# Visually Lossless HTML Compression

Przemysław Skibiński

University of Wrocław, Institute of Computer Science,
Joliot-Curie 15, 50-383 Wrocław, Poland
inikep@ii.uni.wroc.pl

**Abstract.** The verbosity of the Hypertext Markup Language (HTML) remains one of its main weaknesses. This problem can be solved with the aid of HTML specialized compression algorithms. In this work, we describe a visually lossless HTML transform that, combined with generally used compression algorithms, allows to attain high compression ratios. Its core is a transform featuring substitution of words in an HTML document using a static English dictionary, effective encoding of dictionary indexes, numbers, and specific patterns.

Visually lossless compression means that the HTML document layout will be modified, but the document displayed in a browser will provide the exact fidelity with the original. The experimental results show that the proposed transform improves the HTML compression efficiency of general purpose compressors on average by 21% in the case of gzip, achieving comparable processing speed. Moreover, we show that the compression ratio of gzip can be improved by up to 32% for the price of higher memory requirements and much slower processing.

**Keywords:** HTML compression, HTML transform, semi-structural data compression.

## 1 Introduction

Since the origin of World Wide Web, the Hypertext Markup Language (HTML) is a standard for Internet web pages. HTML has many advantages. One of its main advantages is that it is a textual format, what means that HTML is human-readable and can be edited by any text editor. The textual format of HTML is also one of its main disadvantages as it introduces verbosity. Nevertheless verbosity can be coped with by applying data compression.

In this work we will present our specialized algorithm for HTML compression, which achieves much higher compression than the state-of-the-art LSHT algorithm. It also uses a fixed English dictionary. The algorithm is designed in four variants for proper general-purpose algorithms.

The map of the paper is as follows. Section 2 contains a short review of existing HTML compression methods thus putting our work in a proper context. In Section 3 we describe step-by-step our HTML transform, its main ideas, and its most significant details. The next section presents details about optimizations for back-end

G. Vossen, D.D.E. Long, and J.X. Yu (Eds.): WISE 2009, LNCS 5802, pp. 195–202, 2009.

compression algorithms used with our transform. Section 5 contains implementation details, description of files used for experiments, and experimental results. Section 6 gives our conclusions.

## 2 Related Work

HTTP compression [11] is the technology used to compress contents from a web server (an HTTP server) and to decompress them in an user's browser. HTTP compression is a recommendation of the HTTP 1.1 protocol specification as it reduces network traffic and improves page download time on slow networks [12]. It is especially useful when size of the web pages is large.

The popular LZ77-based gzip was intended to be the HTTP compression algorithm. Currently, HTTP servers and clients supports also LZ77-based deflate format. Lighttpd server supports also BWT-based bzip2 compression, but this format is only supported by lynx and some other console text-browsers. Deflate, gzip, and bzip2, however, are general-purpose compression algorithms and much better results can be achieved with a compression algorithm specialized for dealing with HTML documents.

In our previous work [14] we have presented the first two specialized HTML compression algorithms called SDHT and LSHT. These methods are related to text compression [16] as they use respectively a semi-static dictionary of the most frequent words in the document or a static English dictionary. These methods are also related to XML compression [17] as both formats, HTML and XML, are SGML-based. SDHT and LSHT use several techniques from our XML compressor [17] as e.g., effective encoding of numbers, dates and times.

## 3 HTML Transform

In this section we present our Visually Lossless Static HTML Transform (VLSHT). We introduce subsequent parts of our algorithm step by step.

### 3.1 End Tag Encoding

In a well-formed HTML document, every end tag must match a corresponding start tag. This can hardly be exploited by general-purpose compression algorithms, as they maintain a linear, not stack-alike data model. The compression ratio can then be increased by replacing every matching end tag with merely an element closing flag.

Our transform puts elements on a stack when a start tag has appeared. The last inserted element is removed from a stack when an end tag has appeared. The problem with HTML is that not all elements must have a closing tag. It can be solved by ignoring elements that allow an end tag omission. The second problem with HTML is that some tags (e.g. <p>) should have corresponding end tags, but human editors skip these closing tags. Moreover, web browsers do not report errors on documents of this

kind. Therefore our transform allows non-valid HTML documents. The above-mentioned problems do not occur in XHTML.

### 3.2 Quotes Modeling

Attributes of HTML elements usually contain neighboring *equal* and *quotation mark* characters (e.g. `attribute="value"`). Sometimes attributes are encoded using *equal* and *apostrophe* characters (e.g. `attribute='value'`). We have found that replacing these two characters with a flag improves compression performance. We made the same with *quotation mark* and *angle right bracket* (*greater*) characters that closing start tags with attribute(s) (e.g. `<element-name attribute="value">`).

### 3.3 Number Encoding

Numbers appear very often in HTML documents. We found that storing numbers as text is ineffective. Numbers can be encoded more efficiently using a numerical system with base higher than 10.

In our transform every decimal integer number n is replaced with a single byte whose value is $\lceil \log_{256}(n+1) \rceil + 48$. The actual value of n is then encoded as a base-256 number. A special case is made for sequences of zeroes preceding a number – these are left intact.

Our transform encodes in a special way also other numerical data that represent specific information types. Currently our transform recognizes the following formats:

- dates between 1977-01-01 and 2153-02-26 in YYYY-MM-DD (e.g. “2007-03-31”, Y for year, M for month, D for day) and DD-MMM-YYYY (e.g. “31-MAR-2007”) formats;
- years from 1900 to 2155 (e.g. “1999”, “2008”)
- times in 24-hour (e.g., “22:15”) and 12-hour (e.g., “10:15pm”) formats;
- value ranges (e.g., “115-132”);
- decimal fractional numbers with one (e.g., “1.2”) or two (e.g., “1.22”) digits after decimal point.

### 3.4 Static Dictionary

Our transform uses a static dictionary, which is embedded in the compressor and the decompressor. There are two advantages of a static dictionary over a semi-dynamic dictionary: there is no need to make the first pass over the input data to create the semi-dynamic dictionary and there is no need to store the semi-dynamic dictionary within processed data to make decompression possible.

On the other hand a static dictionary is limited to some class of documents e.g. English language. The dictionary must be spread with the compressor and the decompressor. Moreover, a semi-dynamic dictionary contains words that are actually frequent in the document, not words that could potentially be frequent, as it is in the case of a static dictionary. Nevertheless according to [14] for HTML documents a static English dictionary usually gives a better compression ratio than a semi-dynamic dictionary.

### 3.5 Visually Lossless Optimization

HTML documents are usually displayed in a browser. The browser ignores the document layout, therefore tabulators, end of line symbols, and runs of spaces can be replaced with a single space symbol or even removed. The one exception are <PRE> and <STYLE> tags, which should be left intact. We call this technique visually lossless optimization. This technique is well-known from HTML optimizers.

From the other side visually lossless compression is lossy. The lossy compression means in this case that layout, which may be useful for human editors of a document will be modified. Moreover the exact fidelity of the decompressed document with the original required in order to verify the document integrity using a cyclic redundancy check or hash functions will not be supported. If we can accept these disadvantages we will get a high improvement from visually lossless compression.

## 4 Back-End Compression

Succinct word encoding appears to be the most important idea in Visually Lossless Static HTML Transform (VLSHT). There are four modes of encoding, chosen depending on the attached back-end compression algorithm: LZ77-based [20], LZMA/BWT-based [3], PPM-based [5], and PAQ-based [10]. In all cases, dictionary references are encoded using a byte-oriented prefix code, where the length varies from one to four bytes. Although it produces slightly longer output than, for instance, Huffman coding [7], the resulting data can be easily compressed further, which is not the case with the latter. Obviously, more frequent words have assigned shorter codewords.

The LZ77 and PPM optimized transform contain of the biggest possible alphabet for codewords: byte values from 128 up to 255 and most values in range 0–31, plus a few more. These symbols are very rarely used in most HTML documents. If, however, one of these symbols occurs in the document, and is not part of an encoded word, the coder marks it with a special escape symbol. In the LZMA/BWT and PAQ optimized transform the codeword alphabet consists of fewer symbols, because it uses only 128 symbols with byte values from 128 up to 255.

## 5 Experimental Results

This section presents implementation details of the VLSHT algorithm. It also contains description of files used for experiments and discussion on experimental results of the VLSHT algorithm with four different back-end compression methods.

### 5.1 Implementation Details

The VLSHT implementation contains a fast and simple HTML parser built as a finite state automaton (FSA), which accepts proper words and numerical (including date and time) expressions. The parser does not build any trees, but treats an input HTML document as one-dimensional data. It has small memory requirements, as it only uses

a stack to trace opening and closing tags. The parser supports the HTML 4.01 specification (e.g. allowed an end tag omission for some tags).

The VLSHT implementation uses a static English dictionary with about 80.000 words. In this dictionary, words are sorted with the relation to their frequency in a training corpus of more than 3 GB English text taken from the Project Gutenberg library. The words are stored in lower case as VLSHT implements the capital conversion method to convert the capital letter starting a word to its lowercase equivalent and denote the change with a flag. Additionally, VLSHT uses another flag to mark a conversion of a full uppercase word to its lowercase form.

VLSHT is lossy in a sense that the HTML document layout will be modified, but the document displayed in a browser will provide the exact fidelity with the original. The transforms can handle any HTML documents with 8-bit (ISO-8859 and UTF-8) or 16-bit (Unicode) encodings. VLSHT was implemented in C++ and compiled with MS Visual C++ 2008.

### 5.2 HTML Corpus

In compression benchmarking, proper selection of documents used in experiments is essential. To the best of our knowledge, there is no publicly available and widely respected HTML corpus to this date. Therefore, we have based our test suite on HTML files (without images, etc.) downloaded from common Internet web sites. The resulting corpus represents a wide range of real-world HTML documents.

Detailed information for each group of the documents is presented in Table 1; it includes: URL address, number of files and total size of files. The size of a single file spans from 1 up to 296 KB.

**Table 1.** Basic characteristics for the HTML corpus used in the experiments

| *Name* | *URL address* | *no. files* | *Total size* |
|---|---|---|---|
| Hillman | hillmanwonders.com | 781 | 34421 KB |
| Mahoney | www.cs.fit.edu/~mmahoney/ | 11 | 596 KB |
| MaxComp | maximumcompression.com | 61 | 2557 KB |
| STL | www.sgi.com/tech/stl/ | 237 | 2551 KB |
| TightVNC | tightvnc.com | 21 | 289 KB |
| Tortoise | tortoisesvn.net | 393 | 5342 KB |
| Travel | travelindependent.info | 69 | 3841 KB |

### 5.3 Compression Ratio

The primary objective of experiments was to measure the performance of our implementation of the VLSHT algorithm. For comparison purposes, we included in the tests general-purpose compression tools: gzip 1.2.4, LZMA 4.43, PPMVC 1.2, and FastPAQ8, employing the same algorithms at the final stage of VLSHT, to demonstrate the improvement from applying the HTML transform.

**Table 2.** Compression results for HTML datasets in output bits per input character

| | HufSyl | LZWL | gzip | StarNT+gzip | WRT+gzip | mPPM |
|---|---|---|---|---|---|---|
| Hillman | 2.95 | 2.13 | 1.51 | 1.42 | 1.44 | 1.34 |
| Mahoney | 3.31 | 3.23 | 2.72 | 2.54 | 2.49 | 2.30 |
| MaxComp | 3.03 | 2.39 | 1.86 | 1.79 | 1.80 | 1.55 |
| STL | 3.48 | 3.22 | 2.19 | 1.97 | 1.95 | 2.31 |
| TightVNC | 3.44 | 3.26 | 2.34 | 2.17 | 2.13 | 2.24 |
| Tortoise | 3.37 | 3.13 | 2.27 | 2.08 | 2.06 | 2.23 |
| Travel | 2.88 | 2.72 | 2.34 | 2.06 | 1.97 | 1.95 |
| Average | 2.808 | 2.510 | 1.904 | 1.754 | 1.730 | 1.740 |
| Improvement | | | | 7.88% | 9.13% | |

| | gzip | LSHT +gzip | VLSHT +gzip | LZMA | LSHT +LZMA | VLSHT +LZMA |
|---|---|---|---|---|---|---|
| Hillman | 1.51 | 1.23 | 1.17 | 1.29 | 1.13 | 1.07 |
| Mahoney | 2.72 | 2.26 | 2.17 | 2.35 | 2.08 | 2.00 |
| MaxComp | 1.86 | 1.47 | 1.45 | 1.53 | 1.38 | 1.36 |
| STL | 2.19 | 1.85 | 1.74 | 2.13 | 1.79 | 1.69 |
| TightVNC | 2.34 | 2.09 | 1.95 | 2.23 | 1.99 | 1.86 |
| Tortoise | 2.27 | 2.02 | 1.93 | 2.17 | 1.92 | 1.84 |
| Travel | 2.34 | 1.84 | 1.69 | 2.13 | 1.74 | 1.60 |
| Average | 1.904 | 1.595 | 1.513 | 1.729 | 1.504 | 1.428 |
| Improvement | | 16.22% | 20.55% | | 13.02% | 17.43% |

| | PPMVC | LSHT +PPMVC | VLSHT +PPMVC | FPAQ | LSHT +FPAQ | VLSHT +FPAQ |
|---|---|---|---|---|---|---|
| Hillman | 1.19 | 1.06 | 1.00 | 1.14 | 1.01 | 0.96 |
| Mahoney | 2.09 | 1.92 | 1.86 | 2.01 | 1.83 | 1.77 |
| MaxComp | 1.41 | 1.30 | 1.28 | 1.36 | 1.24 | 1.22 |
| STL | 1.91 | 1.71 | 1.62 | 1.90 | 1.67 | 1.58 |
| TightVNC | 1.96 | 1.86 | 1.76 | 1.96 | 1.82 | 1.71 |
| Tortoise | 1.93 | 1.83 | 1.76 | 1.92 | 1.77 | 1.70 |
| Travel | 1.79 | 1.60 | 1.49 | 1.79 | 1.56 | 1.44 |
| Average | 1.535 | 1.410 | 1.346 | 1.510 | 1.363 | 1.298 |
| Improvement | | 8.14% | 12.30% | | 9.77% | 14.07% |

As we are aware of just two specialized algorithms for HTML compression [14] we have also compared our algorithm to well-know word-based text compression techniques: StarNT [18], WRT [16], HufSyl [8], LZWL [8], and mPPM [1]. We have also tried to use XMLPPM [4] and SCMPPM [2], which work well with XHTML files, but it do not support HTML files.

The first part of Table 2 contains results of word-based text compression algorithms. For each program and group of HTML documents a bitrate is given in output bits per input character, hence the smaller the values, the better. The last but one row includes an average bitrate computed for all the seven groups of documents. The last row presents the average improvement of preprocessors for all documents compared to the general purpose algorithms result.

The next parts of Table 2 contain compression results of the introduced HTML corpus using gzip, LZMA, PPMVC, FastPAQ, and our implementation of the VLSHT algorithm combined with gzip, LZMA, PPMVC, and FastPAQ.

VLSHT with gzip achieves compression results better than all word-based text compression algorithms, including a PPM-based mPPM. Compared to the general-purpose compression tools, VLSHT improves compression of the introduced HTML corpus on average by 21% in the case of gzip, over 17% for LZMA, over 12% in the case of PPMVC and about 14% for FastPAQ. Compression and decompression speed of VLSHT in comparison to general-purpose compression algorithms is a little bit lower as there is a need to read a fixed English dictionary. VLSHT, however, allows to read the dictionary only once and processes all HTML documents in one run.

## 6 Conclusions

HTML has many advantages, but its main disadvantage is verbosity, which can be coped with by applying data compression. HTML is usually used in combination with gzip compression, but gzip is a general-purpose compression algorithm and much better results can be achieved with a compression algorithm specialized for dealing with HTML documents.

In this paper we have presented the VLSHT transform aiming to improve HTML compression in combination with existing general purpose compressors. The main disadvantage of VLSHT algorithm is a fixed English dictionary required for compression and decompression.

VLSHT is lossy in a sense that the HTML document layout will be modified, but the document displayed in a browser will provide the exact fidelity with the original. Thanks to the VLSHT transform, however, compression ratio of the introduced HTML corpus was improved by as much as 21% in the case of gzip, 17% for LZMA, 12% in the case of PPMVC and 14% for FastPAQ. VLSHT with FastPAQ gives the best compression effectiveness, which is 32% better than gzip without any transform. Moreover, VLSHT gives over 4% improvement over the state-of-the-art LSHT algorithm for all tested general-purpose compression algorithms.

VLSHT has many nice practical properties. It is implemented as a stand-alone program, requiring no external compression utility, no HTML parser, thus avoiding any compatibility issues.

## References

1. Adiego, J., de la Fuente, P.: Mapping Words into Codewords on PPM. In: Crestani, F., Ferragina, P., Sanderson, M. (eds.) SPIRE 2006. LNCS, vol. 4209, pp. 181–192. Springer, Heidelberg (2006)
2. Adiego, J., de la Fuente, P., Navarro, G.: Using Structural Contexts to Compress Semistructured Text Collections. Information Processing and Management 43(3), 769–790 (2007)

3. Burrows, M., Wheeler, D.J.: A block-sorting data compression algorithm. SRC Research Report 124. Digital Equipment Corporation, Palo Alto, CA, USA (1994)
4. Cheney, J.: Compressing XML with multiplexed hierarchical PPM models. In: Proceedings of the IEEE Data Compression Conference, Snowbird, UT, USA, pp. 163–172 (2001)
5. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Trans. on Comm. 32(4), 396–402 (1984)
6. Deutsch, P.: DEFLATE Compressed Data Format Specification version 1.3. RFC1951 (1996), http://www.ietf.org/rfc/rfc1951.txt
7. Huffman, D.A.: A Method for the Construction of Minimum-Redundancy Codes. In: Proc. IRE 40.9, September 1952, pp. 1098–1101 (1952)
8. Lánský, J., Žemlička, M.: Text Compression: Syllables. In: Proceedings of the Dateso 2005 Annual International Workshop on DAtabases, TExts, Specifications and Objects. CEUR-WS, vol. 129, pp. 32–45 (2005)
9. Mahoney, M.: About the Test Data (2006), http://cs.fit.edu/~mmahoney/compression/textdata.html
10. Mahoney, M.: Adaptive Weighing of Context Models for Lossless Data Compression. Technical Report TR-CS-2005-16, Florida Tech., USA (2005)
11. Nielsen, H.F.: HTTP Performance Overview (2003), http://www.w3.org/Protocols/HTTP/Performance/
12. Radhakrishnan, S.: Speed Web delivery with HTTP compression (2003), http://www-128.ibm.com/developerworks/web/library/wa-httpcomp/
13. Shkarin, D.: PPM: One Step to Practicality. In: Proceedings of the IEEE Data Compression Conference, Snowbird, UT, USA, pp. 202–211 (2002)
14. Skibiński, P.: Improving HTML Compression. To appear in Informatica (2009)
15. Skibiński, P., Grabowski, S.z.: Variable-length contexts for PPM. In: Proceedings of the IEEE Data Compression Conference, Snowbird, UT, USA, pp. 409–418 (2004)
16. Skibiński, P., Grabowski, S.z., Deorowicz, S.: Revisiting dictionary-based compression. Software – Practice and Experience 35(15), 1455–1476 (2005)
17. Skibiński, P., Grabowski, S.z., Swacha, J.: Effective asymmetric XML compression. Software – Practice and Experience 38(10), 1027–1047 (2008)
18. Sun, W., Zhang, N., Mukherjee, A.: Dictionary-based fast transform for text compression. In: Proceedings of international conference on Information Technology: Coding and Computing, ITCC, pp. 176–182 (2003)
19. Wan, R.: Browsing and Searching Compressed Documents. PhD dissertation, University of Melbourne (2003), http://www.bic.kyoto-u.ac.jp/proteome/rwan/docs/wan_phd_new.pdf
20. Ziv, J., Lempel, A.: A Universal Algorithm for Sequential Data Compression. IEEE Trans. Inform. Theory 23(3), 337–343 (1977)