International Journal of Computer (IJC)

ISSN 2307-4523

http://gssrr.org/index.php?journal=InternationalJournalOfComputer&page=index

# Efficient Text Compression Algorithm Based on an Existing Dictionary

Chouvalit Khancome*

*Department of Computer Science, Faculty of Science, Ramkhamhaeng University, Huamark District, Bangkok 10240, Thailand.*

*Email: chouvalit@hotmail.com*

**Abstract**

This research article presents a new efficient lossless text compression algorithm based on an existing dictionary. The proposed algorithm represents the target texts to be compressed in a bit form, and the vocabularies are stored in the existing dictionary. Regarding to the results, the time complexity only takes *O(n)* time of both cases of encoding and decoding scenarios. The space complexity is *O(d)* bit(s) per $2^d$ words where *d=1,2,3,…*The theoretical results showed bits per words and maximum spaces to be saved. These results indicated that the maximum original texts could be compressed more than 99 %.

***Keywords:*** text compression; bit-level compression; dictionary base compression.

## 1. Introduction

Text compression is among the most important principles in computer science especially when the target sources are very large. Solving the problem of text compression algorithm is by reading a target document, and algorithms try to resize the target document(s) to minimal space before keeping in the storage. In contrast, decompression algorithms extract the compressed file(s) to the original document(s). Traditionally, the principle of text compression can be divided into lossy compression and lossless compression.

---

* Corresponding author.
E-mail address: chouvalit@hotmail.com, chouvalit.comsci.ru@gmail.com.

The lossy compression is viewed as some data of original texts can be loss when the decompression algorithm is worked. Meanwhile, the lossless compression algorithm extracts a compression file(s) for all data as well as original texts.

Basically, famous algorithms (e.g.,Huffman, Ziv-Lempel, and Factor) are shown in [11] and called the classic algorithms. Recommended on classic algorithms, [1, 2, 3] and [19, 20, 21] used the keyword to handle text compression. The good reviews can be seen in [19, 20] and [26, 27]. Another principle is the bit representation for target texts. Efficient algorithms are shown in [4, 5, 6, 7] and [16, 17, 18]. Up until now, the bit-level is always challenging of researchers how to keep a more space when the source will be compressed into data in the storage.

In granular of compression methods, dictionary-based algorithms are the method accommodated the keywords of the classic algorithm even the bit-level algorithms. There are several dictionary-based algorithms such as [22, 23, 24, 25, 26, 27]. These algorithms are efficient algorithms to keep the unique keywords called vocabularies of original texts to be compressed. Moreover, the dynamic of dictionary is also shown in [27] for a new algorithm of principle. Recently, a big data is highly important principle; as well as, the storage is larger. Additionally, the speed of network is faster to access the data source such as an existing dictionary, which is stored in somewhere of network (e.g., storage on cloud computing).

Thus, this research article introduces the new bit-level algorithms employing the bits to represent the target texts. The new algorithm employs an existing dictionary which stores the vocabularies to be appeared in the target texts. The new method solves the problem by analyzing the keywords in each document; and, these keywords are added into the provided dictionary. Afterwards, all original keywords are represented by a single integer one by one. In the last step, these numbers are converted to the bit forms for storing in a secondary storage or a compressed file.

This algorithm was evaluated by (1) proofing time and space complexities, and (2) using the demonstration texts to illustrate the theoretical results. Regarding to the results, the time complexity only takes $O(n)$ time in the both cases of encoding and decoding scenarios; where $n$ is the length of documents. In theoretical results and based on 2 words to 1024 Mega-words with the length of 8, 16, and 16 characters, the new solution was able to save a maximum space of 99.61 % when using long lengths of words.

The presented sections are organized as follows. Section 2 shows the related works. Section 3 indicates the basic definitions and demonstrates how to accommodating the target text in a bit-level form including the algorithm scenarios. Section 4 presents the compression algorithms and the decompression algorithms. Section 5 illustrates the theoretical results, and section 6 is the discussion. The conclusion and planned future woks are shown in section 7.

## 2. Related Works

The well known algorithms (Huffman, Ziv-Lempel, and Factor) shown in [11], are implemented by the keyword based algorithms. These algorithms saved a minimum space of 0.09% and the maximum space 73.74 %. For the

past few years, [1, 2, 3, 12, 13] have been the keyword based algorithms which represented the data source by analyzing the keywords in the source data and representing their code in the suitable structure. Emergent principle of keyword base is the dictionary-based compression. These keywords are stored in the dictionary. There are many algorithms shown as follows. Two-Level Dictionary-Based Text Compression Scheme [1] showed two-level to keep the data. The original text could be approximately saved in 75%; however, the granular algorithm employed *gzip* and *bzip* for compression. Meanwhile, the compression ratio is 2.01 bits per input character.

StarNT [2] is the fast lossless text transform algorithm. This algorithm utilizes ternary search tree to expedite trans-form encoding. It's improvement in compression performance were shown as 13% over *bzip2-9*, 19% over *gzip-9*, and 10% over PPMD. In the details of algorithms used PPM, Huffman in the dictionary. Data Compression Using Encrypted Text [3] is to define a unique encryption or signature. This algorithm could save the text in 30-60%. Other one is the survey in [26] shown the parsing problem for dictionary-based text compression, and it's efficient compression shown in 22.51-36.45%. The greedy parsing optimality for dictionary-based text compression [27] showed how to create the dynamic dictionary for text compression and referred to classic algorithms such as LZ77.

Algorithms [4, 5, 6, 7] and [14, 15] are optional bit-level algorithms. The low efficiency algorithm [14] used the Boolean function to implement a set of groups to bits which were then considered as minimum terms. This solution could save more than 10% of space. A more efficient algorithm was shown in [15], which could save a maximum space of about 20%. This solution used a fixed-length Hamming (FLH) algorithm which displays such an enhancement in Huffman code. This algorithm used to compress the multimedia file. More information can be archived in [7]. Superior algorithms such as [4] and [7] can save a maximum space of about 80-97%. The algorithm [4] used the technique called ACW(n) (Adaptive Character Word-length). This principle converted the source data into the binary sequence using the curtain character-to-binary format which uses the ASCII code. Then, the sequence was subdivided into *n-bit* lengths which use $d \leq 256$ where $d$ is the size of the alphabet. Afterwards, this method finds the optimum variable of $n$ ($n=9$ and $n=10$).

The algorithm [7] was called the enhanced algorithm of [4]. This solution known as *ACW(n,s)* added a new technique using a subsequence of bit called *s* value. When using the value of $n$ equal 14, the space was saved about 80-97%. However, some cases showed poor performance such as 2% or 50% (see the experimental results in [7]). The new idea is that using the clustering technique to store the original file illustrated in [19]. Other one is the idea of semi-document for keeping data shown in [20]. Another last idea is that the dictionary could be updated all time mentioned in [27].

## 3. Basic Definitions and Algorithm Scenarios

The new algorithm depends on existing dictionary, which stores the vocabularies in the target texts. This section shows the basic definitions and the algorithm scenarios.

### *3.1. Basic Definitions*

**Definition 1.** A single file which consists of several words $w_1$, $w_2$, $w_3$,..,$w_n$, is called a single document and represented by $D$={ $w_1$, $w_2$, $w_3$,..,$w_n$}.

**Example 1**. If the target text to be compressed is 'aaaa bbbb cccc dddd', then the details of text can be described as the single document, shown below.

$D$={aaaa, bbbb, cccc, dddd}, then $w_1$=Aaaa, $w_2$=bbbb, $w_3$=cccc, and $w_4$=dddd.

**Definition 2.** If there are several single documents $D_1$, $D_2$, $D_3$, …, $D_m$ where $D_1$ = { $w_1$, $w_2$, $w_3$,..,$w_{n1}$}, $D_2$ ={ $w_1$, $w_2$, $w_3$,..,$w_{n2}$}, $D_3$ ={ $w_1$, $w_2$, $w_3$,..,$w_{n3}$}, ..., $D_m$={ $w_1$, $w_2$, $w_3$,..,$w_{nm}$}, then all of them are called the multi-documents, denoted by *MD*.

**Example 2**. Shows the multi-documents mentioned in definition 2 and 5 files to be demonstrated where *MD* consists of $D_1$, $D_2$, $D_3$,$D_4$,$D_5$.

*MD*→ Text Files to be compressed

$D_1$= aaaa bbbb cccc dddd (file 1)

$D_2$=dddd fffff cccc eeee (file 2)

$D_3$= bbbb bbbb cccc dddd (file 3)

$D_4$= aaaa gggg hhhh dddd (file 4)

$D_5$= hhhh gggg eeee dddd (file 5)

**Definition 3.** All unique keywords are contained in *MD*, which are analyzed and contained in the temporary space, is called the temporary dictionary and denoted by *TD*.

**Example 3**. Followed by definition 3, example 3 shows the *TD* of example 2 shown in Fig.1.

**Definition 4.** The integer represented the occurrence of $w_i$ in *TD*, which is shown in *MD*, is called *RN*.

**Example 4**. Followed by definition 4, example 4 represents *RN* of *TD* in example 3 shown in Fig.2.

**Definition 5.** *RN* is converted to the bit form called *DBF*.

**Example 5**. Followed by definition 5, example 5 Show *DBF* of *RN* in example 4; e.g.,1→0001, 2→0020, and 3→0011, shown in Fig.3.

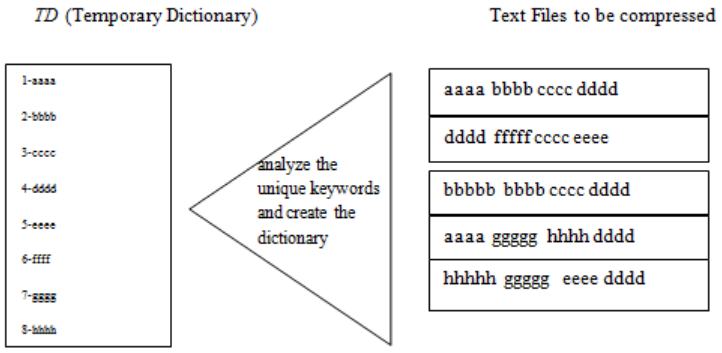


Fig. 1. *TD* of example 2.

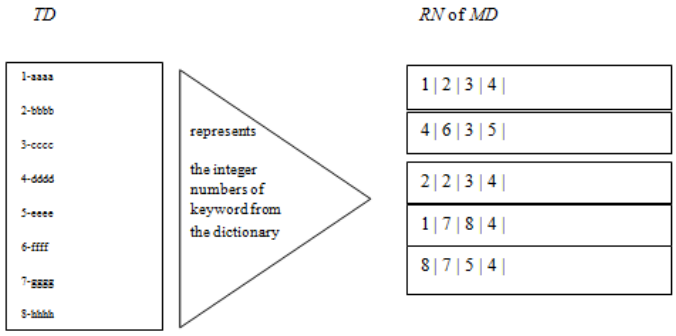


Fig. 2. *RN* of *TD* in example 3.

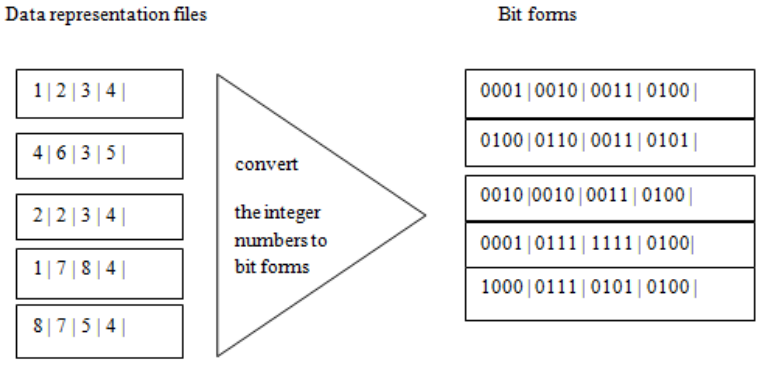


Fig. 3. *DBF* of *RN* in example 4.

**Definition 6.** all keywords in *TD* are stored a special space called application temporary dictionary, denoted by *ATD*.

**Definition 7.** If *DBF* is the compressed bits and stored into the storage, then the file which stores the compressed *DBF* is called the compression file, denoted by *CF*.

**Definition 8.** If all items in *CF* are decompressed into a new file, then that file is called the original file and denoted by *OF*.

### *3.2. Algorithm Scenarios*

According to the mentioned definitions, this sub-section shows the steps of algorithm scenarios. There are 4 steps for text compression shown as below.

Step 1: Read the input, analyze the keywords, and create the dictionary such a Fig 1.

Step 2: Represent data into *RN* such a Fig 2.

Step 3: Convert *RN* to bits form *DBF* such a Fig 3.

Step 4: Use the text compression algorithm to compress employing the existing dictionary, and write the bit forms to the compressed files.

For the decompression algorithm, there are 4 steps to be done.

Step 1: Read the *CF* file

Step 2: Convert to *RN* one by one block of integer

Step 3: Decompress to original words using the existing dictionary

Step 4 : Write the words $w_i$ to the decompressed file one by one.

## 4. Compression and Decompression Algorithms

### *4.1. Compression Algorithm*

Let $n_m$ be the number of documents or target texts, $s_m$ be the total words of each document where $s_m$ depends on the size of each document. The compression algorithm is shown as follows.

**Algorithm 1**: Text Compression Algorithm

Input: *MD, TD, RN*$[n_m]$

Output: *CF*

1. Initiate empty *CF*
2. For $i=1$ to $n_m$ Do
3. For $k=1$ to $s_m$ Do
4. If $w_k$ not exists in *ATD* Then
5. Add current keyword to *ATD*
6. End If
7. Add number of related $w_k$ in *ATD* into $RN[i][s_k]$
8. End For
9. End For
10. Convert the *RN* to *DBF*
11. Convert *DBF* to *CF*
12. Write *CF to storage*

**Theorem 1.** All words in *MD* can be compressed in *O(n)* time where *n* is the total word in *MD*.

**Proof.** The main complexity is from the loop of line 2 and line 3. Line 2 is run equal to $n_m$ times that equals *n* time; meanwhile, the loop of line 3 is run from 1 to each $s_m$ of each document. Thus, each *n* needs to run by inner loop of $s_m$ word. Then, all times of running time equal all of vocabularies in *MD*. The time to convert *RN* to *DBF* and *DBF* to *CF* are also *n* time. The other lines (1, 5, 7, 10,11,and 12) take *O(1)*. Therefore, overall time complexity is O(*n*).□

**Theorem 2.** The space complexity of all words in *MD* is *O(d)* bit(s) per $2^d$ words where $d=1,2,3,\ldots$.

**Proof.** Supposing that the exiting dictionary contains the order of the integer number and the series of keywords, and they are ordered by increasing numbers. If there are one or two words, then the represented bits referring to the number of vocabulary in the dictionary uses only one bit of $2^1$ words. Then, if there are $2^2,2^3,2^4,\ldots, 2^d$ words, then the space are $2,3,4,\ldots, d$ bits. Therefore, the space of all words in *MD* takes *O(d)* bit(s) per $2^d$ words. □

### *4.2. Decompression Algorithm*

Using the decompression algorithm, the temporary space is initiated and denoted by *TM*. Then, the algorithm reads a related block of *RN* one by one and converts the bit form to words, which will be stored in *TM*. If all bits are converted, then the words are stored in the storage (*OF*). Additionally, let $m_{max}$ be the maximum length in $s_m$. The main algorithm is shown as below.

**Algorithm 2**: Decompression Algorithm

Input: *CF*, *ATD*, $m_{max}$,*n*

Output: *OF*

1. Initiate the space *DBF* with size $m_{max}xn$ (*TM*)

2. While *CF* not *EOF* Do
3. Read data from *CF*
4. Convert each data to *RN* form
5. Read record which corresponds *RN* of *ATD* and get word form *ATD*
6. Write that word to *TM*
7. End While
8. Write *TM* to *OF*

**Theorem 3.** If there are the compressed file *CF*, which is compressed by algorithm 1, then extracting *CF* to *OF* takes O(*n*).

**Proof.** Line 1 takes *O(1)* time. The loop of 'while' from line 2 to line 7 needs to be explained. If the size of *CF* is *n*, then the line 3 takes *O(n).* As well as, the line 4, 5 and 6 also take *O(n).* Thus, the time complexity from the line 2 to line 7 takes *4n* which is *O(n).* In the last line, line 8 takes *O(1).* Therefore, the time complexity of decompression algorithm is only *O(n).* □

**Theorem 4.** If there are the compressed file *CF*, then the extraction of *CF* to *OF* takes $O(m_{max}xn)$ space where *n* is the size of *CF* , $m_{max}$ is the maximum length of blocks in *CF*.

**Proof.** Assuming that each line or block in CF keeps $n_1,n_2,n_3,\ldots$ words where $n_1+n_2+n_3+,.. = n$. The length of lines, which contains $n_1+n_2+n_3+,..$ words, are $m_1,m_2,m_3,... m_{max},\ldots$ where $m_{max}$ is the maximum length of them and $m_1+m_2+m_3+..+m_{max,}\ldots=m$. Corresponding to line 1 of algorithm, the required space for keeping *TM* is *mxn* space which implies $O(m_{max}xn)$. □

**5. Theoretical Results**

The theoretical results are focused on the saved space per word and the target texts from kilo-word(s) to mega-word(s) within the given length of words. Table 1 to table 3 show the space could be saved per word when the vocabularies 2-1024 words while the length of word 8, 16, and 32 characters, respectively. Table 4 to table 6 show the saved spaces when the target texts are 2 -1024 words. Additionally, table 7 shows the large target texts from 2-1024 Mega-words when the word length is 32 characters.

The formula for calculation of table 1 to table 3 is shown as below.

*% of Saved Space (per word) = ((Original Bits)-Required Space)/Original Bits)*100* (1)

Table 1. Saved space per word when the length of word is 8 (Original Bits=64 ASCII, 128 Unicode).

| # of vocabularies(words) | The required space per word (bits) | Saved space of ASCII text (% per word) | Saved space of Unicode text (%) |
|---|---|---|---|
| 2 | 1 | 98.43 | 99.21 |
| 4 | 2 | 96.87 | 98.43 |
| 8 | 3 | 95.31 | 97.65 |
| 16 | 4 | 93.75 | 96.87 |
| 32 | 5 | 92.18 | 96.09 |
| 64 | 6 | 90.62 | 95.31 |
| 128 | 7 | 89.06 | 94.53 |
| 256 | 8 | 87.50 | 93.75 |
| 512 | 9 | 85.93 | 92.96 |
| 1024 | 10 | 84.37 | 92.18 |

Table 2. Saved space per word when the length of word is 16 (Original Bit=128 ASCII, 256 Unicode).

| # of vocabularies(words) | The required space per word (bits) | Saved space of ASCII text (% per word) | Saved space of Unicode text (%) |
|---|---|---|---|
| 2 | 1 | 99.22 | 99.61 |
| 4 | 2 | 98.44 | 99.22 |
| 8 | 3 | 97.66 | 98.83 |
| 16 | 4 | 96.88 | 98.44 |
| 32 | 5 | 96.09 | 98.05 |
| 64 | 6 | 95.31 | 97.66 |
| 128 | 7 | 94.53 | 97.27 |
| 256 | 8 | 93.75 | 96.88 |
| 512 | 9 | 92.97 | 96.48 |
| 1024 | 10 | 92.19 | 96.09 |

Table 3. Saved space per word when the length of word is 32 (Original Bit=256 ASCII, 512 Unicode).

| # of vocabularies(words) | The required space per word (bits) | Saved space of ASCII text (% per word) | Saved space of Unicode (%) |
|---|---|---|---|
| 2 | 1 | 99.61 | 99.80 |
| 4 | 2 | 99.22 | 99.61 |
| 8 | 3 | 98.83 | 99.41 |
| 16 | 4 | 98.44 | 99.22 |
| 32 | 5 | 98.05 | 99.02 |
| 64 | 6 | 97.66 | 98.83 |
| 128 | 7 | 97.27 | 98.63 |
| 256 | 8 | 96.88 | 98.44 |
| 512 | 9 | 96.48 | 98.24 |
| 1024 | 10 | 96.09 | 98.05 |

The formula for calculation of table 4 to table 6 is shown as below.

*% of Saved Space = ((Original Bytes)-Required Byte Spaces)/Original Bytes)*100*
*(2)*

*Original Byte = Kilo-Words *8, Required Byte = Required Bits/8*
*(3)*

Table 4. Compression results when the original text equals 1 Kilo-Words with 8 character lengths (ASCII=8192 Bytes, Unicode=16394 Bytes)

| # of vocabularies(words) | The required space (Bytes) | Saved spaces (%) of ASCII code | Saved space (%) of Unicode |
|---|---|---|---|
| 2 | 256 | 96.88 | 98.44 |
| 4 | 384 | 95.31 | 97.66 |
| 8 | 512 | 93.75 | 96.88 |
| 16 | 640 | 92.19 | 96.09 |
| 32 | 768 | 90.63 | 95.31 |
| 64 | 896 | 89.06 | 94.53 |
| 128 | 1024 | 87.50 | 93.75 |
| 256 | 1152 | 85.94 | 92.97 |
| 512 | 1280 | 84.38 | 92.19 |
| 1024 | 1408 | 82.81 | 91.41 |

Table 5. Compression results when the original text equals 1 Kilo-Words with 16 character lengths (ASCII=16384 Bytes, Unicode=32768 Bytes)

| # of vocabularies(words) | The required space (Bytes) | Saved spaces (%) of ASCII code | Saved spaces (%) of Unicode |
|---|---|---|---|
| 2 | 256 | 98.44 | 99.22 |
| 4 | 384 | 97.66 | 98.83 |
| 8 | 512 | 96.88 | 98.44 |
| 16 | 640 | 96.09 | 98.05 |
| 32 | 768 | 95.31 | 97.66 |
| 64 | 896 | 94.53 | 97.27 |
| 128 | 1024 | 93.75 | 96.88 |
| 256 | 1152 | 92.97 | 96.48 |
| 512 | 1280 | 92.19 | 96.09 |
| 1024 | 1408 | 91.41 | 95.70 |

Table 6. Compression results when the original text equals 1 Kilo-Words with 32 character lengths (ASCII=32768 Bytes, Unicode=65536 Bytes)

| # of vocabularies(words) | The required space (Bytes) | Saved spaces (%) of ASCII code | Saved spaces (%) of Unicode |
|---|---|---|---|
| 2 | 256 | 99.22 | 99.61 |
| 4 | 384 | 98.83 | 99.41 |
| 8 | 512 | 98.44 | 99.22 |
| 16 | 640 | 98.05 | 99.02 |
| 32 | 768 | 97.66 | 97.66 |
| 64 | 896 | 97.27 | 98.83 |
| 128 | 1024 | 96.88 | 98.63 |
| 256 | 1152 | 96.48 | 98.44 |
| 512 | 1280 | 96.09 | 98.05 |
| 1024 | 1408 | 95.70 | 97.85 |

The formula for calculation of table 7 is shown as below.

*% of Saved Space = ((Original Mega-Bytes)-Required Mega-Byte Spaces)/Original Mega-Bytes)*100 (3)*

*Original Mega-Byte = Mega-Words *1048576*word length, (4)*

*Required Mega-Byte = (Required Bits* (Mega-Words *1048576))/1048576*
*(5)*

Table 7. Compression results when the original text in Mega-Words with 32 character lengths and 32-bits per word.

| # of vocabularies(Mega-words) | The required space per word (bits) | ASCII text (Bytes)/Unicode (MB) | The required space (MB) | Saved spaces (%) of ASCII code | Saved spaces (%) of Unicode |
|---|---|---|---|---|---|
| 2 | 21 | 64/128 | 5.25 | 91.80 | 95.90 |
| 4 | 22 | 128/256 | 11 | 91.41 | 95.70 |
| 8 | 23 | 512/1024 | 23 | 95.51 | 97.75 |
| 16 | 24 | 1024/2048 | 48 | 95.31 | 97.66 |
| 32 | 25 | 2048/4096 | 100 | 95.12 | 97.56 |
| 64 | 26 | 4096/8192 | 208 | 94.92 | 97.46 |
| 128 | 27 | 8192/16384 | 432 | 94.73 | 97.36 |
| 256 | 28 | 16384/32768 | 896 | 94.53 | 97.27 |
| 512 | 29 | 32768/65536 | 1856 | 94.34 | 97.17 |
| 1024 | 30 | 65536/131072 | 3840 | 94.14 | 97.07 |

## 6. Discussion

This algorithm only shows the theoretical results that need the space to keep the packed filed less than 20% per each result. However, the algorithm was assumed that there is an existing dictionary. That is, the dictionary

could be used to keep all unique words. If there are several unique keywords and no redundant of words, then this algorithm could be used space more than the original text. Suggestion that the dictionary should use the efficient algorithms (e.g., [16,17, 18] or [26, 27]) to stored the vocabularies.

## 7. Conclusion

Based on an existing dictionary, the new algorithm was presented. The proposed algorithm employed the bit-level to store the target texts. The time complexity of the compressing algorithm takes *O(n)* time, and the time complexity of the decompressing algorithm also takes *O(n)* time where *n* is the length of the source data. In compressing phase, the space complexity takes *O(d)* per $2^d$ words where *d=1,2,3,…*, and takes $O(m_{max}xn)$ space in decompressing phase where $m_{max}$ is the maximum words in the block of compressed files. The theoretical results showed that the average spaces to be saved per word were from 87.66-99.41% when target texts were from 2-1024 mega-words. As well as, a maximum saved space was 99.80%.

## References

[1] A. Mofat, and R.Y.K. Isal. Word-based text compression using the burrows-wheeler transform. *Information Processing and Management*, Vol. 41, No. 5, 2005, 1175-1192.

[2] J. Adiego, and P. de. la Feunte, On the use of words as source alphabet symbols in PPM. *In Proceedings of Data Compression Conference, IEEE*, 2006, 435.

[3] J. Lánský and M. Žemlička. Text compression: Syllables. *In Proceedings of the Dateso Workshop on Database, Texts, Specifications and Objects*, 2005, 32-45.

[4] H. Al-Bahadili and S. M. Hussain. An adaptive character wordlength algorithm for data compression. *Computers & Mathematics with Applications*, Vol. 55, No. 6, 2008, 1250-1256.

[5] S. Nofal. Bit-level text compression. *In Proceedings of the 1st International Conference on Digital Communications and Computer Applications, Irbid, Jordan*, 2007, 486-488.

[6] A. Rababáa. An Adaptive Bit-Level Text Compression Scheme Based on the HCDC Algorithm. *M.Sc., dissertation, Amman Arab University for Graduate Studies, Amman, Jordan*, 2008.

[7] H. Al-Bahadili and S. M. Hussain. A Bit-level Text Compression Scheme Based on the ACW Algorithm. *International Journal of Automation and Computing*, Vol. 7 No. 1, 2010, 123-131.

[8] C. Monz and M. de. Rijke, (2006, August, 12). Inverted Index Construction. Available: http://staff.science.uva.nl/~christof/ courses/ir/transparencies/clean-w-05.pdf.

[9] O. R. Zaïane, (2001, September 15), CMPUT 391: Inverted Index for Information Retrieval, *University of Alberta.* Available: http://www.cs.ualberta.ca/~zaiane/courses /cmput39-03/.

[10] R. B. Yates and B. R. Neto. Mordern Information Retrieval. *The ACM press. A Division of the Association for Computing Machinery, Inc*. 1999, 191-227.

[11] M. Crochemore, and W. Rytter, (2010, March, 18). Text Algorithms. Available: http://monge.univ-mlv.fr/~mac/REC/text-algorithms.pdf.

[12] R. Y. K. Isal and A. Moffat. Word-Based Block-Sorting Text Compression. *ACSC '01: Proceedings of the 24th Australasian conference on Computer Science, IEEE Computer Society*, 2001, 92-99.

[13] R. Y. K. Isal, A. Moffat and A. C.H. Ngai. Enhanced Word-Based Block-Sorting Text Compression. *ACSC '02: Proceedings of the twenty-fifth Australasian conference on Computer science, Australian Computer Society, Inc*., Vol. 4, 2002, 129-137.

[14] G. Caire, S. Shamai and S. Verdu. Noiseless data compression with low density parity check codes. *Advancees in Network Information Theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, P. Gupta, G. kramer, A. J. van Wijingaarden, Ed.*, Vol. 66, 2004, 263-284.

[15] A. A. Sharieh. An enhancement of Huffman coding for the compression of multimedia file. *Transactions of Engineering Computing and Technology*, Vol. 3, No. 1, 2004, 303-305.

[16] C. Khancome. Bit-level Text Compression Algorithm Using Position of Characters**.** *2010 2nd International Conference on Information and Multimedia Technology (ICIMT 2010).* Vol. 1-242, *2010,* 242-245.

[17] C. Khancome. New Full Text Compression Algorithm Based on Position of Character **.***2010 3rd International Conference on Computer and Electrical Engineering (ICCEE 2010).* IEEE Conference, Vol. 5, 2010, 631-634.

[18] C. Khancome. Text Compression Algorithm Using Bits for Character Representation. *International Journal of Advanced Computer Science.* Vol. 1, No. 6, 2010, .219-215

[19] J. Dvorsk´y, J. Martinovi˘c. J. Pokorn´y, V. Sn´a˘sel, and K. Richta (Eds.). Improvement of Text Compression Parameters Using Cluster Analysis, *Dateso 2007*, 115–126

[20] L. Galambos, J. Lansky, M. Zemlicka, and K. Chernik. Compression of Semistructured Documents. *INTERNATIONAL JOURNAL OF INFORMATION TECHNOLOGY VOLUME 4 NUMBER 1 2007*.

[21] M. Crochemore and F. Mignosi and A. Restivo and S. Salemi. Text Compression Using Antidictionaries. In 26th Internationale Colloquium on Automata, Languages and Programming (ICALP). 1998, 261-270.

[22] Z. Karim Zia, D. Fayzur Rahman, and C. Mofizur Rahman. Two-Level Dictionary-Based Text Compression Scheme . Proceedings of 11th International Conference on Computer and Information Technology (ICCIT 2008) 25-27 December, 2008, Khulna, Bangladesh, 13-18.

[23] W. Wen-Yen and J. W. Mao-Jiun, "Two-dimensional object recognition through two-stage string matching," Image Processing, IEEE Transactions on, vol. 8, 978-981, 1999.

[24] F. Amar Mukherjee. Data Compression Using Encrypted Text Robert. Proceedings of ADL '96 ,1996, 130-138.

[25] G. Hwee Ong and S. Ying Huang. A Data Compression Scheme for Chinese Text Files Using Huffman Coding and a Two-Level Dictionary. INFORMATION SCIENCES 84, 85 99 (1995) 85-99.

[26] A. Langiu. On parsing optimality for dictionary-based text compression—the Zip case, *Journal of Discrete Algorithms*, 20 (2013) 65–70.

[27] M. Crochemore, A. Langiu, and F. Mignosi. Note on the greedy parsing optimality for dictionary-based text compression. Theoretical Computer Science 525 (2014) 55–59.