the values in Table 4.

Table 2. Entropy & Bit rate for sample text (English).

$Letter_i$	$P(letter_i)$	$log_2(P(letter_i))$	Entropy: $H(letter_i)$	code length (l_i)	Huffman code	Bit rate: $P(letter_i)$ x l_i
e	0.12702	-2.977	0.378	3	110	0.38106
t	0.09056	-3.465	0.314	3	100	0.27168
a	0.08167	-3.614	0.295	4	0100	0.32668
o	0.07507	-3.736	0.280	4	0110	0.30028
i	0.06966	-3.844	0.268	4	0000	0.27864
n	0.06749	-3.889	0.262	4	0001	0.26996
s	0.06327	-3.982	0.252	4	0010	0.25308
h	0.06094	-4.036	0.246	4	1110	0.24376
r	0.05987	-4.062	0.243	4	1010	0.23948
d	0.04253	-4.555	0.194	5	01010	0.21265
l	0.04025	-4.635	0.187	5	01110	0.20125
c	0.02782	-5.168	0.144	5	00110	0.1391
u	0.02758	-5.180	0.143	5	00111	0.1379
m	0.02406	-5.377	0.129	5	11110	0.1203
w	0.0236	-5.405	0.128	5	10110	0.118
f	0.02228	-5.488	0.122	5	10111	0.1114
G	0.02015	-5.633	0.114	6	010110	0.1209
y	0.01974	-5.663	0.112	6	010111	0.11844
p	0.01929	-5.696	0.110	6	011110	0.11574
b	0.01492	-6.067	0.091	6	111110	0.08952
v	0.00978	-6.676	0.065	7	0111110	0.06846
k	0.00772	-7.017	0.054	7	1111110	0.05404
j	0.00153	-9.352	0.014	8	01111110	0.01224
x	0.0015	-9.381	0.014	8	01111111	0.012
q	0.00095	-10.040	0.010	8	11111110	0.0076
z	0.00074	-10.400	0.008	8	11111111	0.00592
Sum	1.00000		4.1758			4.21008

Table 3. Redundancy values for each model (English).

Compression Model	Bit rate (*l*)	Entropy (*H*)	Redundancy (*R*)
Plain Huffman	4.2101	4.1758	0.0343
LM + Huffman	4.1373	4.1022	0.0351
DV + Huffman	3.4040	3.3418	0.0622

Table 4. Entropy & Bit rate for sample text (Turkish).

$Letter_i$	$P(letter_i)$	$log_2(P(letter_i))$	Entropy: $H(letter_i)$	code length (l_i)	Huffman code	Bit rate: $P(letter_i)$ x l_i
a	0.1439	-2.797	0.402	3	100	0.4318
e	0.0821	-3.607	0.296	4	1010	0.3284
n	0.0714	-3.807	0.272	4	0110	0.2857
i	0.0672	-3.896	0.262	4	0011	0.2687
r	0.0672	-3.896	0.262	4	0100	0.2687
l	0.0544	-4.201	0.228	4	0001	0.2175
k	0.0533	-4.230	0.225	4	0000	0.2132
i	0.0480	-4.382	0.210	5	11111	0.2399
s	0.0469	-4.414	0.207	5	11110	0.2345
b	0.0458	-4.447	0.204	5	11101	0.2292
d	0.0405	-4.626	0.187	5	10110	0.2026
u	0.0405	-4.626	0.187	5	10111	0.2026
m	0.0352	-4.829	0.170	5	01011	0.1759
y	0.0352	-4.829	0.170	5	01110	0.1759
t	0.0320	-4.967	0.159	5	00101	0.1599
ü	0.0192	-5.704	0.109	6	111000	0.1151
o	0.0181	-5.786	0.105	6	011111	0.1087
h	0.0171	-5.873	0.100	6	011110	0.1023
ş	0.0171	-5.873	0.100	6	010101	0.1023
ç	0.0128	-6.288	0.080	6	001000	0.0768
g	0.0128	-6.288	0.080	6	001001	0.0768
z	0.0128	-6.288	0.080	7	1110011	0.0896
p	0.0075	-7.066	0.053	7	0101000	0.0522
c	0.0043	-7.873	0.034	8	01010010	0.0341
ö	0.0043	-7.873	0.034	8	01010011	0.0341
v	0.0043	-7.873	0.034	8	11100100	0.0341
f	0.0032	-8.288	0.027	9	111001010	0.0288
ğ	0.0032	-8.288	0.027	9	111001011	0.0288
j	0.0000		0.000			
Sum	1.0000		4.2518			4.4893

Using equations 1,2, and 3 we can compute the redundancy value for plain Huffman on sample Turkish text as R = 4.4893 – 4.2518 = 0.2375.

A drawback of our scheme is that, the better we get in compression, the higher gets the error rate, which is measured as the number of false positives after decompression. Figure 7 illustrates the compression improvement vs. error rate for English test set.

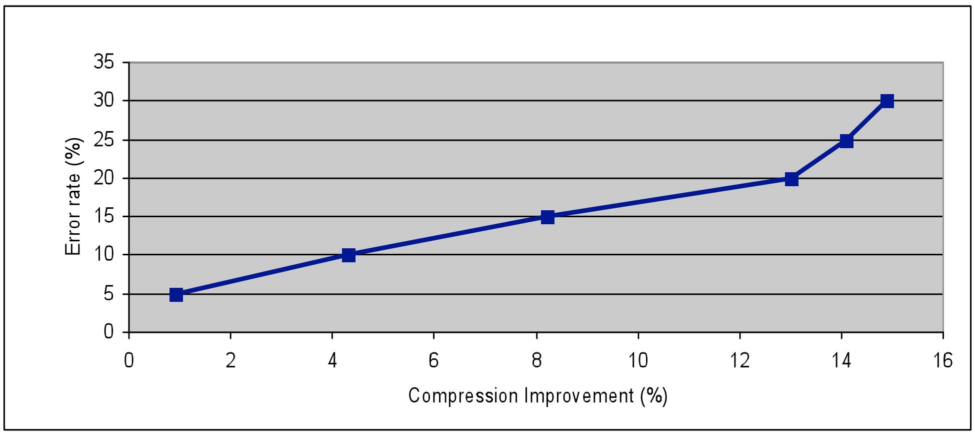

Figure 7. Compression improvement vs. Error rate (English).

Apparently, our scheme achieves better compression improvements to the expense of losing from recovered text accuracy.

5. CONCLUSION

This work presents three novel models for lossy text compression to achieve better compression rates. Regardless of the implementation details, each model simply introduces a front end encoding mechanism that can be complemented with a conventional lossless compression scheme afterwards. The paper utilizes Huffman algorithm and Arithmetic Coding for this purpose.

The first model introduced is letter mapping (LM) and replaces a certain number of the least frequent letters with the same number of most frequent letters on a 1:1 mapping basis. The second model is called dropped vowel (DV) technique, and simply replaces vowels of the source language with one single character. The third model is called replacement of characters (RC) model and based on the idea of short hand representation of long sequences of characters with one single symbol. The determination of this sequence is left to the user's discretion; therefore this scheme is highly probable to outperform the former two models.

We present experimental work on each model for English and Turkish test sets and demonstrate that in terms of compression rates, the models are ordered as RC, DV, and LM for English; and DV, RC, and LM for Turkish from best to worst performance.

6. FUTURE WORK

To decrease the false positives that we may encounter during LM compression, we suggest combining this model with a second pass, through which a human tester or an automated semantic parser detects semantically ambiguous words and corrects them. This idea is very promising and introduces a new edge to our work: The possibility of combining it with a natural language processing (NLP) implementation.

Due to the characteristics of three techniques employed, i.e. letter mapping (LM), dropped vowels (DV) and replacement of characters (RC), they yield different performances on different source languages. So, another avenue that we will further our work on is to apply these models on different source languages other than English and Turkish, and compare compression performances. This may help us introduce a new parameter to cross-language comparison studies.

Although we employed Huffman and Arithmetic Coding, theoretically, the compression module succeeding the encoder can be replaced with any lossless compression algorithm. So, as part of future work, we will employ several other lossless compression algorithms and measure the rate of performance improvement.

REFERENCES

Bose, R. and Pathak, S. 2006. "A Novel Compression and Encryption Scheme Using Variable Model Arithmetic Coding and Coupled Chaotic System", IEEE Transactions on CCts and Systems. 848-857.

Broder, A. and Mitzenmacher, M. 1996. "Pattern-based Compression of Text Images", **Proceeding of Data Compression Conference**, March 31-April 3, 1996, Snowbird, Utah, USA. 300-309.

Gilchrist, J. and Cuhadar, A. 2007. "Parallel Lossless Data Compression Based on the Burrows-Wheeler Transform", **AINA 2007**. 877-884.

Howard, P. G. 1996. "Lossless and Lossy Compression of Text Images by Soft Pattern Matching", IEEE Transaction. 210-219.

Jorgensen, P. E. T. and Song, M. 2009. "Analysis of Fractals, Image Compression, Entropy Encoding, Karhunen-Loève Transforms", Acta Applicandae Mathematicae: An Int'l Survey Journal on Applying Math. and Mathematical Appls. 108 (3), 489-508.

Kaufman, Y. and Klein, S. T. 2004. "Semilossless Text Compression", **Prague Stringology Conf**., Aug. 30 - Sept. 1, 2004, Prague, Czech Republic.

Kewley-Port, D. Burkle, T. Z. and Leed, J. H. 2007. "Contribution of Consonant Versus Vowel Information to Sentence Intelligibility for Young Normal-Hearing and Elderly Hearing-Impaired Listeners", Acoustical Soc. of America. 2365–2375.

Korodi, G. and Tabus, I. 2008. "On Improving the PPM Algorithm", **ISCCSP 2008.** 1450-1453.

Kruger H., Schreiber R., Geiser B. and Vary, P. 2008. "On Logarithmic Spherical Vector Quantization", **ISITA 2008**. 1-6.

Lee, Y. H., Kim, D. S., and Kim, H. K. 2009. "Class-Dependent and Differential Huffman Coding Of Compressed Feature Parameters For Distributed Speech Recognition", **ICASSP 2009.** 4165 – 4168.

Lewand, R. E. 2000. "Cryptological Mathematics", The Mathematical Association of America, USA.

Liu, G. and Zhao, F. 2007. "An Efficient Compression Algorithm for Hyperspectral Images Based on Correlation Coefficients Adaptive Three Dimensional Wavelet Zerotree Coding", **Int'l Conf. on Image Processing.** 341-344.

Nagarajan, S. and Sankar, R. 1998. "Efficient Implementation of Linear Predictive Coding Algorithms", **IEEE Southeastcon '98**. 69-72.

Nevill, C. and Bell, T. 1992. "Compression of Parallel Texts", Inf. Processing & Mgmnt., 28, 00-00.

Palaniappan, V. and Latifi, S. 2007 "Lossy Text Compression Techniques", **ICCS 2007**. 205-210.

Palit S. and Garain, U. 2006. "A Novel Technique For The Watermarking Of Symbolically Compressed Documents", **DIAL 2006.** 291-296.

Pollo, T. C., Kessler, B. and Treiman, R. 2005. "Vowels, Syllables, and Letter Names: Differences Between Young Children's Spelling in English and Portuguese", Journal of Experimental Child Psychology. 92 (2), 161-181.

Shang, J., Liu, C. and Ding, X. 2006. "JBIG2 Text Image Compression Based on OCR". **SPIE 2006**.

Shukla, P.K., Rusiya, P., Agrawal, D., Chhablani, L. and Raghuwanshi, B.S. 2009. "Multiple Subgroup Data Compression Technique Based on Huffman Coding", **CICSYN 2009.** 397-402.

Witten, I. H, Bell, T. C., Moffat, A., Nevill-Manning, C. G., Smith, T. G. and Thimbleby, H. 1994a. "Semantic and Generative Models for Lossy Text Compression", The Computer Journal. 37 (2), 83-87.

Witten, I. H, Bell T. C., Moffat A., Nevill-Manning C. G., Smith T. G. and Thimbleby H. 1994b. "Textual Image Compression: 2-Stage Lossy/Lossless Encoding of Textual Images", **Proceedings of the IEEE**. 82 (6), 878-888.

Ye, Y. and Cosman, P. 2001. "Dictionary Design for Text Image Compression with JBIG2", **Proceedings of IEEE for Image Processing**. V. (10), 818-828.

Ye, Y. and Cosman, P. 2003. "Fast and Memory Efficient Text Image Compression with JBIG2", **Proceedings of IEEE for Image Processing.** V. (10), 944-956.

Zhou, J. and Chen, P. 2009. "Generalized Discrete Cosine Transform", **PACCS 2009**. 449-452.