Semi-Lossless Text Compression: a Case Study

BRUNO CARPENTIERI
Dipartimento di Informatica
Università di Salerno
Italy
bc@dia.unisa.it

Abstract: - Text compression is generally considered only as lossless compression. Kaufman and Klein in [1] introduce the idea of semi-lossless text compression: the decompressed text will not be identical to the original text, but, just as for a decompressed JPEG image of good quality that is not identical to the original but can be used in the place of the original in many applications, our brain will adjust the data to make it usable and understandable. In this paper we experiment with semi-lossless compression on a case study of small text files in Italian language.

Key-Words: - Text Compression, Lossless Compression, Semi-Lossless Compression.

1 Introduction

The following text circulated on the internet in September 2003:

"Aoccdrnig to rscheearch at Cmabrigde Uinervtisy, it deosn't mttaer what oredr the ltteers of a wrod are in; the olny iprmoetnt tihng is taht the frist and lsat ltteer be at the rghit pclae. The rset can be a total mses and you can sitll raed it wouthit a porbelm. Tihs is bcuseae the huamn mnid deos not raed ervey lteter by istlef, but the wrod as a wlohe."

The interesting thing that comes out from this example and from similar texts that are easily available on the internet is that we are capable to understand written texts even if there are many errors or typos. In this paper we are interested in the compression of written text files.

Data compression is the coding of data to minimize its representation. Compression is motivated by the economic and logistic needs to save space in storage media and to save bandwidth in communication.

On the downside, compressed data must be decompressed to be used, and this extra processing may be detrimental to some applications.

Data compression is generally called *lossless* if the reconstructed data is identical to the original; otherwise, it is called *lossy* compression (also irreversible or noisy).

Text compression is generally considered only as lossless compression.

Kaufman and Klein in [1] introduce the idea of *semi-lossless text compression*: the decompressed text will not be identical to the original text, but, just as for a decompressed JPEG image of good quality that is not identical to the original but can be used in the place of the original in many applications, our brain will adjust the data to make it usable and understandable.

Kaufman and Klein in [1] propose to use this semi losslessly compressed text in applications in which the correct spelling is not important, such as short email messages or SMS data exchanged through cellular phones or via internet applications like WhatsApp or others.

In this paper we experiment with semi-lossless compression on a case study of small text files in Italian language.

This paper is organized as follows: the next section outlines the case study formulation. Section 3 presents our experimental results and Section 4 our conclusions.

2 A Case Study

To explore the potentialities of semi-lossless text compression we have experimented a number of well known text compression algorithms (Huffman coding [2], Gzip [3], Bzip [4]) on a test set of five small text files in the Italian language. The files are of different sizes but they are all small files, because of the hypothesis in [1] that semi-lossless compression could be useful for small text messages or small files.

Each word in the text set has been rearranged accordingly with the example shown in the previous section: the first and the last letter in the word are in the right place and the inner parts of the word are rearranged.

2.1 Rearranging the letters in a word

There are several ways to adjust (or to mess up the words by leaving n place the first and last character of each word.

The first solution could be to write the inner letters in a word in *random* order. Another solution could be to organize these letters in *alphabetical* order. A third possibility is to arrange the characters by *frequency*: the distribution of character in the Italian language is well known, and it is possible to sort the letters in their of frequency order from the most frequent to the least frequent (E, A, I, O, N and so on ...).

A last approach may be to try to group the characters based on the *probability* of a particular letter to appear after another. In our experiments we have tried all four approaches.

3 Experimental Results

We have experimented the four ways of rearranging the letters in a word on the set data composed of five small text files in the Italian language.

From the point of view of readability, all the resulting files are enough readable, with a clear slow down in the reader's speed in understanding the words.

We have compressed the resulting files by using Huffman coding, Gzip, and Bzip.

Rearrangement	*Input size*	*Compressed size*
Original	4242 bytes	2562 bytes
Random	4242 bytes	2562 bytes
Alphabetic	4242 bytes	2562 bytes
Probabilistic	4242 bytes	2559 bytes
Frequency	4242 bytes	2539 bytes

Table 1: Huffman coding file F1

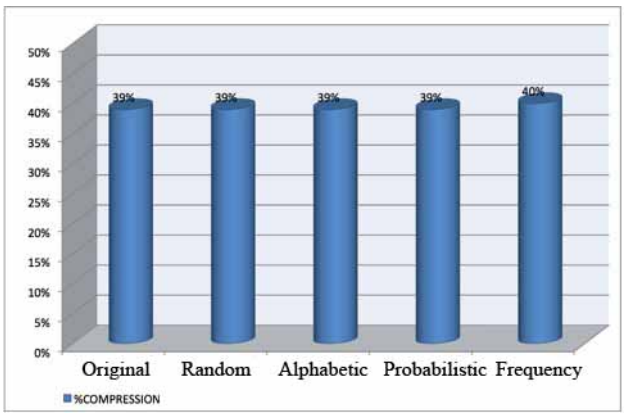

Figure 1: Huffman coding file F1

Tables 1, 2, and 3 show the results obtained by compressing the first file F1.

Table 1 describes on each line the results obtained compressing via Huffman coding the file F1 where the inner letters of each word, have been rearranged, respectively, with the *Random*, *Alphabetic*, *Probabilistic* or *Frequency* rearrangement methods described in the previous section.

The first line (*Original*) refers to the original file in which no mess up with the inner letters has been done.

Figures 1, 2, and 3 are instead a graphical representation of the compression results obtained by Huffman coding, Gzip and Bzip on the file F1.

The compression, for each compressed file, is shown as a percentage of the dimensions of the original file .

Table 4, Table 5 and Table 6 show the results obtained on file F2.

Table 7, Table 8 and Table 9 show the results obtained on file F3.

The figures from Figure 4 to Figure 9 are the graphical representation of the compression results obtained by Huffman coding, Gzip and Bzip on the test files F2 and F3.

Rearrangement	*Input size*	*Compressed size*
Original	4242 bytes	1785 bytes
Random	4242 bytes	2381 bytes
Alphabetic	4242 bytes	1839 bytes
Probabilistic	4242 bytes	1844 bytes
Frequency	4242 bytes	1863 bytes

Table 2: Gzip coding file F1

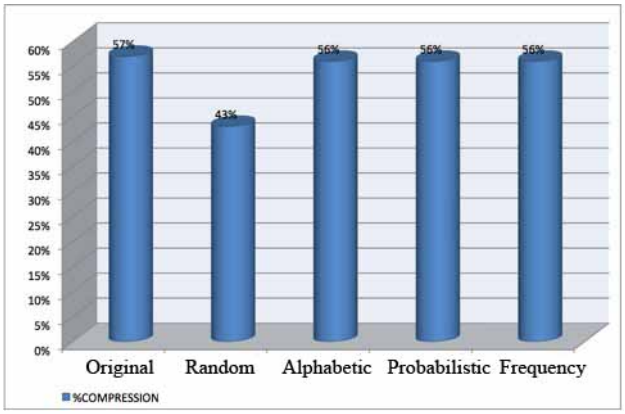

Figure 2: Gzip coding file F1

Rearrangement	*Input size*	*Compressed size*
Original	4242 bytes	1716 bytes
Random	4242 bytes	2308 bytes
Alphabetic	4242 bytes	1723 bytes
Probabilistic	4242 bytes	1774 bytes
Frequency	4242 bytes	1814 bytes

Table 3: Bzip coding file F1

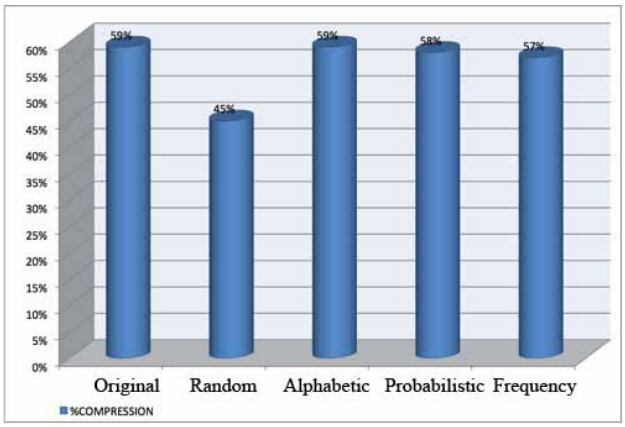

Figure 3: Bzip coding file F1

Rearrangement	*Input size*	*Compressed size*
Original	46888 bytes	28228 bytes
Random	46888 bytes	28228 bytes
Alphabetic	46888 bytes	28228 bytes
Probabilistic	46888 bytes	28098 bytes
Frequency	46888 bytes	27672 bytes

Table 4: Huffman coding file F2

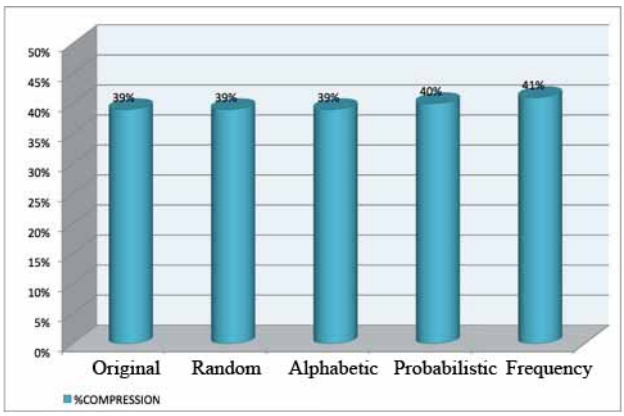

Figure 4: Huffman coding file F2

Rearrangement	*Input size*	*Compressed size*
Original	46888 bytes	4885 bytes
Random	46888 bytes	22037 bytes
Alphabetic	46888 bytes	5042 bytes
Probabilistic	46888 bytes	5050 bytes
Frequency	46888 bytes	5099 bytes

Table 5: Gzip coding file F2

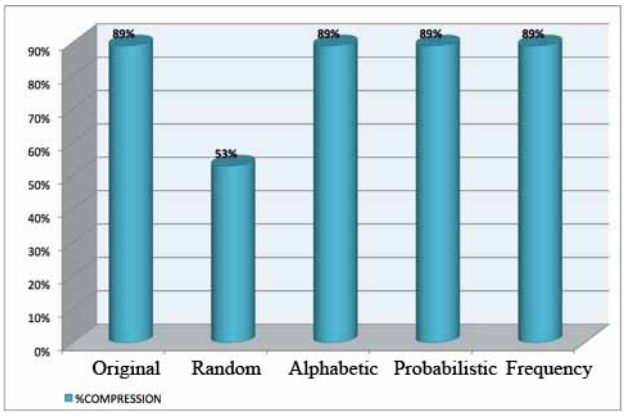

Figure 5: Gzip coding file F2

Rearrangement	*Input size*	*Compressed size*
Original	46888 bytes	5994 bytes
Random	46888 bytes	20082 bytes
Alphabetic	46888 bytes	6086 bytes
Probabilistic	46888 bytes	6242 bytes
Frequency	46888 bytes	6218 bytes

Table 6: Bzip coding file F2

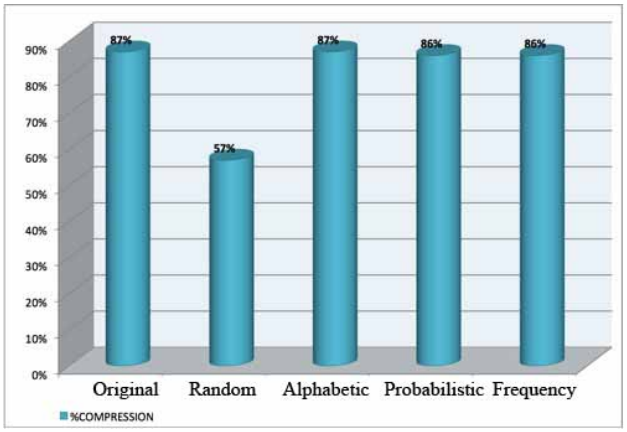

Figure 6: Bzip coding file F2

Rearrangement	*Input size*	*Compressed size*
Original	1037 bytes	689 bytes
Random	1037 bytes	689 bytes
Alphabetic	1037 bytes	689 bytes
Probabilistic	1037 bytes	689 bytes
Frequency	1037 bytes	678 bytes

Table 7: Huffman coding file F4

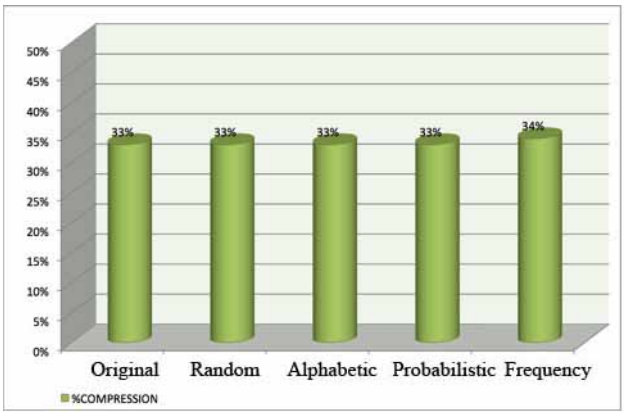

Figure 7: Huffman coding file F3

Rearrangement	*Input size*	*Compressed size*
Original	1037 bytes	526 bytes
Random	1037 bytes	637 bytes
Alphabetic	1037 bytes	536 bytes
Probabilistic	1037 bytes	540 bytes
Frequency	1037 bytes	541 bytes

Table 8: Gzip coding file F3

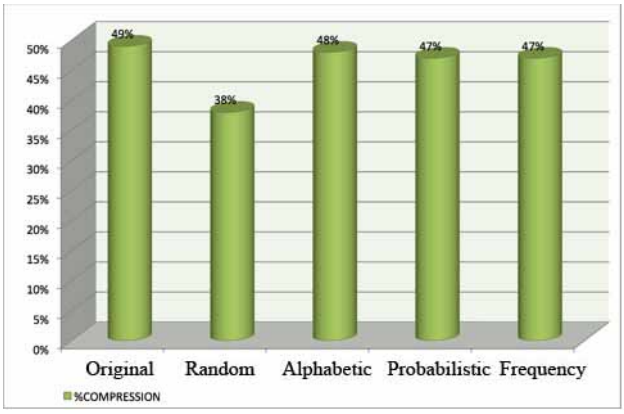

Table 8: Gzip coding file F3

Rearrangement	*Input size*	*Compressed size*
Original	1037 bytes	555 bytes
Random	1037 bytes	659 bytes
Alphabetic	1037 bytes	558 bytes
Probabilistic	1037 bytes	563 bytes
Frequency	1037 bytes	581 bytes

Table 9: Bzip coding file F3

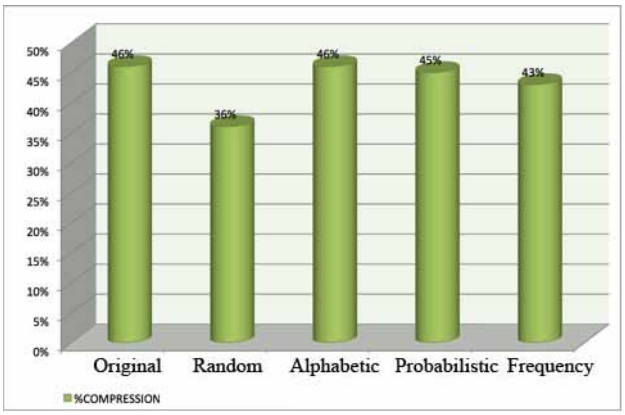

Figure 9: Bzip coding file F3

Similar results are available also for file F4 (that is the larger file) and for file F5. Those results are omitted here for brevity but they are consistent with the ones shown.

Table 7 summarizes the combined results for all the five files.

Rearrangement	*Huffman*	*Gzip*	*Bzip*
Original	235364	18854	21751
Random	235364	31434	40997
Alphabetic	235364	19401	22055
Probabilistic	234285	19438	22542
Frequency	230750	19607	22493

Table 10: Compression results on the test data set (in bytes)

As it is shown in tables 1, 4, 7 and 10, the *Probabilistic* and the *Frequency* rearrangements slightly improve the performances of Huffman Coding.

On the other side, for Huffman coding the performances obtained by *Random* and *Alphabetic* are identical to the performance of *Original*.

This is due to the inner nature of Huffman coding that has a statistical core that is sensible to a reordering that is based on frequency and that is not sensible to randomness.

Tables 2, 5, 8, and 7 show that Gzip does not take advantage of the rearrangements. The same is for Bzip.

This is probably because both compression algorithms are not based on statistical coding and therefore the rearrangements we have applied are not an improvement with respect to the original situation.

This does not necessarily mean that semi-lossless compression does not work for those algorithms, but only that the specific rearrangements we have tried are not suitable to improve those algorithms.

It is interesting to note that for both Gzip and Bzip the *Random* rearrangement gives the worst performance, because it breaks the correlation between parts of the word and reduce the effectivity of the two algorithms.

From the data compression point of view, the most effective algorithm in this case study is Gzip: because of the small length of the input files Bzip (that is generally more performing than Gzip), does not achieve the compression that often gets on larger files.

4 Conclusion

In semi-lossless text compression the decompressed text will not be identical to the original text, but our brain will adjust the text data to make it usable and understandable.

In this paper we have experimented with semi-lossless compression on a case study of five small text files in Italian language.

We have reported the experimental results for Huffman coding, Gzip and Bzip on the files where the inner letters of each word have been rearranged, respectively, with the *Random*, *Alphabetic*, *Probabilistic* or *Frequency* rearrangement methods.

All the resulting files were enough readable: the text was still fully understandable, with a clear slow down in the reader's speed in understanding the words.

The results we have obtained show a slight compression improvement when using Huffman Coding and *Probabilistic* or *Frequency* rearrangements, but no improvement (actually a worsening) when using Gzip or Bzip.

This does not necessarily mean that semi-lossless compression does not work for those algorithms, but only that the specific rearrangements we have tried are not suitable to improve those algorithms.

Future research will involve testing on different rearrangement methods that might be more suited for non-statistical compression algorithms and also testing with files in different languages.

Acknowledgements

I wish to thank my students: Marco Lettieri, Alessia D'Andria, Antonella Masi, Antonella Palladino and Isabella Ruggiero that have conducted a preliminary set of experiments on semi-lossless compression of Italian texts.

References:

[1] Y. Kaufman and S. T. Klein, "Semi-lossless text compression", *International Journal of Foundations of Computer Science*, Vol. 16, No. 6, 2005, pp. 1167-1178.

[2] I. H. Witten, A. Moffat, and T. Bell, *Managing Gigabytes*. NY: Van Nostrand Reinhold, 1994.

[3] *The Gzip home page*, www.gzip.org.

[4] *Bzip2: home*, www.bzip.org.