Dynamic with Dictionary Technique for Arabic Text Compression

Fatima Thaher Ahmad Aburomman

ABSTRACT
In this research paper we build a new, reliable, and sufficient algorithm for Arabic text language. The proposed algorithm should combine the features of the Huffman and Lempel Ziv algorithms, and is expected be able to reduce the general compression ratio .

Our approach is different from Huffman algorithm in the sense that it assigns codes to n-gram symbols where n is a positive integer that is greater than or equal to one. Compared to Huffman algorithm, which assigns a code to each symbol individually, our approach is expected to assign codes to symbols in average.

Our approach is different from Lempel Ziv algorithm in the sense that the size of dictionary that we build does not grow in an uncontrolled manner. The size of the dictionary is fixed and its size can be expected prior to process the text files that are to be compressed. This is because the size of each word in the dictionary we build is fixed and is equal to n. So for example, given that the number of different symbols in the text file at hand is m and that n is 2, the total number of entries in the dictionary that we propose to build will be m*m in the worst case.

Keywords
Data compression, Text compression, Arabic Text File data compression.

1. INTRODUCTION
Different types of data are produced, stored, transmitted or exchanged. For example, mobile phone messaging is widely used nowadays to exchange data. Transferring data among mobile devices or computers can be time consuming. Moreover, storing such enormous amount of data can be prohibitively expensive (Omer and Khatatneh, 2010). For these reasons, there has been a demand to reduce the amount of data before saving or transferring it. This process of reduction in the amount of data is called Data Compression.

Data compression is the process of representing a document by using less space (Omer and Khatatneh, 2010).

1.1 Data Decompression
The word compression implies two processes: compression process and decompression process (ibid, 2006). In many papers, decompression process is not mentioned as compression process, because it is obvious and easy to be derived from the compressed file.

According to the efficiency of data that is to be decompressed, the compression techniques can be classified to two main categories:

- Lossless (reversible) data compression
- Lossy (irreversible) data compression

Data compression can be lossless if the decompressed data is the same as the original data before being compressed; this is like in Text compression, and medical image compression. (Al-Daoud, 2010; Ida Mengyi Pu, 2006)

The data compression is lossy if the decompressed data loses parts or some details of its original content during the compression process. Sometimes these details can't be easily noticeable/visible, such as in multimedia images and videos (ibid, 2006).

1.2 Text Compression
Text file compression is the process of transforming string of characters to another new shorter one by using less space and without losing the data content (Hjouj Btoush et. al, 2008; Al-Daoud, 2010; Omer and Khatatneh, 2010). There are many text compression techniques, but the preferred technique over the others can be based on the following criteria: (Musa et.al, 2010; Hjouj Btoush et. al, 2008)

- Compression speed or time consumed to complete the compression process.
- Amount of memory size consumed to complete the compression process.
- The implementation complexity (ease of implementing).
- The compression ratio bit/char.

All of these criteria, joined together, play a major role in deciding which compression algorithm is the best. (Elabdalla and Irshid, 2001).

Thus, in the last few years, many researches/studies have been done in the text compression field to develop new efficient text compression techniques that achieve these criteria (ibid, 2001).

1.2.1 Text Compression Techniques
Text compression technique can be divided into two main methods: (Musa et.al, 2010)

- Static text compression.
- Dynamic (dictionary) text compression.

Each of these two methods has its own compression algorithm and its requirements.

1.2.1.1 Static Text Compression
Using Huffman algorithm in the static text compression approach, a given string is mapped to a predefined codeword (dictionary), so it is classified as file-independent, based on the frequency of the given string in the original text file. The strings with high frequency are mapped to short codeword, while small frequency strings are mapped to long codeword. (Al-Daoud, 2010; Omer and Khatatneh, 2010).

1.2.1.2 Dynamic Text Compression

In the dynamic text compression method the compression process is performed by mapping the given string to codeword that changes from text file to another (file-dependent), Lempel Ziv algorithm is used in this method (Musa et.al, 2010, Al-Daoud, 2010, Omer and Khatatneh, 2010).

1.2.2 Text Compression Algorithms

There are many text compression algorithms studied in computer science. These algorithms are different from each other in terms of efficiency (Hjouj Btoush et. al, 2008).

1.2.2.1 Huffman Algorithm

Huffman algorithm is the most widespread source technique in data compression (Elabdalla and Irshid, 2001). David Huffman invented it in 1952 (Musa et.al, 2010). The idea began when Professor Rober M. Fano assigned a problem of finding the most efficient binary code in a term paper. Huffman solved this problem based on the frequency sorted binary tree, which is the most efficient method (Wordiq, 2010). Later on, Huffman developed his method by building the bottom up tree instead of his professor method that builds the tree from top to down (Wordiq, 2010).

Huffman algorithm can save from 20% to 90% typical, depending on the characteristics of the data being compressed (Coremen, 2002). It uses a "defined-word" scheme where sequences of input symbols are transmitted using variable length code (Bently et. al, 1986). It is based on building bottom up tree for input symbols and giving short code for the relatively frequent symbols and long code for infrequent (Elabdalla and Irshid, 2001; Ghawanmeh et. al, 2006; Hjouj Btoush et. al, 2008).

However, the drawback of Huffman algorithms is that it does better with small files than large ones (Hjouj Btoush et. al, 2008).

1.2.2.2 LZW Algorithm

Lempel Ziv algorithm is the general algorithm that works on almost any type of data (Hjouj Btoush et. al, 2008). It transmits strings of characters to a single code using fixed length code (ibid, 2008). It is based on building a table of strings and adding every new string of character to this table, and it does better with large files rather than small files (ibid, 2008).

There are many types of data such as text, image, sound, and any combination of all these data such as video. However, in this research we are concerned with textual data compression which is an approach of text compression field (Elabdalla and Irshid, 2001; Hjouj Btoush et. al, 2008).

1.3 Dynamic and Dictionary-based Technique

Moreover, the Arabic language suffers from the absence of efficient algorithms for text compression. The availability of such algorithms is one of the biggest challenges that face the Arabic language in the text compression field.

In this research proposal, we try to build a new, reliable and sufficient algorithm for Arabic text language. The proposed algorithm should combine the features of the Huffman and Lempel Ziv algorithms, and is expected be able to reduce the general compression ratio .

Our approach is different from Huffman algorithm in the sense that it assigns codes to n-gram symbols where n is a positive integer that is greater than, or equal to one. Compared to Huffman algorithm, which assigns a code to each symbol individually, our approach is expected to assign codes to symbols in average.

Our approach is different from Lempel Ziv algorithm in the sense that the size of the dictionary that we have built does not grow in an uncontrolled manner. The size of the dictionary is fixed and its size can be expected prior to processing the text files that are to be compressed. This is because the size of each word in the dictionary we have built is fixed and is equal to n. So for example, given that the number of different symbols in the text file at hand is m and that n is 2, the total number of entries in the dictionary that we propose to build will be m*m in the worst case.

2. LITERATURE REVIEW (RELATED WORK)

In the last few years, a number of research papers/studies were published that cover the text compression for Arabic files. However, in the literature review we will focus only on the most recent available Arabic texts compression researches that have been carried out.

A group of researchers introduced a new technique, based on a dynamic mapping rather than static to minimize the compression ratio of Arabic texts using bitwise Lempel Ziv algorithm (Musa et.al, 2010). This technique is based on mapping each character in the input Arabic Text File (ATF) to its corresponding code word using static or dynamic scheme, then splitting the file into sub files to apply Lempel Ziv algorithm on each sub file that have smaller size than the original file individually. (Musa et.al, 2010).

Below is the abstract model of compression and de-compression process based on the mapping scheme adapted from Musa et.al, 2010.

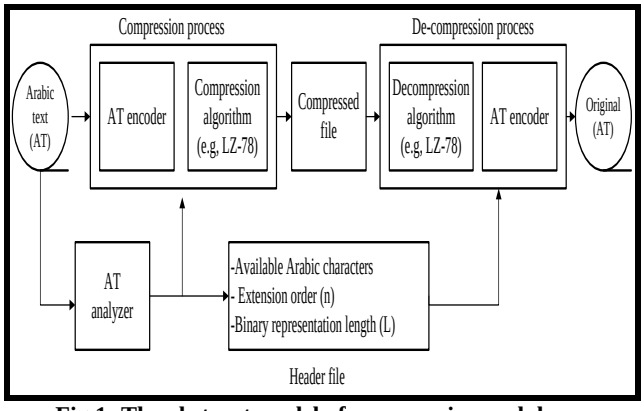

Fig 1: The abstract model of compression and de-compression process based on mapping scheme (Musa et.al, 2010).

Moreover, this research divides its work to two main parts:

1. Preprocess :character mapping schemes
2. Bitwise Lempel Ziv algorithm.

The mapping process is based on that the large character's frequency will take large hamming weight using this rule:

$$\binom{N}{K} = \frac{N!(N-K)!}{K!}$$

In the static scheme, however, each character is given a code word according to the ASCII table, the dictionary table size is equal to the number of characters in the file, but in the dynamic scheme, each character is given a code according to its frequency, so the size of dictionary table will be less than that in the static scheme, and it depends on the frequency of characters in the file and the file size. The dynamic scheme is considered to be time-consuming, because of the time consumed to construct the mapping table; yet, the advantages of using the dynamic scheme over the static scheme according to Musa et. al are: First, compression ratio will be decreased. Second, speed will be higher and finally, this scheme can be used for any type of text files (file-dependent) (Musa et.al, 2010).

The next step is applying Bitwise Lempel Zive compression process, depending on the extension-order (n). In this technique, the extension- order is equal to two.

The compression dictionary is initialized by values equal to (2n), then; the algorithm will read the input stream as block of 2 bits. In addition, check if this block is in the table or not, if it exists, the algorithm reads the second block and checks the existence of these two blocks together, if not it adds this block to the table.

After reading all input stream, the algorithm splits each row into the table into two parts. The second part contains one block, then checks the first part to find out in which row was it, so as to present it as a pair of (numerical value-index, the second block).

The L value indicates the number of rows presented in pairs, now each pair is mapped to its compressed value by presenting the value of (L+ n) in binary.

All this process is presented in the figure below adapted from Musa et.al, 2010

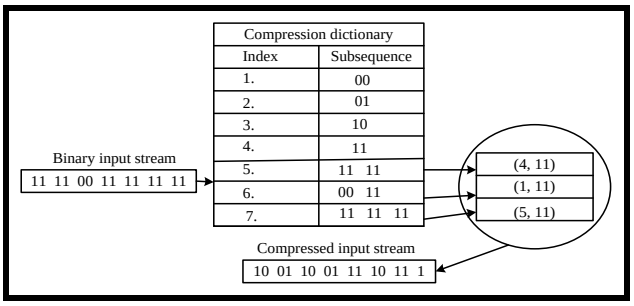

Fig 2: Examples of bitwise LZ-78 compression algorithm with extension-order 2 (Musa et.al, 2010).

This technique was applied on different text files such as " الرحيق المختوم.txt " and " تحفة العروس .txt" . It has achieved a compression ratio within 4.25 to 4.7 bit/characters (Musa et.al, 2010). However, in our proposed study, we will use the dynamic method to mapping a string of data from the source file instead of characters, and we will examine our algorithm on the same text files that were used, as well as any additional text file.

The dynamic Huffman coding was applied on Arabic and English texts, using variable length bit coding (Ghawanmeh et. al, 2006). It splits the task of compression into two processes: the first is encoding process: representing the text message with fewer bits by concatenating the codes of consecutive characters together. Hence, it is easier to decide the start and the end of each code, so as to guarantee that no code would share the prefix of the other (ibid., 2006). The second is decoding process in which reconstructing the original message from the compressed representation is done by binary tree (ibid, 2006). However, in our proposed study we will apply Huffman encoded on the dictionary table itself instead of applying it on each character, which will decrease the time consumed in building and processing the binary tree.

The example below is adapted from this paper, it is applied on an Arabic text, if we have the message "بسم الله الرحمن الرحيم", and then the Huffman data compression will be as shown below:

Table 1: Huffman data compression (Ghawanmeh et. al, 2006)

Symbol	Occurrence	Probability	Codeword
ب	1	1/22	000001
س	1	1/22	000000
م	3	3/22	101
ا	3	3/22	001
ل	4	4/22	01
ه	1	1/22	10000
ر	2	2/22	1001
ح	2	2/22	0001
ن	1	1/22	10001
ي	1	1/22	00001
Space	3	3/22	11
Sum	22	1	

- The required size to store this message is (176) bits.
- The compression ratio is equal to (42.61%)

The tree for this example is illustrated in the figure below:

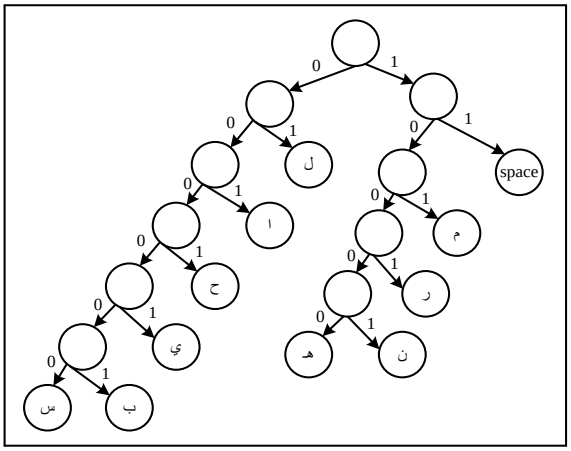

Fig 3: Huffman tree (Ghawanmeh et. al, 2006)

Abdel-Rahman et al, 2006 showed that the entropy of Arabic Text Files (ATF) can be highly reduced if the source text characters are mapped according to their frequency, then the resulting files splitted into several sub files; each with one or more bit. This technique can be exploited to device

compression algorithm with better compression ratio. Moreover, it proved that all Arabic characters could be encoded using 7 bits only, wherein the most Arabic character needs actually only 6 bits (Jaradat, Irshid and Nassar, 2006). This study can be summarized as follow:

1. It starts by mapping each character in the original text to 8-bits codeword based on its occurrences using Huffman and arithmetic coding techniques, the result of this step is a non- ASCII binary file.
2. It split the result binary file into two, four or eight sub-files with equal parts of the original file, these sub-files are used to achieve better compression ratio by applying bit-wise rather than character-wise in each sub-file; individually.

In our proposed algorithm we have a different file size, so splitting it in such cases will not do any better, so we propose to take it as a whole and apply Lempel Ziv algorithm to utilize a table based compression for repeated n-grams strings in the input file, then the table based itself is often a Huffman encoded.

In 2010, Daoud presented a new combined method of root dictionaries and proposed coding scheme, his method is based on decomposing the word to its root and its affix, and then storing it in a dictionary. This dictionary contains 8000 three-character root and 700 four-character root (Al-Daoud, 2010).

Daoud, presented in his paper the Arabic language features that are not taken by the Arabic compression algorithms, these features are:

- Highly inflected and derived language.
- Syntactical.
- Phonological.
- Morphological.
- Diacritical, which called "Tashkeel".
- Semantical.

However, Daoud have manipulated with the Arabic morphological feature, by extracting it to increase the compression rate and decrease the computational cost, he explained that all words in Arabic are derived from a list of roots.

In our proposed algorithm, we deal with these types of Arabic text such as:

- Text without diacritical text.
- Partially diacritical text, such as a text contains parts of Al- Qur'an verses.

Another study was done in 2010 in which the researchers described a new technique for Arabic short text messages sent by mobile phone. The study used different schemes from different sources: such as, splitting files and hybrid dynamic coding, then it applied the statistical Huffman tree in a method similar to that used for fax machine code. The long codes are given to the short frequency characters, and the long frequency takes the short code (Omer and Khatatneh, 2010).

3. DYNAMIC AND DICTIONARY-BASED TECHNIQUE

3.1 Introduction

By analyzing this technique, the current algorithm provides a new technique for compressing text files and combining the features of the two algorithms (Huffman and Lempel ZIV).

Our improvement for these two algorithms was done in two steps:

1. Change the number of grouped characters, by allowing the program to read (n) character at a time, which we called (n-gram) where n is appositive integer number of characters greater than or equal to one (n>=1).
   - When (n) is equal to one, this is the same as normal Huffman that reads each characters individually.
   - If (n) is equal to two, the program will read two characters at a time, and etc.
2. Change the method that compresses the original text, based on "window-size w", where (w) is appositive integer number greater than or equal to two and less than or equal to five (between two and five 2 <= w <= 5).

Window size: is the number of characters that are compressed at a time.

- When (w) is equal to one is the same as Normal Huffman, so we assume (w) greater than or equal to two.

The figure below shows our proposed model for Dynamic and Dictionary-based technique.

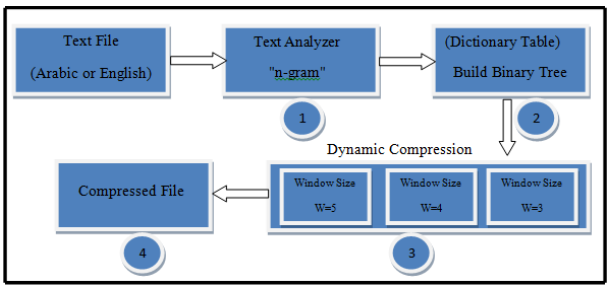

Fig 4: Model of Dynamic-Dictionary based technique.

- The text file can be either English or Arabic text file with different size and with characters and symbols text.

1. The first step is to analyze the text by reading it in integer positive value of characters at a time based on the n-gram.

n-gram: is a positive integer number greater than or equal to one, refers to the number of characters taken together at a time.

This is the first improvement in normal Huffman. Because we deal not only with English text files but also with Arabic text files, we need to read each Arabic character with its inflections, so we assume (n=1, 2) and read each two characters and calculate its frequencies.

- We can use n =1, when the Arabic text file doesn't have inflections

- We can use n =2, when the Arabic text file has inflections
- We assume n =2, to be suitable for different types of Arabic and English text files.

After reading the text characters based on n-gram, we calculate the frequencies for each n-gram.

The output of this process is a table from each n-gram character (n=1 and 2) and its frequency.

1. The next step is to build the binary tree, using the table designed in the previous process. This process is the most important process preparing to assign a code for each n-gram characters.

The output of this process is a table from each ***n-gram*** character, its frequency and it's coding.

2. This step is the final step called Compression to produce a compressed file. In this step, we will replace ``each n-gram character with its coding based on something called window-size w, which is the most important improvement to minimize the number of bits used to describe the original text to achieve compression ratio less than the original value.

Window-size: is a positive integer number between two and five, refers to the number of characters that will be used each time to be compressed.

If we assume window-size starts from one then this process is like a normal Huffman, where each character is replaced with its code individually, so we consider that window-size starts from two.

- When w = 2, we compress each two characters together at a time and compress them.
- When w = 3, we compress each three characters together at a time and compress them.
- When w = 4, we compress each four characters together at a time and compress them.
- When w =5, we compress each five characters together at a time and compress them.

1. In this process, the original text is replaced with a coding text based on window-size, and then this code is saved in a file**.**

3.2 Tiny- Corpus example

If we have this tiny corpus as example: "الحَمدُ لله." which is 12 characters length *(m=12).*

1. By applying the first step, the tables below present all possible characters for different n-gram and their frequencies.

Table 2: Character-frequency table when (n = 1) and (n = 2).

Number	Character	Frequency	number	Character	Frequency
-1-	"ا"	1	**-2-**	"ال"	1
-3-	"ل"	3	**-4-**	"لح"	1
-5-	"ح"	1	**-6-**	"حَ"	1
-7-	"َ"	1	**-8-**	"مَ "	1
-9-	"م"	1	**-10-**	"مد"	1
-11-	"د"	1	**-12-**	"دُ"	1
-13-	"ُ"	2	**-14-**	" ُ"	1
-15-	space	1	**-16-**	"ل "	1
-17-	"لل"	1	**-18-**	"له"	1
-19-	"ه"	1	**-20-**	".ه"	1

1. In this step, we will build the binary tree from this table to assign a code for each n-gram

- The output of this process is a table from each n-gram character, its frequency, its code, and code length.

The table below presents the final table before the compression process.

Table 3: Final Table when (n = 1) and (n = 2)

Char.	Freq.	Code	Code Length	Char.	Freq.	Code	Code Length
"ا"	1	0010	4	"ال"	1	0011	**4**
"ح"	1	0001	4	"لح"	1	0000	**4**
"َ"	1	0111	4	"حَ"	1	0110	**4**
"م"	1	0101	4	"مَ"	1	0100	**4**
"د"	1	11011	5	"مد"	1	11010	**5**
"ُ"	1	11001	5	"دُ"	1	11000	**5**
" ُ"	1	11110	5	Space	1	11111	**5**
"ل "	1	11100	5	"لل"	1	11101	**5**
"له"	1	10010	5	"ه"	1	10011	**5**
".ه"	1	1000	4	"ل"	1	101	**3**

- As we see in this table, when the frequency increases, the number of bits needed to code this character will decrease, this is the efficiency that our technique has proposes.

1. In this process we will compress the original text

- الحَمدُ لله. (when w=2) CT2. 00110110110101111011011000 ➔ Code length = 27
- الحَمدُ لله. (when w=3) CT3. 00100000011111010111010110010 ➔ Code length = 30

- الحَمدُ لله. (when w=4) CT4. 00110110110101111010110010 ➔ Code length = 26
- الحَمدُ لله. (when w=5) CT5. 00100000010011000111001011000 ➔ Code length = 29

4. CONCLUSION

In this study, a new technique for compressing text files based on of D-DBT is presented. The proposed technique tries to compress text files with compression ratio less than the original value using n-gram and window size.

The proposed technique can be used to reduce the size of both Arabic and English text files; in order to be transferred through transmitted channels, or stored on a disk. The efficiency of this technique is generating dynamic dictionary from n-gram characters, and splitting the text file into groups of size equal to window-size, then represent each group by its code from the dictionary using the minimum number of bits based on calculating many solutions for each window-size, then select the minimum one.

By using both n-gram and window size together, the produced text files have compression ratio less than either using n-gram alone such as in Huffman, or window-size alone such as in LZW.

Moreover, the proposed technique compresses the text files with average compression ratio that is equal to 43.87% using window-size that is equal to four.

The properties of this technique over other algorithms could be summarized in the following points:

1. It can be used for different text file sizes.
2. It uses n- gram technique together with window size.
3. It uses different text files either Arabic or English.
4. Simple and easy to implement.
5. The average compression ratio achieved is 43.87% using CT4.

However, more research is needed; we recommend an improvement to this technique by trying to apply it on a large number of text files, and on other languages as well.

Moreover, we hope that through further research in the future to increase the window size in order to reach to a point where the compression ratio will have the smallest value.

5. REFERENCES

[1] Al-Daoud, A. (2010). "Morphological Analysis and Diacritical Arabic Text Compression." International journal of ACM Jordan (ISSN 2078-7952).

[2] Bently J. L., Sleator D. D., Trajan R. E. and Wel V. K., (1986). "A Locally Adaptive Data Compression Scheme. Communications ACM. 29(4): 320-330.

[3] Blelloch, E., (2002). "Introduction to Data Compression." Computer Science Department, Carnegie Mellon University.

[4] Cheok Yan Cheng, "Introduction On Text Compression Using Lempel, Ziv, Welch (LZW) method".

[5] Coremen, Thomas H., Charles E. Leiserson, Ronald L. Rivest. (2002). "Introduction to Algorithms." Second Edition.

[6] Elabdalla, A. R. and Irshid, M. I.,(2001). "An efficient bitwise Huffman coding technique based on source mapping." Computer and electrical engineering 27(1): 265 – 272.

[7] Ghawanmeh, S.; Al-Shalabi, R. and Kanaan, G., (2006). "Efficient Data Compression Sheme using Dynamic Huffman Code Applied on Arabic Language." J. Comput. Sci. 2(1): 885-888. http://www.scipub.org/fulltext/jcs212885-888.pdf

[8] Hjouj Btoush M, siddiqi, M., J.; Akhgar, B. and Dawawdeh, Z. (2008) "Observation on Compressing Text Files of Varying Length". Proceedings of ITNG.

[9] Ida Mengyi Pu. (2006). "Fundamental_Data_Compression".

[10] Jaradat, A. M.; Irshid, M.I. and Nassar, T. T., (2006). "Entropy Reduction of Arabic Text Files."Asian J.Inform.Technol.5(1):578583. http://medwelljournals.com/fillext/ajit/2006/578-583.pdf

[11] Musa, A.; Al-Damour, A., Fraij, F.; Al-Khaleel, O. And Irshid, M. (2010). "A Dynamic and Secure Arabic Text Compression Technique Using Bitwise Lempel-Zive Algorithm." Information technology journal 9(4):673-679.

[12] Omer, I. and Khatatneh, K. (2010). "Arabic Short Text Compression." J. Comput. Sci. 6(1): 24-28.

[13] Arabic-Language, Arabic language history (2011), Retrieved March 22, 2011 from http://www.arabic-language.org/arabic/history.asp

[14] Arturo San Emeterio Campos, Huffman Algorithm, making codes from probability, Retrieved March 22, 2011 from http:// www.arturocampos.com/cp_ch3-1.html

[15] Matt Powell, University of Canterbury, Retrieved March 22, 2011 from http://corpus.canterbury.ac.nz

[16] Wordiq, Huffman Algorithm – Definition (2010), Retrieved March 20, 2011 from http://wordiq.com/definition/Huffman_coding.