Asian Journal of Information Technology 5(6): 578-583, 2006
© Medwell Online, 2006

Entropy Reduction of Arabic Text Files

[1]Abdel-Rahman M. Jaradat, [2]Mansour I. Irshid and [3]Talha T. Nassar
[1]Department of Electrical and Computer Engineering, University of Sharjah, P.O. Box 27272 United Arab Emirates
[2]Department of Electrical Engineering, Jordan University of Science and Technology
[3]Department of Computer Science, Jordan University of Science and Technology, Irbid, 22110 Jordan

Abstract: In this study we consider several approaches for reducing the entropy as applied to Arabic text files. First we apply the source mapping technique to produce a binary file which we tested the performance on such mapping using both Huffman and Arithmetic coding. Next we implemented file splitting technique for the reduction of the nth-order entropy of text files which was proposed by the authors. The technique is based on splitting the binary file into several subfiles each contains one or more bits from each codeword of the mapped binary file. The resulting subfiles were used to achieve better compression ratios when conventional compression techniques are applied to these subfiles individually and on a bit-wise basis rather than on character-wise basis. The technique was applied on Arabic text files and it was found that considerable reduction in their entropy was achieved. Applying Huffman as well as arithmetic coding on the binary encoded files showed promising results.

Key words: Arabic text, entropy, text file compression

INTRODUCTION

In conventional computer-based text compression techniques, the characters in the alphabet of the input text file are transformed into an arbitrary fixed-length binary code such as the standard 8-bit ASCII code and then the compression algorithm manipulates the resulting binary text file on a character-wise basis (byte-wise basis)[1,2]. As it is well known, high compression ratios can be achieved by working on higher-order extensions of the original text source by finding the probabilities of the occurrence of all the combinations resulting from two or more characters. But increasing the order of extension from the 0th order to the nth order requires increasing the number of combinations to (256) n + 1 which is (256) 2 = 65536 combinations for the 1st order and (256) 3 = 16777216 combinations for 2nd order compared to 256 combinations for the 0^{th} order. This mean that the higher-order extended sources requires from the compression algorithms to deal with a very large number of combinations which in turns requires very large memories and very long execution times. On the other hand, if a bit-wise source extension is used, the number of combinations can be increased in smaller jumps as we increase the order of extensions where the number of combinations is equal to 2n + 1 for the nth order extended source, i.e. 2,4,8,16,32, ... etc. It is found that a very limited research has been done on text compression based on a bit-wise basis[3]. One method based on bit-wise approach has been proposed in[4] where a non-ASCII codewords are assigned to the characters of the Arabic text so that it would be more effective for the bit-wise run-length compression algorithm. In previous work of the authors, an efficient source encoding technique is proposed which is based on mapping the non-binary information sources with a large alphabet onto an equivalent binary source using weighted fixed-length code assignments[4,5]. The weighted codes are chosen such that the entropy of the resulting binary source multiplied by the code length is made as close as possible to that of the original non-binary source. It is found that a large saving in complexity, execution time and memory size is achieved when the commonly-used source encoding algorithms are applied to the bit-wise nth-order extension of the resulting binary source. This saving is due to the large reduction in the number of symbols in the alphabet of the new extended binary source where bit extensions of 1, 2, 3, 4 ... etc can be used in the bit-wise procedure instead of bit extension of 8, 16, 24 ... in the character-wise procedure. In this study, we apply the mapping technique on the original text file and transform it into a non-ASCII binary file using a new codeword assignment method[6]. The

Corresponding Author: Abdel-Rahman M. Jaradat, Department of Electrical and Computer Engineering, University of Sharjah, P.O. Box 27272 United Arab Emirates

resulting binary file is split into several binary subfiles each contains one or more bits from each codeword of the original binary file. The statistical properties of the split files were studied and it was reported that they reflect the statistical properties of the original text file which is not the case when the ASCII code is used as mapper[6]. The interesting properties of the resulting split files can be used to achieve better compression ratios when conventional compression techniques are applied to these files individually and on a bit-wise basis rather than on character-wise basis. The source mapping method and the bit-wise nth order entropy of an Arabic text file is found using the proposed mapping method and it is compared with that resulting from using ASCII code mapping. Both Huffman and Arithmetic coding were implemented and applied to the mapped binary file. The file splitting technique[6] and the bit-wise nth-order entropy of an Arabic text file is found for the individual split files. Conclusions are given in section 4.

SOURCE MAPPING

In previous work, the authors successfully applied a new source mapping technique to compress text files using conventional compression techniques but working on the binary file on a bit-wise basis rather than on a character-wise basis[4,5] and recently in[7] it was extended to the compression of multimedia files. The mapping technique is based on mapping each character in the original text file into an 8-bit codeword based on the frequency of occurrence of the original character. The mapping is done by arranging the different characters of the alphabet of the given text file in a descending order of their probabilities of occurrence and then the codewords are given to the different characters such that each codeword is the 8 bits binary representation of the position of that corresponding character in the list. The most frequent character is represented by an 8-bit binary sequence of all zeros (which is the binary representation of decimal zero), the second most frequent character is represented by all zeros but a one in the LSB of the binary sequence (which is the binary representation of decimal one) and so on. A source mapper for the Arabic language (256 symbols) is designed according to the proposed mapping rule and the probabilities and the assigned codewords for the different characters are shown in Table 1 (a sample of the characters is shown in the table). Let the original information source S has an alphabet of M = 2N symbols with probabilities {p0, p1, p2, …, pM-1} and let the equivalent binary source B has a probability of P0 for symbol 0 and a probability of P1 = 1- P0 for symbol 1. The 0th-order entropy of the original text file is

Table 1: The probabilities and the proposed code assignments for a sample of the Arabic text alphabet

Character	Probability	Assigned code
Space	0.178	11111111
ا (alef)	0.112	11111110
ل (lam)	0.0863	11111101
ي (ya)	0.0578	11111011
م (meem)	0.0414	11110111
ن (noon)	0.0441	11101111
و (waw)	0.0414	11011111
ت (ta)	0.0373	10111111
ر (ra)	0.0341	01111111
ع (ain)	0.0265	11111100

calculated on a character-wise basis using the following well-known entropy equation[8]:

$$H_c = -\sum_{i=0}^{255} p_i \log p_i \tag{1}$$

When substituting the probabilities of the different characters of the Arabic language given in Table I in (1), it is found that the entropy of the original text source is 4.74 bits/character. The 0th entropy of the binary file which results from mapping the original source using the above proposed mapping method can be found by calculating the probabilities of symbol 0 and symbol 1 of the resulting binary source using the following equation:

$$P_0 = 0.8,\ \ P_1 = 0.2 \tag{2}$$

where the first term gives the probability of symbol 0 in the LSB and the second term gives its probability in the next bit position and so on. By substituting the probabilities of the original text symbols in equation 2, the probabilities of symbol 0 and symbol 1 of the resulting binary file are found to be 0.85 and 0.15, respectively. The entropy of the resulting 0th-order binary source is found by substituting the probabilities of 0 and 1 in the following binary entropy equation:

$$H(B) = -P_0 \log_2 P_0 - P_1 \log_2 P_1, \tag{3}$$

and it is found to be 0.609 bits/symbol. The entropy for n bits of the binary file is n times that of its 0th order entropy, i.e., 0.6n bits/symbol. The entropy of the 8 bits of the binary source which is equivalent to one character of the original file is found to be 4.89 bits/character compared to 4.47 bits/character calculated on character-basis for the original text file.

For comparison purposes, the entropy of the binary source resulting from using the standard ASCII code mapper is calculated and it is found that the probabilities of symbol 1 and symbol 0 are 0.44 and 0.56, respectively and the entropy of the 0th-order binary source is 0.99

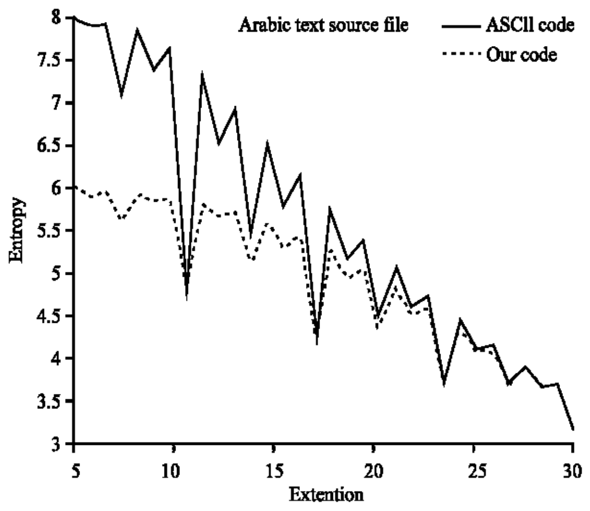

Fig. 1: The nth -order entropy of the Arabic text file for the proposed Code and the ASCII Code for various bit extensions

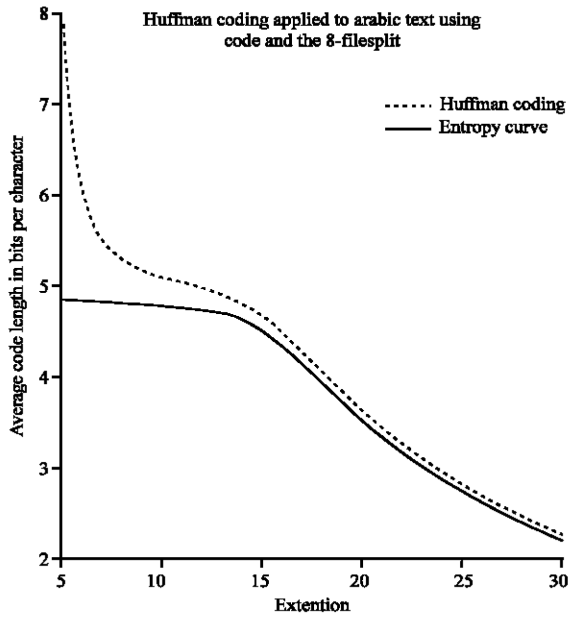

Fig. 2: Average code length using Huffman coding and the entropy curve for various bit extensions when applied to the binary encoded split files of Arabic text

bits/symbol and the entropy of the 8 bits which is equivalent to one character is 7.9 bits/symbol which is much greater than the entropy of the original source. This comparison indicates that bit-wise compression technique cannot achieve any compression when applied to binary sources using ASCII code or any other random code.

The nth-order entropy for the binary source resulting from the proposed mapping method and that resulting from the ASCII code is calculated for various values of bit extensions where n takes values from 1 to 32 where a computer program is written to determine these entropies by finding the probabilities of occurrence of the 2n bit combinations in the binary file. Fig. 1 shows the nth entropy as function of bit extensions for an Arabic text file using the proposed source mapper and the ASCII code mapper. As it is shown from this Fig, the two mappers give the same nth entropy for the 8, 16, 24 and 32 bit extensions and this result is as expected since these extensions correspond to the 0th, 1st, 2nd and 3rd extensions based on character-wise entropy calculations which are independent of codeword assignments. For bit extensions which are not multiples of number 8, the entropy of the ASCII code is quite larger than that of the proposed code specially for lower extension values but the difference decreases as the bit extension increases. For bit extensions lower than eight, there is a saving of approximately two bits in the value of the entropy when using the proposed mapping method. This saving in entropy at low bit extensions can help in designing simple compression techniques with large saving in complexity, execution time and memory requirements[4-6].

Huffman coding was implemented and tested on the binary encoded file at various bit extensions. At an extension of 1 bit, Huffman showed no compression. Fig. 2 shows the average code length per character achieved by Huffman compression algorithm when applied to the binary encoded split files as compared

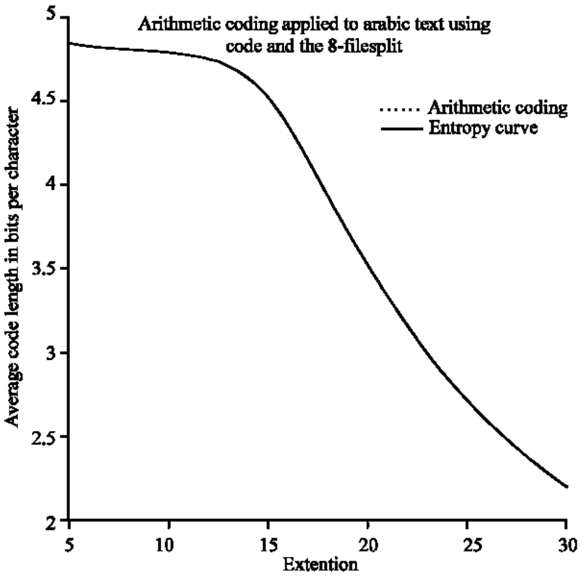

Fig. 3: Average code length using arithmetic coding and the entropy curve for various bit extensions when applied to the binary encoded split files of Arabic text.

with the minimum entropy curve. At higher bit extensions, Huffman achieved values comparable to that of the entropy.

Compression via arithmetic coding can achieve the entropy limit. Fig. 3 shows the comparison of the results achieved via arithmetic coding as compared to the entropy when applied to the binary encoded text file. The average code length achieved by arithmetic coding was found to be identical to the value of entropy at every bit extension.

FILE SPLITTING

To achieve more compression by reducing the nth order entropy of the text file by splitting the mapped binary file into several subfiles and the total entropy of the file is found by determining the entropy of the subfiles and summing them up. To achieve compression ratios below that of conventional methods, the compression algorithm has to be done on the subfiles rather than on the whole binary file[6]. By examining the variation in the probabilities of the binary symbols in the different bit position, we find that the probabilities of symbol 1 and symbol 0 in the least significant bit of the codewords are almost equal while the difference between these two probabilities start to increase as we move towards the most significant bit where it is almost one for symbol 0 and almost zero for symbol 1. This means that the entropy of the LSB is nearly equal one while it is nearly equal zero for the MSB and has values between zero and one for the rest of the bit positions. This interesting fact leads us to device a technique which utilizes these wide variations in the entropy of the different bit positions in the binary text file.

The technique is based on splitting the binary file into two, four or eight subfiles each of them contains equal parts of the original file based on dividing the bits of the codewords in specified manner.

The probabilities of symbol 0 and symbol 1 and the corresponding entropies for the various bit positions are shown in Table 2. The total 0th -order entropy HT of the mapped binary file when it is split into 8 subfiles is found to be 4.841 bits/character. This entropy value is almost equal to the 0th-order entropy of the original text file calculated on character-wise basis which is 4.782 bits/character. This is a very encouraging result since we can implement simple compression techniques working on a bit-wise basis and achieving compression ratios comparable to that working on character-wise basis but with much lower symbol combinations. It is obvious from Table 2 that the four least significant bits have the largest contribution to the overall entropy of the mapped binary file while the four most significant bits have the lowest contribution where the contribution from the four LSB is 3.732 bits/character and from the four MSB is 1.1 bits/character. Since it is very difficult to determine the nth order entropy of the subfiles mathematically, a computer program is written to determine these entropies by finding the probabilities of the 2n bit combinations in the each subfile and then substituting the results in the entropy equation. The effective entropy of the whole split binary file is equal to the sum of the individual entropies of the subfiles. Fig. 4 shows the overall entropy of the mapped binary file as function of bit extensions for one, two, four and eight subfiles where the case of one subfiles is that of the un-split mapped binary file. It is obvious from this Fig. that the three cases of split files give overall entropy which is much less than that of the un-split case except for bit extensions which are multiple of number eight which is the length of character codeword. Also it is obvious that as the number of subfiles increases the entropy curve becomes more smother. For bit extensions less than eight, the 8 subfile case gives the lowest entropy which is nearly constant and it is almost

Table 2: The probabilities and the entropies of the different bit positions of the proposed codewords in an Arabic text file

A. Bit	P_O	P_1	B. Entropy
7	1.000	0.000	0.0000
6	0.999	0.001	0.0097
5	0.924	0.076	0.3869
4	0.804	0.195	0.7127
3	0.726	0.273	0.8464
2	0.664	0.336	0.9213
1	0.586	0.414	0.9787
0	0.571	0.429	0.9855

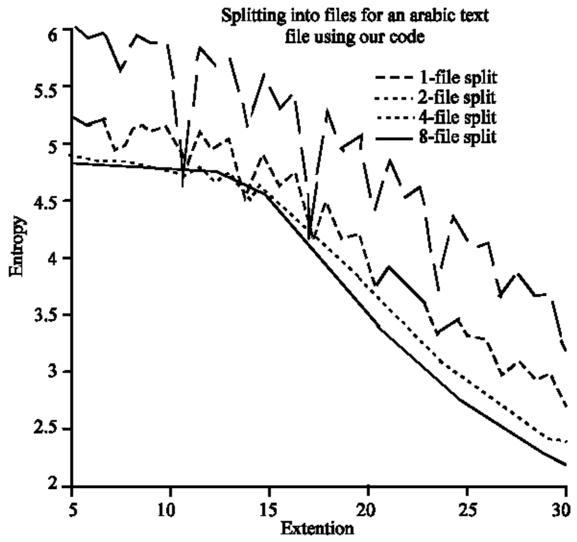

Fig. 4: The n^{th} -order entropy of the split Arabic text file as function of bit extensions using the proposed Code for the various splitting cases

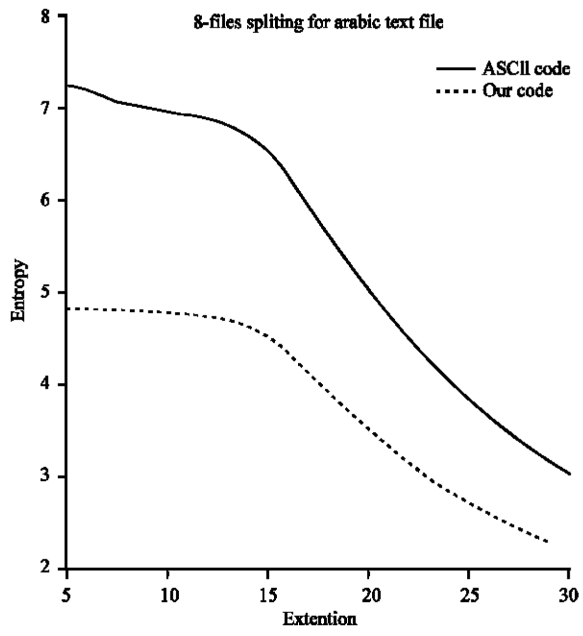

Fig. 5: The n^{th} -order entropy of the split Arabic text file as function of bit extensions using the proposed Code and the ASCII code for the 8 subfile splitting cases

equal to the 0th order character-wise entropy of the original text file and this interesting property can be utilized to device a simple method for determining the 0th order character-wise entropy for sources having large number of alphabets. The two-subfile case has the best entropy properties for values of bit extension lying between 8 and 16 where its entropy at the 8 and 16 bit extensions is less than that of the corresponding 0th and the 1st character-wise entropy of the original text file. Moreover, the entropy of the two-subfile case at the 12 bit extension is equal to 4 bit/character compared to 4.75 bit/character for the un-split binary file and compared to 4.74 bit/character for the 0th order character-wise entropy and compared to 34.42 bit/character for the 1st order character-wise entropy. This means that a good compression can be achieved at the 12 bit extension of the two-subfile case where the compression algorithm deals with 212 = 4096 binary combinations compare with 216 = 65536 binary combinations required for the 16 bit extension (1st order character-wise extension). The four-subfile case has the best entropy properties for values of bit extension greater than 16 where it has the least values of entropy compared with the other cases. This case has an entropy of 3 bit/character at the 24 bit extension compared with 3.76 bit/character for the corresponding 2nd order character-wise extension and an entropy of 2.3 bit/character at the 32 bit extension compared with 3.05 bit/character for the corresponding 3rd order character-wise extension. The reduction in the entropy of split file below that of the original text file is due to the fact that when extending the different subfiles by n bits, the actual extension in the original text file is 2n, 4n and 8n bits for the two, four and eight subfiles cases. For comparison purposes, Fig. 5 shows the nth-order entropy of the split Arabic text file as function of bit extensions using the proposed Code and the ASCII code for the 8 subfile splitting cases which shows that this file splitting technique works only with specific codeword assignment methods similar to that proposed in this study.

CONCLUSION

In this study, we implemented source mapping on Arabic text and demonstrated the achievements in lowering the entropy of the mapped text file using both Huffman and Arithmetic coding techniques. Next file splitting technique was applied on the mapped binary file. The nth-order entropy of these subfiles are determined and it is found that their sum is less than that of the original text file for the same values of extensions. The reduction in the entropy of the resulting subfiles can be exploited to device compression algorithms with better compression ratios. This can be done by applying the conventional compression techniques to the subfiles individually and on a bit-wise basis rather than on character-wise basis. By applying the technique to the same text file but with ASCII mapping, it is found that this file splitting technique works only with specific codeword assignment methods similar to that proposed in[6].

REFERENCES

1. Bell, T.C., J.G. Cleary and I.H. Witten, 1990. Text Compression, Prentice-Hall, Englewood cliffs NJ.
2. Salomon, D., 2000. Data Compression: The Complete Reference, Springer-Verlag, New York.
3. Lynch, M.F., 1973. Compression of bibliographic files using an adaptation of run-length coding, Information Storage and Retrieval, 9, pp: 207-214.
4. Elabdalla, A.M. and M.I. Irshid, 2001. An efficient bitwise Huffman coding technique based on source mapping Computers and Electrical Engineering, 27, pp: 265-272.
5. Jaradat, A.M. and M.I. Irshid, 2001. A simple binary run-length compression technique for non-binary sources based on source mapping Active and Passive Electronic Components, pp: 211-221.

6. Jaradat, A.M., M.I. Irshid and T.T. Nassar, 2006. A File Splitting Technique for Reducing the entropy of Text Files, accepted for publication, Intl. J. Inform. Tech.
7. Sharieh, A.A., 2004. Enhancement of Huffman Coding for the Compression of Multimedia Files, Intl. J. Inform. Tech., pp: 211-213.
8. Cover, T.M. and J.A. Thomas, 1991. Elements of Information Theory, New York: John Wiley and Sons.