CHAPTER 14

Compression of Unicode Files

PETER FENWICK

14.1 INTRODUCTION

Since the middle 1960s most computers have handled characters using either the 7-bit ASCII or 8-bit EBCDIC character sets and representations. These codes were generally satisfactory for English, less satisfactory for other European languages based on the Roman alphabet, and totally unsuitable for East Asian languages. Various extensions filled the unused 128 characters of the "8-bit" ASCII set or escaped into alternative alphabets using the ASCII "SO—Shift Out" and "SI—Shift In" codes. But there were many such standards and they were generally messy and incompatible.

Unicode [1] represents a concerted effort to develop a unified representation for all known alphabets and ideographic systems. The "canonical" Unicode representation is the 16-bit UCS-2. Other Unicode representations are the UTF-8 recoding, which allows ASCII characters to be represented in 8 bits, but expands others to 2 or 3 bytes and is often used as a distribution format, and UTF-7, which uses 7-bit codes for transmission of Unicode over e-mail and similar systems. Unicode is used in most modern operating systems and programming languages and is a very important development with possible repercussions in areas such as data compressibility.

Because conventional lossless compressors work in octet or byte units while Unicode characters are often represented by several bytes, there is a reasonable suspicion that Unicode compression might differ from that of more conventional files. After a brief overview of Unicode, this chapter will report some work on the compression of UCS-2 and UTF-8 data and especially its behavior with different compressors.

14.2 UNICODE CHARACTER CODINGS

UCS-2 is based on "blocks" of either 128 or 256 characters, with blocks generally allocated to a single language or other special usage. Some examples are shown in Table 14.1. The Unicode Standard lists all of the codes, together with many comments on the design and usage of Unicode

Table 14.1 Examples of Unicode Block Allocation

Code Range	Name	Code Range	Name
U+0000-U+007F	C0 Controls & Basic Latin	U+0F00-U+0FBF	Tibetan
U+0080-U+00FF	C1 Controls & Latin-1 Supplement	U+10A0-U+10FF	Georgian
U+0100-U+017F	Latin Extended-A	U+20D0-U+20FF	Combining Diacritical Marks
U+0370-U+03FF	Greek	U+2200-U+22FF	Mathematical Operators
U+0400-U+04FF	Cyrillic	U+2700-U+27BF	Dingbats
U+0530-U+058F	Armenian	U+3000-U+303F	CJK Symbols and Punctuation
U+0590-U+05FF	Hebrew	U+3040-U+309F	Hiragana
U+0600-U+06FF	Arabic	U+30A0-U+30FF	Katakana
U+0980-U+09FF	Bengali	U+3200-U+32FF	Enclosed CJK Letters and Months
U+0A80-U+0AFF	Gujarati	U+3300-U+33FF	CJK Compatibility
U+0B80-U+0BFF	Tamil	U+4E00-U+9FA5	CJK Ideographs
U+0E00-U+0E7F	Thai	U+AC00-U+D7A3	Hangul Syllables

Table 14.2 Comparison of ASCII Data "Promoted" to Big- and Little-Endian UCS-2

	Byte Order Mark	c	o	d	e
Big-endian	FE FF	00 63	00 6F	00 64	00 65
Little-endian	FF FE	63 00	6F 00	64 00	65 00

and conventions for each language; for now it is enough to recognize that its canonical representation, UCS-2, is a 16-bit code, generally structured according to the language or alphabet. The coding of a UCS-2 character is usually written U+wxyz, where the initial U+ signals a hexadecimal code and the 4 hexadecimal digits wxyz give the 16 bits of representation.

14.2.1 Big-endian versus Little-endian

A byte order mark U+FEFF may be prefixed to the Unicode file to resolve any problems of Byte order if files are moved between "big-endian" and "little-endian" computers. A big-endian file will start with the two bytes FE and FF, while a little-endian file will start with FF FE. Table 14.2 shows the bytes of a file with the letters "code" in the two forms.

14.2.2 UTF-8 Coding

While the 16-bit UCS-2 representation is the preferred representation within a computer, transmission often uses sequences of 8-bit bytes or octets according to the UTF-8 standard (UCS Transmission Format 8-bit) as in Table 14.3. Each letter in this table represents a single bit.

A standard ASCII character is emitted "as is" in UTF-8 with a high-order 0 bit. Larger values are broken into 6-bit groups, from the least significant bit. Each group except the most significant is prefixed by the bits "10" and emitted as a byte. The first byte starts with as many 1's as there are bytes in the code, followed by a 0 (a unary code). While UTF-8 can handle 32-bit characters, only 2- and 3-byte codes are used for UCS-2 characters.

Table 14.3 UCS-2 and UTF-8 Coding

Data Bits	Input Bit Pattern (UCS-2)	Coding into Successive Bytes (UTF-8)
7	0000 0000 0abc defg	0abc defg
11	0000 0abc defg hijk	110a bcde 10fg hijk
16	abcd efgh ijkl mnop	1110 abcd 10ef ghij 10kl mnop

A text string in UCS-2 may, when converted to UTF-8,

1. contract if it contains mostly ASCII characters,
2. remain essentially the same size if it is mostly Cyrillic, Hebrew, or Arabic, or
3. expand if it is predominately some Asian alphabet.

14.3 COMPRESSION OF UNICODE

A Unicode file is a sequence of characters in either UCS-2 or UTF-8 representation. On disk, these files are held as a sequence of bytes and can be compressed by any byte-oriented lossless compressor. Breaking the characters into their component bytes often degrades the compression from what might be expected. As this effect depends on the actual compressor, it is necessary to consider some of the main classes of compressor and how they might behave. The discussion assumes ASCII data, expanded to 16-bit UCS-2 symbols.

14.3.1 Finite-Context Statistical Compressors

These compressors, exemplified by traditional PPM compressors [2, 5], predict a symbol from a context of say the previous 4 symbols. That means 4 *bytes*, which in UCS-2 is only 2 *characters*. The compressor is then working at only half the expected order and may be expected to achieve rather poorer compression. While it is often possible to increase the order, that all too easily leads to a combinational explosion. While order 4 with 8-bit ASCII has a total of 4.3×10^9 possible contexts, order 4 with 16-bit Unicode has 1.8×10^{19} possible contexts. While no compressor will actually use all of these possible contexts, it must provide data structures to allow them, at least in some dense groups of contexts. There is no guarantee that a given implementation will permit this. With UTF-8 the context is difficult to determine. With English files it is probably unchanged, with European files it may be halved, while with Asian languages it may be reduced to one-third of that expected. (But with ideograms being equivalent to several letters, the compressor works at a much higher effective order than it would with letters and the degradation might be rather less.)

14.3.2 Unbounded-Context Statistical Compressors

These compressors, especially PPM* [6] and Burrows–Wheeler block sorting [4, 7], resemble their finite-context brethren, but have data structures or other techniques which allow contexts to grow indefinitely large. These compressors may be expected to adjust to the longer byte-wise contexts of Unicode by increasing the context order as needed. The operation may be slowed, but compression should remain good. Similar remarks apply to UTF-8 files.

14.3.3 LZ-77 Compressors

These have a sliding window or its equivalent of recent text and emit pointers into the window giving {`phrase_displacement, phrase_length`} couples. With a byte-oriented compressor, the UCS-2 symbols force both the displacement and the length to cover only half the expected number of symbols. Both of these effects reduce the compression, but only slightly. Byte-oriented UTF-8 may or may not have reduced coverage; the compressor can always detect character boundaries (and strictly speaking comparisons should always recognize the UTF-8 symbol alignment).

14.4 TEST COMPRESSORS

These tests use a variety of compressors and compression methods. While not all are state of the art, all but one are widely available and provide a good coverage of compression techniques. All are used with default options and parameters.

GZIP: The Unix compressor, released by the Free Software Foundation, is used as a good example of an LZ-77 class compressor.

BZIP: Seward's implementation of the Burrows–Wheeler block sorting compressor, also released by the Free Software Foundation [8]. (Note that this work used the original BZIP, rather than the later BZIP2.)

COMP-2: A PPM compressor described by Mark Nelson [3].

Compress: The well-known and traditional Unix compressor, implementing the LZW derivative of an LZ-78 compressor.

LZU: This is a special LZ-77 compressor, designed to operate in both 8-bit (LZU-8, for ASCII or UTF-8) and 16-bit (LZU-16, for UCS-2) modes and intended to compare similar compressors for the two modes. Although really designed only for UCS-2 compression, the 16-bit mode is tried for all files. The compressor is not optimized for good compression nor does it recognize byte order marks.

14.4.1 The Unicode File Test Suite

The Unicode file test suite is a multilingual collection of sample Unicode files gathered from various Unicode sources. The total length is about 85,000 characters, of which about 53,500 or 63% are ASCII. Other files have been simulated by expanding three text files from the Calgary Corpus, to provide a control of known text files.

The files are handled by a variety of techniques:

- All of the files are derived from UTF-8 (or ASCII) originals, expanded to UCS-2.
- The UTF-8 original files are compressed as bytes.
- The files, both UCS-2 and UTF-8, are compressed as bytes, with no regard to the 16-bit structure.
- The UCS-2 files are split into two component files, one containing the more significant bytes of each UCS-2 character and one containing the less significant bytes. The two components are individually compressed with a byte compressor and the total size of the two compressed files is combined. (For ASCII files one component is identical to the original and the other is all-zero.)

The important point is how a given compressor behaves on different versions of the same file rather than the relative performance of the compressors on each file—the differences *between* compressors are well known.

Data compression results are traditionally given in output bits per input byte, or bits per byte, with a recent move to give performance in bits per character (bpc). As Unicode no longer retains the identity between bytes and characters, the procedure in this paper is to present all results in bits per character, related to the 16-bit units of the UCS-2 files. Thus ASCII files are shown as bits per byte, while Unicode results are always in bits per Unicode character. The compression of UTF-8 files, considered as arbitrary files without respect to their encoded information, is a quite different matter, which is left until Section 14.6.

14.5 COMPARISONS

Results for the different compressors and file formats are shown in Table 14.4. In all cases the results are simply given as bits per character, averaged over each corpus. (The compression for UCS-32 often looks poor, but remember that it is given in bits per *two* bytes and the value must be halved for comparison with the traditional "bits per byte".)

- The "split" files are not successful, because the division destroys too much contextual information.
- The constant-order PPM compressor shows the expected degradation on the UCS-2 files.
- Unix Compress also gives poor performance on the UCS-2 files.
- The two 8-bit LZ-77 compressors, GZIP and LZU-8, both give a small deterioration in UCS-2 compared with UTF-8.
- The 16-bit LZ-77 compressor LZU-16 gives quite different results for the different "endian" files, emphasizing the need to get the alignment correct, even in the full 16-bit character mode.

Table 14.4 Compression Results, in Bits per Character

		UTF-8	UCS-2 Big-endian	UCS-2 Little-endian	Split
COMP2 order-4	ASCII	2.38	2.95	2.95	2.38
Constant order PPM	Unicode	5.47	5.96	6.06	6.12
BZIP	ASCII	2.30	2.31	2.31	2.31
Burrows–Wheeler	Unicode	5.03	5.24	5.39	5.79
Compress	ASCII	3.66	4.55	4.55	3.72
LZ-78 or LZW	Unicode	7.87	7.91	7.30	7.30
GZIP	ASCII	2.67	3.25	3.25	2.68
LZ-77	Unicode	5.69	6.29	6.32	6.13
LZU-8	ASCII	3.24	4.30	4.30	3.24
LZ-77, 8-bit mode	Unicode	6.76	7.61	7.72	7.39
LZU-16	ASCII	4.21	3.39	3.65	4.20
LZ-77, 16-bit mode	Unicode	8.60	6.54	7.97	9.04

Table 14.5 Compression of UTF-8 Files (in Bits per Byte)

	BZIP	GZIP	Compress
Unicode	3.60	4.04	5.00
ASCII	2.30	2.67	3.66
Ratio Unicode: ASCII	1.56	1.52	1.37

A pleasing result is the performance of LZU-16 on UCS-2 files, compared with the similar LZU-8 on UTF-8 data. It shows that there may be considerable benefit from compressors which acknowledge the 16-bit structure of UCS-2 data.

14.6 UTF-8 COMPRESSION

As it is often necessary to compress information which is already in UTF-8 format, the compressibility of UTF-8 files is given for the three standard compressors, BZIP, GZIP, and Compress. The results, in Table 14.5, are now shown in bits per byte, this being a more appropriate measure for UTF-8 files of unknown content. (Remember that there is no simple relation between bytes and characters in UTF-8.)

Most of the files are reasonably compressible, though not to the extent usually expected for text files. The final file size is about 50% greater than would be expected for an ASCII text file (4.0 bpb vs 2.7 bpb for GZIP, or 3.6 vs 2.3 for BZIP). Although not apparent from Table 14.5, files which are essentially monoalphabetic in a Southeast Asian language may compress very well. Each character has a constant 2-byte prefix, which largely disappears with many compressors. A reasonable compression of say 3 bits per character then converts to a very respectable 1 bit per byte.

14.7 CONCLUSIONS

Unicode files have different compression characteristics from files of more traditional character representations. Accepted "good" compressors such as finite-context PPM do not necessarily work well, although unbounded context statistical compressors are quite satisfactory. Good dictionary or Liv–Zempel compressors also maintain their performance. Tests with a special test compressor indicate that better results may be obtained from compressors which work in the 16-bit units of canonical Unicode. Several possibilities arise for changing compressors to work more efficiently with Unicode files.

1. If a byte compressor detects that it is compressing a UCS-2 file, it could preprocess it into UTF-8 format.
2. Compressors could work with 16-bit symbols, much as LZU-16 has demonstrated. GZIP should certainly be a good candidate for extension, retaining its efficient output coding. The Burrows–Wheeler compressor (BZIP) may be amenable to 16-bit conversion, but at the Cost of a Slower and more difficult sort phase. The compressors may have to recognize the endian nature of files, to suit their coding details or internal data structures. A 16-bit compressor might well convert a UTF-8 (or ASCII) file to UCS-2 for compression.

Either of these conversions should yield compressors which are well tuned to the special requirements of Unicode data.

14.8 REFERENCES

1. The Unicode Standard, Version 2.0, 1996. *The Unicode Consortium*, Addison-Wesley, Reading, MA.
2. Bell, T. C., J. G. Cleary, and I. H. Witten, 1990. *Text Compression*. Prentice Hall, Englewood Cliffs, NJ.
3. Nelson, M., 1991. Arithmetic coding and statistical modelling. *Dr. Dobbs Journal*, Feb 1991. Anonymous FTP from `wuarchive.wustl.edu/systems/msdos/msdos/ddjmag/ddj9102.zip`
4. Burrows M., and D. J. Wheeler, 1994. A Block-sorting Lossless Data Compression Algorithm, SRC Research Report 124, Digital Systems Research Center, Palo Alto, CA. Available at `gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-124.ps.Z`.
5. Cleary, J. G., and I. H. Witten, 1984. Data compression using adaptive coding and partial string matching. *IEEE Transactions and Communications*, COM-32, Vol. 4, pp. 396–402.
6. Cleary, J. G., W. T. Teahan, and I. H. Witten, Unbounded length contexts for PPM. *Data Compression Conference, DCC-95*, pp. 52–61.
7. Fenwick, P. M., 1996. The Burrows–Wheeler transform for block sorting text compression—Principles and improvements. *The Computer Journal*, Vol. 39, No. 9, pp. 731–740.
8. Seward, J., 1996. Private communication. For notes on the released BZIP see http://hpux.cae.wisc.edu/man/Sysadmin/bzip-0.21.