A New Way of Accelerating Web by Compressing Data with Back Reference-Prefer Geflochtener

Kushwaha Singh[1], Challa Krishna[2], and Saini Kumar[1]
[1]Department of Computer Science and Engineering, Rajasthan Technical University, India
[2]Department of Computer Science and Engineering, Panjab University, India

Abstract: *This research focused on the synthesis of an iterative approach to improve speed of the web and also learning the new methodology to compress the large data with enhanced backward reference preference. In addition, observations on the outcomes obtained from experimentation, group-benchmarks compressions, and time splays for transmissions, the proposed system have been analysed. This resulted in improving the compression of textual data in the Web pages and with this it also gains an interest in hardening the cryptanalysis of the data by maximum reducing the redundancies. This removes unnecessary redundancies with 70% efficiency and compress pages with the 23.75-35% compression ratio.*

Keywords: *Backward references, shortest path technique, HTTP, iterative compression, web, LZSS and LZ77.*

Received April 25, 2014; accepted August 13, 2014

1. Introduction

Compression is the reduction in data size of information to save space and bandwidth. This can be done on Web data or the entire page including the header. Data compression is a technique of removing white spaces, inserting a single repeater for the repeated bytes and replace smaller bits for frequent characters Data Compression (DC) is not only the cost effective technique due to its small size for data storage but it also increases the data transfer rate in data communication [2].

Compression held in two categories: lossless compression and lossy compression [3, 4]. Lossless compression reforms a compressed file similar to its original form. On the other hand, lossy compression removes the unnecessary data but can't be reproduced exactly. There exist many old and new algorithms for lossless compression which are to be studied e.g., LZ77 [16], LZSS [16], Zopfli [14].

2. Literature Survey

In this section, existing LZ77 variants and compression algorithms are studied.

2.1. LZ77 Compression

In LZ77, the data is changed with the location to single copy in the uncompressed ingressed stream. A match in lookaheadBuffer is to be fixed by the length-distance pairs. To discontinue such matches, the compressor stores recent data in window. With this window begin to fall at the end and progress backward as the compression is preponderated and the window will conclude it's sliding.

Algorithm 1: LZ77 Compression

```
if a sufficient length is matched or it may correlate better with next input.
While (! empty lookaheadBuffer)
{
get a remission (position, length) to longer match from search buffer;
if (length>0)
{
Output (position, length, nextsymbol);
transpose the window length+1 position along;
}
else
{
Output (0, 0, first symbol in lookaheadBuffer);
transpose the window 1 position along;
}
}
```

Recently matched encoded characters are stored in search buffer [16] and remaining part in LookAheadBuffer [1]. But performance declines when the character repeated are larger than the search buffer.

2.2. LZSS Compression

LZSS is based on the dictionary encoding technique. In comparing to LZ77, LZSS omits such references where the dictionary references may be longer than the search buffer. Besides addition of one-bit flag indicates whether the data is a literal (byte) or a referral to an offset/length pair.

Algorithm 2: LZSS Compression.

```
While (! empty lookaheadBuffer)
{
get a pointer (position, match) to the longest match;
```

```
if (length > MINIMUM_MATCH_LENGTH)
{
output (POINTER_FLAG, position, length);
transpose the window length characters along;
}
else
 {
output (SYMBOL_FLAG, first symbol of lookaheadBuffer);
transpose the window 1 character along;
}
}
```

This like LZSS yields a better performance over the LZ77 compression by adding extra flag.

2.3. ItCompression

Following various compression techniques, the main risks revealed in deciding the fine set of representative rows. Reappearing with iterations, previous rows may be relocated by newly represented tuples [13]. Though, the representative rows keep changing, each iteration monotonically improves the quality globally. Moreover, each iteration requires a single scan over the data, guiding to a fast compression scheme.

Algorithm 3: ItCompression.

```
Input: A table T, a user specified value k and an error tolerance
vector e.
    Output: A compressed table Tc and a set of representative
rows P = {P1,…….,Pk}
     Pick a random set of representative rows P
      While totalcov (P, T) is increasing do
{
For each row R in T, find Pmax (R)
 Recomputed each Pi in P as follow:
{
For each attribute XJ
Pi [XJ] = f v (XJ, G(Pi) )
  }
}
```

In this each row R in T is assigned to a representative row $P_{max}(R)$ that gives the most coverage among the members of representative set P. Next, a new set of P is computed. Here, the sliding window of size $2*e_j$ is moved along the sorted micro-intervals to find the range that is most frequently matched. Hence, it is found that by varying the representative rows compression ratio improves but with the increment of rows it reduces the CPU cycles. Hence, after experimenting variations it is considered that it should be limited to 100 for the best as it should neither increase the time slices as well as nor decreasing the compression for our proposed system.

3. Shortcoming

LZSS was introduced as search buffer is much longer than LookAheadBuffer in LZ77 due to which the non-matched pairs waste the space. While considering all theories Google give a new heuristic zopfli. But it still uses the previous length in its output and appends every time the length and distance to the arrays even when the length is less than the shortest matched string that again waste the space. So the modifications are proposed that save the space and use recent length for better compression, which will be described in the section 5.

4. Our Contribution

HTTP compression is the method of the Apache server to provide the better bandwidth and advances the hits over the web [15]. And the browsers supporting mostly two types of encodings namely gzip and deflate [10]. Both the schemes encode the content using LZ77 algorithm. Recently Google gives a Zopfli compression technique Vandevenne [6, 17].

Distinctly the proposed system that is conferred below does not concern about the bareness in the dictionary when the window slides over the data until lookaheadBuffer is full. Therefore, <0, 0, store> is assert to encode the characters in store that does not match in dictionary [8].

On the other hand, lengthscore is introduced for large lengths with the indexing purposes which is a pair of length and distance. This directs the most outstanding sequence from the biggest match for the better proficiency compressing diminutive characters before the bulky ones.

5. Proposed System

The projected system is found on iterative entropy model and a shortest path search to find a low bit cost. Hence, to overcome issues over length/distance in Zopfli as mentioned in section 2.4, it is proposed to evaluate the lengthscore itself in place of the earlier lengths in tuple. Also, it considers only the single perspective of LZ77 store for the huge matchless characters to affix the length and distance to an array.

A compressor whose algorithm is discussed in section 5.1 and a cleaner to flush the compressed streams are proposed in the system. When the user query, the server is being tried to lookup for pages and collects the data without any additional setup. So when these fetched pages are being prepared to answer they pass through the proposed system where the data being compressed in a format attuned to the browser and then place over the Internet. Hence, when the Internet acknowledged with the HTTP “OK” message, the system decide to clean the storage. The complete scenario is illustrated in Figure1.

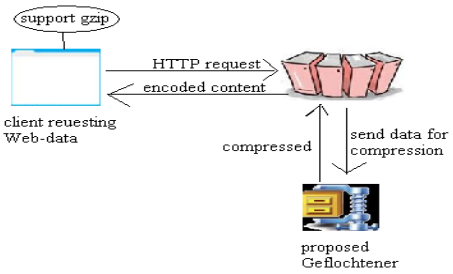

Figure 1. Scenario of the proposed system.

5.1. Proposed BRP Geflochtener Compression

Algorithm 4: Proposed BRP Geflochtener.

```
for (i=instart; i< inend; i++)
{
      Maximum amount of blocks to split into 100;
        Update the sliding hash value;
        Find the longest matched cache;
        Gets a score of the length given the distance;
       if (lengthscore >= MIN_MATCH)
 {
Verifies if length and dist are indeed valid, assert;
output StoreLitLenDist(lengthscore, dist, store);
shift the window length characters along;
 }
Else
{
 output StoreLitLenDist(0, 0, store);
       shift the window 1 character along;
            }
}
```

In this algorithm, instart is the preliminary position of the window, inend is the ending position of the window size, Litlen contains the literal symbols or length values i.e., literal per length, lengthscore is the length itself, dist indicates the distance and MIN_MATCH is the shortest distance in length. Literal symbols and the distance both are about the equivalent size.

5.2. Compression Process

The Litlen distance producer transforms data into literals with the particulars of their length and distance. Next these literals and lengths exceed through scorer where each literal get the score on the basis of the distance. Now these would be authenticated before the matching of the literals. When they authenticated then they pass to the iterator where the bits are evaluated to get the longest match from the backward references. With this the longest matched bits are cached into a Longest Matched Cache (LMC) and applying the shortest paths technique with the best length first. Then, the lengthscore is verified and clears the length. Now swing the window slider for the next matches. Similarly, the matched pairs after traversing all the paths pass on the matched phrases to the entropy encoder which are similar to the Huffman tree bits to value of symbols and acquire down to compressed stream which will be spread over Internet with the header bits situated to content encoding gzip.

5.3. Compression Strength

For detailed analysis, let us presume the input stream as "0123456789" at very begin so it has not establish any backward references. When the window slides the next stream hits "0123456789" then it disperses the distance and length as -10 where '-' illustrates the back movement. Similarly when it hits with "0000056789", it has various choice for the references to encode 0's with distance and length as -(1) -1 and 4 (2) -10 and 5 respectively. Next when "0003456789" comes into follow then to encode '000' it gets back matched with distance -8 and length 3; distance -9 and length 3 and distance -10 and length 3 each with diverse probable tuple. Likewise, the prime of distances are being measured which is more probable statistically and leads to smaller entropy.

5.4. Algorithmic Analysis

Proposed Geflochtener compresses the data by encoding phrases from lookahead buffer as references in sliding window so that the lookahead buffer is laden with the symbols. Here, the current bytes are tracked in instart and inend is used to keep trail of the current byte writing to buffer of the compressed data. While the longest matched cache LMC identified to find longest match and revisit the length of it. Now LMC places the offset subsequently to the symbol in the look-ahead buffer immediately behind the match. Considering the situation, the window size is kept 32 KB concluding how far back corresponding phrases is explored in the data and the length limits to 258 that locate the undersize distances. Hence, only 256 out of 259 are used for the handiness of array which would make 3 longer. Generally it is good scheme to search far back for matchings but it must be balanced beside search time through sliding window. Also, balance it against the space penalty by using additional bits for offsets. However, when the data has many repeated long phrases then choose buffer size such that excessively diminutive marks in multiple phrase tokens get just one. The network function htonl is applied to convert the token in big-endian format. This is the format required to store the compressed data on top of uncompressed data.

Since our technique is based on shortest path, cost model taking entropy into account and the iterations for optimum search. To encode efficiently, we have improved the Back Reference Prefer (BRP) as shown in Figure 2 which is based on Huffman encodings. Hence, BRP select bits cost of entropy encoding based on given formulae:

$$BR(pref) = log\ 2\ [tot\ (pref)\ ] - bits\ [non\_huffman\ (pref)\ ] \quad (1)$$

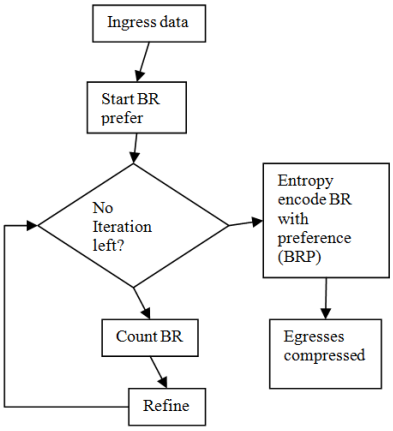

Figure 2. Mathematical model of BRP.

Where *tot*(*pref*) represents number of *BR*, log_2 [*tot*(*pref*)] evaluate bit cost of entropy encoding and *bits*[*non_huffman*(*pref*)] is number of non-entropy encoded bits. The resulted *BR*(*pref*) selects the best order to encode.

6. Implementation

The proposed algorithm does not require any update at the client-side applications. It is a good approach and viable tool for cutting the cost from heavy traffic websites. The proposed system is employed in C with the concern for its lenience and compatibility over the different platforms. Here, for the sake of inbuilt libraries, run it on Red Hat Enterprise Linux operating system with kernel 2.6.18-128 (x86_64) on Intel Pentium Dual core CPU E2160 at 1.80 GHz. Also the complete source code has been compiled on GNU Compiler Collection (GCC) version 4.6.3 with a single walled output for the portability, so that it can be directly used anywhere without any pre-configuration. And the benchmarks are decided on the basis of their contented characteristics as such Calgary composed of collection of small text with some binary files and Enwik8 which stores 100 million bytes of English Wikipedia large content which are the best for our testing reasons as they have all the essential content that are always seen in websites while transmitting on the HTTP. The only compression libraries are declared in it, existing software can be used for their decompression. This provides better functionality with gzip, deflate and compatible with all browsers.

7. Outcomes of Experiment

All the backward offsets are tracked including even those which has no backward references are there and then choose from it that generates the shortest amount of bits. Every length is gathered and finest sequence is preferred by reverse traversal of the buffer according to the given formula in section 5.4. To experimental proving corpora's used: Calgary corpus [5], Canterbury corpus [7] and enwik8 [9] which are illustrated in Table 1.

Table 1. Comparison of compression by proposed BRP Geflochtener with the existing compressors.

Benchmarks	Corpus Size (KB)	Gzip-9 (KB)	7Zip (KB)	Kzip(KB)	Proposed BRP Geflochtener (KB)
Calgary	3141622	1017624	980674	978993	974067
Canterbury	2818976	730732	675163	674321	668456
Enwik8	100000000	36445248	35102976	35025767	34986660

The results are evaluated in terms of compression percentage as per formulae: compression percentage $_{(CP)} = \frac{LO - LC}{LO} \times 100$, where LO: Length of Original text and LC: Length of Compressed text.

On the converse, it is also estimated Compression Gain (cg) which can be defined as the amount of space recovered as a result of compression and can be calculated by:

$$cg = 100 - \left\langle \frac{compressedfilesize}{originalfilesize} \times 100 \right\rangle \quad (2)$$

Where original file size and compressed file size should be in same unit (bits, bytes, Kbytes, Mbytes, etc.,) as shown in Figure 3. Hereby, it is clear that proposed BRP Geflochtener although crossholding to conventional Kzip and 7-zip compressors but it yields better compression gain than Gzip-9 which is mostly supported by present day browsers for content-encoding. As from the Table 1, it is depicted that the proposed Geflochtener has eliminated about 69% redundancy in Calgary which is 1.6% greater than that of gzip-9; likewise for Canterbury it removes 76.22% which is 2.2% better and for English Wikipedia 65% which is 1.5 % greater than that of gzip-9. This yields an outstanding change in compression by removing the redundancies of data about an average of 70%.

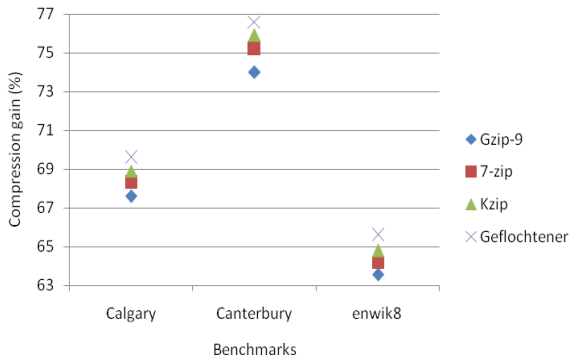

Figure 3. Effect on compression gain using the proposed system in Web traffic.

Based on the above formulae the performance calculated is shown by graphically in Figure 4 in which lowest blue line of Geflochtener proves the highest compressibility among the existing compressors.

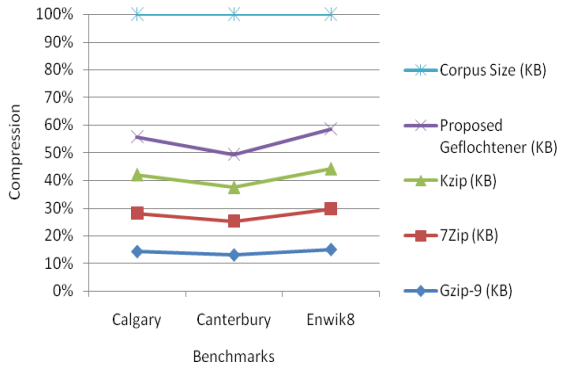

Figure 4. Effect on compression using proposed system in the Web.

As discussed, after compressing the data it is needed to send it over Internet whose transmission time(s) are recorded (while transmitting the compressed Web data over 10 MBps speed connection) in Table 2 based on which it proves the effect of proposed Geflochtener in accelerating transmission graphically in Figure 5.

Table 2. Transmission time of proposed BRP Geflochtener over existing compressors.

Benchmarks	Gzip-9(ms)	7Zip(ms)	Kzip(ms)	Proposed BRP Geflochtener(ms)
Calgary	99.4	95.8	95.6	95.2
Canterbury	71.4	65.9	65.9	65.3
Enwik8	3559.1	3428	3420.5	3417.6

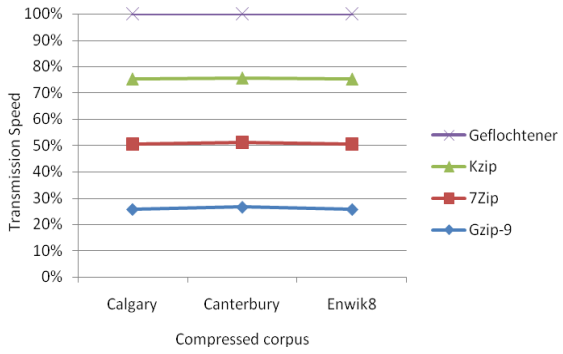

Figure 5. Effect on throughput using the proposed system in web traffic.

8. Results and Discussion

Frequently preceding lengths are taken into account while obtaining the length scores, this removes the overhead of deciding the length from where it starts next comparison. And the output fabricated is 4.0-8.52% smaller than that of gzip-9 and save the space on server with the deprivation of 514 to 9086 bytes.

Over 10 Mbps speed connection, the proposed Geflochtener (with the improvement) transmit Enwik8 content over web in 3 second, Calgary content in 95 milliseconds, and Canterbury content in 65milliseconds which is much significant in accelerating our web traffic.

The proposed system's strength for binary blobs that changes infrequently, if ever, or are downloaded with enough frequency to increase download speed. This also helps in reducing battery usage in mobile networks and less strain on subscriber's data plan. Not only this, but it also proved its worth by implementing a stand-in in IDE to create a compact distributable APK files for the users.

As the redundancy brings the vulnerability for cryptanalysis, our Geflochtener compression makes such cryptanalysis harder by reducing the redundancies densely which would be a great benefit while transmitting the encrypted confidential contents after compression over Internet like in Email services.

Henceforth, it is proven to be better phenomenon with a little more compression in not only wired network but also in common wireless spectrum where the mobile data transfers lead to raising the cost to implementation levels. This can further be improved by implementing threading in program to run concurrently [11, 12] and also scope of making it purely online. Also it needs to decide when the cleaner should run which will still remains the question of discussion.

References

[1] Arya G., Singh A., Painuly R., Bhadri S., and Maurya S., "LZ squeezer A Compression Technique based on LZ77 and LZ78," *The SIJ Transactions on Computer Science Engineering and its Applications*, vol. 1, no. 1, pp. 29- 32, 2013.

[2] Akman I., Bayindir H., Ozleme S., Akin Z., and Misra S., "Lossless Text Compression Technique Using Syllable Based Morphology," *The International Arab Journal of Information Technology*, vol. 8, no. 1, pp. 66-74, 2011.

[3] Awan F. and Mukherjee A., "LIPT: A Lossless Text Transform to Improve Compression," *in Proceeding of International Conference on Information Technology: Coding and Computing*, Las Vegas, pp. 452-460, 2001.

[4] Burrows M. and Wheeler D., *A Block-Sorting Lossless Data Compression Algorithm*, Digital Systems Research Centre, 1994.

[5] Calgary corpus, http://www.data-compression.info/Corpora/CalgaryCorpus/index.htm, Last Visited 2014.

[6] Vandevenne L., http://googledevelopers.blogspot.in/2013/02/compress-data-more-densely-with-zopfli.html Googledevelopers.blogspot.com, Last Visited 2014.

[7] Canterbury Corpus., http://corpus.canterbury.ac.nz/resources/cantrbry.zip, Last Visited 2014.

[8] Data Compression the Dictionary Way, http://www.i-programmer.info/babbages-bag/515-data-compression-the-dictionary-way.html?start=1, Last Visited 2014.

[9] Enwik8 Corpus, http://mattmahoney.net/dc/enwik8.zip, Last Visited 2014.

[10] Fielding R., Gettys J., Mogul J., Frystyk H., Masinter L., Leach P., and Berners-Lee T., "RFC2616-Hypertext Transfer Protocol-HTTP/1.1," 1999.

[11] Gilchrist J. and Cuhadar A., "Parallel Lossless Data Compression using on the Burrows-Wheeler Transform," *International Journal of Web and Grid Services*, vol. 4, no. 1, pp. 117-135, 2008.

[12] Gilchrist J. and Cuhadar A., "Parallel Lossless Data Compression Based on the Burrows-Wheeler Transform," *in Proceeding of Advanced Information Networking and Applications*, Niagara Falls, pp. 877-884, 2007.

[13] Jagadish H., Ng R., Ooi B., and Tung A., "ItCompress: an Iterative Semantic Compression Algorithm, Data Engineering," *in Proceeding of 20th International Conference on Data Engineering*, Boston, pp. 646-657, 2004.

[14] Jyrki A. and Lode V., Data compression using Zopfli, https://zopfli.googlecode.com/files/Data_compression_using_Zopfli.pdf, Last Visited 2014.

[15] Microsoft Corporation, http://www.microsoft.com/technet/prodtechnol/WindowsServer2003/Library/IIS/d52ff289-94d3-4085-bc4e-24eb4f312e0e.mspx?mfr=true, Last Visited 2014.

[16] Shanmugasundaram S. and Robert L., "A Comparative Study of Text compression Algorithms," *International Journal of Wisdom Based Computing*, vol. 1, no. 3, pp. 68-76, 2011.

[17] Zopfli Compression Algorithm-Google Project Hosting, https://code.google.com/p/zopfli/downloads/list, Last Visited 2014.

Satpal Kushwaha is an Associate Professor, at MITRC, Alwar (Rajasthan). He has done his M.Tech. from RTU, Kota, B.E. from University of Rajasthan, Jaipur. He has 8 years of teaching and research experience. His research interests are Information Security, Network Security and Big Data.

Rama Challa is Professor, at NITTTR, Chandigarh. He has done his Ph.D. from IIT Kharagpur, M.Tech. from CUSAT, Cochin and B. Tech from JNTU, Hyderabad. He has 18 years of teaching and research experience. His research interests are Wireless Networks, Distributed Computing, Cryptography, and Network Security.

Hemant Saini is pursuing M. Tech in Computer Science and Engineering from Rajasthan Technical University, Kota. He is a Red hat Certified Engineer. He has completed his B. Tech in Information Technology from MLV Government Textile and Engineering College. He is having 2 years of industrial experience and one year of academic experience. His research interests are Computer Network and Security, Wireless Networks.