Algorithms for Data-Compression in Wireless Computing Systems

Upasana Mahajan

Prashanth C.S.R
Prof & HOD, Dept. of CSE, NHCE,Bangalore,India

Abstract
Compression is a technique used for reducing data size by manipulating data redundancy; so that the packet transmission time and storage cost can be reduced. This can be achieved with the use of suitable data compression algorithms. Choosing the right algorithm can be accomplished by analyzing the performance of the algorithm. This paper presents the survey of various lossless data compression algorithms.

Keywords
Lossless data compression, irreducible substitution tables, wireless sensor networks, compression algorithms, compression ratio

1. INTRODUCTION

Data Compression can be defined as encoding the information using the small number of bits rather than original representation. There are two types of data compression, lossless and lossy compression. The lossy compression is a method of data encoding, in which compression is done by discarding/losing some data. This is commonly used in multimedia data, especially in applications like streaming media and internet telephony. In this some loss of information is acceptable. Dropping nonessential detail from the data source can save storage space. There are two basic lossy compression schemes1lossy transform codecsand lossy predictive codecs. The lossless data compression can be defined as reducing the bits by identifying and eliminating statistical redundancy. The lossless data compression is reversible of lossy compression, such thatthe exact original data to be reconstructed from the compressed data. Lossless compression can be used for images, audios etc. but mostly it is used for text data like executable program, text documents and source code. In this paper focus is only on the lossless data compression. There are different types of lossless data compression algorithms2 like Huffman's coding, Run Length encoding, Dictionary coders (LZW) etc. Based on the algorithm performance factors10 like compression ratio, saving percentage and compression time, we choose the algorithm for compressing the data. The ultimate goal is to study different algorithms and select the best for compression.

Manuscript received September 5, 2014
Manuscript revised September 20, 2014

2. LITERATURE SURVERY

2.1. Lossless Data Compression Algorithms Based on Substitution Tables [3] [4]

This paper introduces a class of new lossless data compression algorithm. Each algorithm first tries to transforms the original data, which is to be compressed into an irreducible table representation and then uses anarithmetic code to compress the irreducible table representation. These are generally known as universal coding algorithms as they try to achieve the compression rate. These new range of lossless data compression algorithm has been developed to improve overall compression rate and performance with the help of different variants of hierarchical transformations.
Firstly, the tables are formed with the help of parallel substitution which ends up with a unique string using the reduction rules. In this research 5 different reduction rules have been implemented and with the help of which, less complex tables are formed. For example, Let x be a string from A which is to be compressed. Starting from the table T consisting of only one row (s, x), a hierarchical transformation applies repeatedly the reduction rules 1-5 in some order to reduce T to an irreducible substitution table. To compress x, the corresponding algorithm then uses the zero order arithmetic code to compress the irreducible table. After receiving the code word of T', one can fully recover T' from which x can be obtained via parallel substitution. Some examples of hierarchical transformation are Greedy SequentialTransformation, SEQUITUR Transformation, Multilevel Pattern Matching Transformation (MPM),
The greedy sequential transformation parses the sequence, into non-overlapping substring and build a sequentially an irreducible table for each substring. This algorithm helps in sequential compression. The SEQUITUR algorithm has two main rules:
1. No pair of adjacent symbols appears more than once in the grammar.
2. Every rule is used more than once.
This helps to build irreducible table for each prefix and then append a substring to the end of the row at last apply

the reduction rules 1-5 to reduce the table. It transforms the binary sequence.

The MPM transformation bisects each distinct substring repeatedly, until the length of substring is 2. Then, assign a unique token to each substring and create a substitution table. The MPM code and Lempel-Ziv code have similarities like both are pure pattern matching codes, so they do not directly compress the data. But there are differences like MPM is a hierarchical transformation so it does pattern matching at multiple levels and the LZE is non-hierarchical. The MPM code was developed for, strictly for data of length a power of two, and named the bisection algorithm.

This research helps in trying to solve a problem of performance of an algorithm. It can be evaluated mainly by calculating and comparing the two facts:Frequency of a block of a sequence and Empirical Entropy of a sequence.

2.2. A Simple Algorithm for Data Compression in Wireless Sensor Networks [5] [6]

Sensor Nodes have small batteries which cannot be changed or recharged frequently, so the WSN have an issue of Energy. Power saving can be done by either duty cycling (coordinated sleep/wakeup schedules between nodes) or by in-network processing (compression/aggregation techniques). Data compression is the best option and appreciated only if the execution of compression algorithms requires lesser amount of energy than the one saved in reducing transmission. This paper introduces the algorithm known as Lossless Entropy Compression (‘.,LEC), which shows the correlation between the data collected by sensor nodes and the entropy compression. This algorithm follows same scheme used in baseline JPEG algorithm for compressing the DC-coefficients of a digital image. The Huffman table proposed in JPEG to entropy encoding the groups has been adopted.

```
Encode (di, Table)
 IF di=0 THEN
    SET ni = 0
 ELSE
    SET ni = ⌜ log2(|di|) ⌝
ENDIF
    SET si TO Table [ni]
IF ni= 0 THEN
    SET bsi = si
ELSE
  IF di > 0 THEN
    SET ai= (di)|ni
ELSE
     SET ai = (di – 1)|ni
   ENDIF
   SET bsi TO <<si, ai>>
  ENDIF
RETURN bs
```

Pseudo-code of the encode algorithm

n_i	s_i	d_i
0	00	0
1	010	-1,+1
2	011	-3,-2,+2,+3
3	100	-7..-4,+4..+7
4	101	-15..-8,+8..+15
5	110	-31..-16,+16..+31
6	1110	-63..-32,+32..+63
7	11110	-127..-64,+64..+127
8	111110	-255..-128,+128..+255
9	1111110	-511..-256,+256..+511
10	11111110	-1023..-512,+512..+1023
11	111111110	-2047..-1024,+1024..+2047
12	1111111110	-4095..-2048,+2048..+4095
13	11111111110	-8191..-4096,+4096..+8191
14	111111111110	-16383..-8192,+8192..+16383

Table: The Huffman variable length codes used in the experiment.

The difference d_i computed by algorithm for the input to an entropy encoder. The $d_i = bs_i$(bit sequence) = $s_i|a_i$, s_i codifies the number $n_i$and a_i represents d_i. If

i. $d_i > 0$, $a_i = n_i$ low order bits of the 2’s complement representation of d_i.
ii. $d_i < 0$, $a_i = n_i$ low order bits of the 2’s complement representation of $d_i - 1$.
iii. $d_i = 0$, $s_i$is coded as 00 and a_i is not represented.

EXPERIMENTAL RESULTS:

The performance of a compressed algorithm can be defined by compression ratio as shown below:

$$\text{Comp Ratio} = 100 \text{ X } 1 \left[- \frac{\text{Comp\_Size}}{\text{Orig\_Size}} \right]$$

With the help of datasheets given in SHT11 for temperature & relative humidity and using the above formula, following ratios are obtained. Thus the comparison between other compression algorithm ratios as per following results shows that the LEC algorithm performs better.

	Temprature		Relative Humidity	
	Comp_size	*Comp_ratio*	*Comp_size*	*Comp_ratio*

LEC	7605bits	66.99%	7527bits	67.33%
S-LZW	16760bits	27.25%	13232bits	42.57%
gzip	15960bits	30.73%	13320bits	42.19%
Bzip2	15992bits	30.59%	13120bits	43.05%

2.3. Online Adaptive Compression in Delay Sensitive Wireless Sensor Networks [7] [8]

In wireless sensor networks (WSN), compression reduces the data size by exploiting the redundancy residing in sensing data. This reduction of the data can be measured as compression ratio which is calculated as original data size divided by the compressed data size. The higher the compression ratio means more data reduction is done and results in shorter communication delays. To understand the effect of compression, firstly obtain the processing time of compression, which depends on several factors like compression algorithm, CPU frequency, processor architecture and compression data. There are so many compression algorithm have been developed, but one of the best is Lempel-Ziv-Welch (LZW). LZW is a dictionary based lossless compression algorithm suitable for sensor nodes which replaces the strings of characters with single codes in the dictionary. To calculate the compression delay, the software estimation approach is adopted. The source code of this algorithm is written in C and then converted into the assembly codes, which have fixed number of execution cycles.

LZW Compression Algorithm

```
STRING = get the first character

while there are still input character

 C = get next character

look up STRING + C in the dictionary
if STRING+C is in the dictionary
        STRING = STRING + C
else
        output the code for STRING
        add STRING+C to the dictionary
        STRING = C
        end if
end while
output the code for STRING
```

The total count of cycles can be obtained at the completion of LZW algorithm. The processing time of the algorithm can be calculated by dividing the total execution cycles by the working (i.e. CPU) frequency. There are different experiments conducted in the NS-2 simulator to check out the effect of compression on the packet delays. The results of the experiments shows that delay can cause severe performance degradation underlight traffic load and if traffic loadis heavy than compression reduces the delay of packet, increase the maximum throughput. So the compression is favored only when the packet generation rate is higher than the threshold rate. Therefore to determine whether the compression of data is required or not the online adaptive algorithm has been developed.

The adaptive compression algorithm is distributively implemented on each sensor node as ACS (Adaptive Compression Service) in an individual layer created in a network stack. The main goal of this algorithm is to take a right decision, that whether packet transmission is required or not at a particular node. Before moving to algorithm, let's have a look of the architecture of ACS. There are 4 functional units: 1) Controller manages the traffic flow and makescompression decisions on each incoming packet in this layer.2) The LZW compressor performs actual compression of packet with the help of LZW algorithm.3) The information collector helps in collecting local statics information about network and hardware conditions. 4) The packet buffer helps in temporarily storing the packets to be compressed.

As compression is managed by the node state, so the adaptive algorithm helps to determine the node state. In this algorithm the utilization of the queuing model is done for estimation of the node state conditions. The queuing model includes the network model and the MAC model. The network model defines the network topology and traffic (i.e. estimates the arrival rates of each node). The MAC model defines the packet service time with the help of DCF (Distributed Coordination Function), which can be calculated as the time when packet enters the MAC layer to the time when packet is successfully transmitted or discarded.

The Adaptive Compression Algorithm is divided into two stages: Information collection and State determination. Firstly, in ACS the information collector collects the statistics information like compression statistics (compression ratio rc, average compression processing time Tp, the coefficient of variance of processing time cp), MAC layer service time and packet arrival rates. Once the collector finishes its job, the controller in the ACS defines the state of the node i.e. whether compression is required or not. For making the decision the following State Determination Procedure has been adopted which is performed at the end of each time slot for a node in a No-Compression state.

For each node at level i:

if state = No-Compression **then**
read statistics from the information collector
compute T_{com}, ΔT_{min}
if $T_{com} \leq \Delta T_{min}$ **then**
set state to Compression
else
set i to the node's level number
$\Delta T_{mac} = 0$
while i > 0
calculate λ^i and
compute reduction $\Delta T_{mac}(i)$
add $\lambda^i \Delta T_{mac}(i)$ to ΔT_{mac}
decrease i by one
end while
if $\lambda_c T_{com} \leq \Delta T_{mac}$ then
set state to Compression
end if
end if

For each node at level i:
if state = No-Compression **then**
read statistics from the information collector
compute T_{com}, ΔT_{min}
if $T_{com} \leq \Delta T_{min}$ **then**
set state to Compression
else
set i to the node's level number
$\Delta T_{mac} = 0$
while i > 0
calculate λ^i and
compute reduction $\Delta T_{mac}(i)$
add $\lambda^i \Delta T_{mac}(i)$ to ΔT_{mac}
decrease i by one
end while
if $\lambda_c T_{com} \leq \Delta T_{mac}$ then
set state to Compression
end if
end if

T_{com} the average packet waiting time at the compression queue.
ΔT_{min} lower bound of total delay reduction
ΔT_{mac} MAC layer service time
$\Delta T_{mac}(i)$ Delay reduction in level i
λ_c Arrival rate compression
λ^i Mean arrival rate for nodes in level i

With help of the queuing model, it is possible to calculate the terms/equation used in algorithm. So the outcome this paper is that using the online adaptive compression algorithm, each node can decide whether the packet is compressed or not, adapting to the current network and hardware environment.

2.4. A Statistical Lempel-Ziv Compression Algorithm for Personal Digital Assistant (PDA) [9]

This paper introduces a compression algorithm named as Statistical Lempel-Ziv Compression algorithm (SLZ), which is suitable for the applications of hand held PDAs and can be viewed as a variant of LZ77.

The first step of the algorithm is to build a dictionary which may include up to $2^{|c|}$ entries (supposing each fixed length codeword *c* contains |c| bits). To build a good dictionary, a two pass approach is adopted. The first pass is to collect most useful phrases from the file for building a dictionary. The second pass is to do compression by creating codewords that refer to the phrases in the dictionary. While building a dictionary, there must be a balance between the dictionary size and codeword length to avoid large number of phrases. Therefore for a file of *T* symbols long, the total number of phrases will be:

$$\sum_{i=0}^{T-1} T-i = \frac{T^2 + T}{2} \approx O(T^2)$$

The sliding window approach which has been used in LZ77 can be adopted to reduce the number of phrases. Let's imagine the sliding window of size W symbols, such that W<<T then

$$\sum_{i=0}^{W-1} T-i = \frac{W(2T - W + 1)}{2} \approx O(WT)$$

Once done with the number of phrases, time is to decide which phrases have to collect. If the phrase collected from the file is found in the dictionary, then there is no need to add that phrase in the dictionary but the number of counts/ frequency of that phrase is incremented. On the other hand, two identical phrases having overlap in the input file must be counted as single occurrence instead of two. This overlap detection can be done by adding a time stamp (last time at which the phrase occurred in the file) to each entry in the dictionary. When a phrase is fetched from the input

file and an identical phrase is found in the dictionary, compare the timestamp of that phrase in the dictionary with the current time stamp. If time stamp difference is less than the phrase length, overlap is detected.

After collecting the phrases, time to put all phrases in the dictionary with respect of dictionary size. The size of dictionary should not be too large and too small; it must contain all useful phrases. The number of entries in the dictionary can be reduced by pruning the phrases having unit frequency. Which means prune the phrases which occur once in a file not the phrases that are one symbol long, and it can be done at end of the first pass. With this method most of the time the newly appears phrases are purged. So to avoid this problem another method of pruning the phrases known as Move-To-Front approach is used.

In this approach, when a new phrase inserted it is move to the front of the dictionary.The time dictionary is full, discard the phrase at the end of the dictionary. With this method the phrases which have high frequency will be at the front and phrases with least frequency located at the end of the dictionary. Once the dictionary has been build, it's time for compression which can be done with the help of entropy coding method. The symbols of the file are shifted into the sliding window and once it's full, the symbol sequence in window is compared with the phrases one by one in the dictionary in the order of entropy. When matched phrase is found, the matched symbols in the window are coded by the index of that phrase. The symbols that matched the phrases are removed from the sliding window and new symbols are moved to sliding window. As soon as window is full repeat the process again until all the symbols get coded.

We can't say that this is the best compression algorithm but a simple entropy coding scheme designed using the prefix codes to eliminate look-up table for decoding. Using the combination of dictionary based algorithm and sliding window approach, the overall compression ratio decreases.

2.5. Comparison of Lossless Data Compression Algorithms for Text Data[10]

Data compression helps in reducing the size of the file, in other words compression represents the information in a compact form rather than its original form without any data loss. When data compression is done while transmitting the data, the main concern is speed. Speed of the transmission depends upon the number of bits sent, the time required for the encoder to generate the coded message and the time required for the decoder to recover the original message. Sometimes the lossless compression algorithms are also known as reversible algorithms, as the original message can achieve by the decompression process. Some of the famous lossless compression algorithms are Run-Length Encoding (RLE), Huffman Encoding, Adaptive Huffman Encoding, Shannon Fano algorithm, Arithmetic Encoding and Lempel Zev Welch algorithms.

This paper introduces the comparison of performances of above algorithms, based on different factors. There are many different ways to measure the performance of a compression algorithm. The main concern is space and time efficiency, while measuring the performance. Following are some factors used to evaluate the performances of the lossless algorithms.

Compression Ratio =size after compression
sizebefore compression

Compression Factor = size before compression
size after compression

Saving Percentage =
size before compression –size after compression % size before compression

Compression Time can be defines as time taken to compress particular file. Time taken for the compression and decompression should be considered separately. For a particular file, if the compression and decompression time is less and in an acceptable level, it means that algorithm is acceptable with respect to time.

Entropy can be used as a performance factor, if the compression algorithm is based on statistical information of the source file. Let set of event be S= $\{s_1,s_2,s_3 ,\ldots s_n\}$for an alphabet and each s_j is a symbol used in this alphabet. Let the occurrence probability of each event be $p_j$for event s_j. Then the self-information I(s) is defined as follows:

$I(s) = \log_b 1/ p_j$or $I(s) = - \log_b 1/ p_j$

The first order Entropy value H(P) can be calculated as follows:

$H(P) = {}^{n}\sum_{j=1} p_j I(s_j)$ or $H(P) = - {}^{n}\sum_{j=1} p_j I(s_j)$

Code Efficiency is the ratio between the entropy of the source and the average code length.

$$E(P,L) = \frac{H(P)}{\bar{l}(P,L)} 100\% ,$$

E (P,L) is the code efficiency, H(P) is entropy and $\bar{l}(P,L)$ is average code length.

Average code length defined as the average number of bits required to represent a single code word. It can be calculated as: $\bar{l} = {}^{n}\sum_{j=1} p_j l_j$, where $p_j$is the occurrence probability of j^{th} symbol of the source message, l_j is the length of the particular code word for that symbol and $L = \{l_1, l_2, \ldots\ldots, l_n\}$.

In order to test the performance of above mentioned lossless compression algorithms, first step is to implement them and then test them with some set of files. Performances evaluated by computing above mentioned factors. After the implementation and testing the results shows that the Adaptive Huffman algorithm needs larger time period for processing, because the tree should be updated or recreated. LZW works better as the file size grows up to certain limit, because there are more chances of replacing the words by using the small index number. But it cannot be used for all cases, so can't say it is one of the efficient algorithms.
Arithmetic Encoding algorithm has an Underflow problem, which gives an erroneous result after few numbers of iterations. Therefore it is not suitable for comparison. Huffman Encoding and Shannon Fano algorithm shows similar results except in compression times. Shannon Fano algorithm has faster compression time than Huffman Encoding, so this factor can be used to determine the more efficient algorithm from these two.
While considering the major performance factors like compression time, decompression time and saving percentages of the all the selected algorithms. The Shannon Fano algorithm is considered as the most efficient algorithm, as the values of this algorithm lies acceptable range and it also shows better results for the large files.

3. CONCLUSION

This study introduces data compression and simple algorithms for compression. Each algorithm has its own advantages and disadvantages. With the help of various performance factors, it is easy to choose algorithms that are more efficient. This paper demonstrates that if we use the right data compression techniques, it will certainly be helpful in reducing the storage space and the computational resources. This is definitely more critical in the case of wireless systems where network bandwidth is always a cause for concern.

REFERENCES

[1] https://en.wikipedia.org/wiki/Lossy_compression.
[2] https://en.wikipedia.org/wiki/Lossless_compression.
[3] John.C.Kieffer and En-hui Yang, "Lossless Data Compression Algorithms Based on Substitution Tables", IEEE,1998.
[4] J.C.Kieffer, E-H.Yang, G.Nelson, and P.Cosman, "Lossless data compression via multi-level pattern matching," IEEE, 1996.
[5] Massimo Vecciho, "A Simple Algorithm for Data Compression in Wireless Sensor Networks", IEEE,June 2008.
[6] https://en.wikipedia.org/wiki/Huffman_coding.
[7] Xi Deng and Yuanyuan Yang, "Online Adaptive Compression in Delay Sensitive Wireless Sensor Networks", IEEE, October 2012.
[8] https://en.wikipedia.org/wiki/Lempel–Ziv–Welch.
[9] S.Kwong and Y.F.Ho , "A Statistical Lempel-Ziv CompressionAlgorithm for Personal Digital Assistant (PDA)", IEEE, February 2001.
[10] S.R.Kodituwakku and U.S.Amarasinghe, "Comparison of Lossless Data Compression Algorithms for Text Data", Indian Journal of Computer Science and Engineering, Vol 1 No. 4 416-425.