Compression of Unicode files

Peter Fenwick and Simon Brierley
Department of Computer Science, The University of Auckland,
Private Bag 92019, Auckland, New Zealand.
peter-f@cs.auckland.ac.nz

Abstract

The increasing importance of Unicode for text files, for example with Java and in some modern operating systems, implies a possible doubling of data storage space and data transmission time, with a corresponding need for data compression. However it is not clear that data compressors designed for 8-bit byte data are well matched to 16-bit Unicode data. This paper investigates the compression of Unicode files, using a variety of established data compressors on a mix of genuine and artificial Unicode files. It is found that while Ziv-Lempel and unbounded context compressors work well, finite-context compressors are less satisfactory on Unicode. Tests with a simple special compressor intended for 16-bit data show that it may be useful to design compressors specifically for Unicode files.

1. Introduction

An early important example of fixed-length binary coding for text was the well-known 5-bit Murray Teletype code. Early computers quickly moved to various proprietary 6-bit codes and then, in the early 1960's, to 7-bit ASCII and 8-bit EBCDIC, these becoming the standard text representations for the next quarter century.

EBCDIC and ASCII are both designed for representing the 26-letter Roman alphabet, in both upper and lower case but without accent marks, together with a useful selection of punctuation and similar symbols. As such they are satisfactory for English, barely suitable for most other European languages and quite unsuited to the major languages of the Middle East, South Asia and East Asia. Ad hoc coding standards have been developed for many of these languages, often using 16-bit representations but usually incompatible between languages and therefore unsuited to mixed-language text.

Unicode[9] 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. The UTF-8 recoding 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. The conversion between UCS-2 and UTF-8 is illustrated in Appendix 1. Another important coding, UTF-7, uses 7-bit codes for transmission of Unicode over email and similar systems but is not discussed here. Unicode has been used in operating systems such as Plan 9 and much of Windows 95, and in the Java language; it is therefore an important development. For our purposes it is sufficient to note the following aspects, based on the UCS-2 encoding –

1. Unicode is largely based on blocks of 256 codes,
2. The conventional ASCII character set and an extension occupy the first block of 256 symbols, so that the letter ‘A’, represented as 0x41 in ASCII becomes 0x0041 in Unicode (written as U+0041 in Unicode conventions).
3. Other alphabets are allocated to blocks of 256 symbols, but generally avoiding duplication of symbols already defined in other blocks. Thus the Cyrillic alphabet is allocated the U+04xx block, and Thai U+0Exx, as shown in Appendix 3.
4. Ideographic languages such as Chinese and Japanese are allocated suitably large blocks of codes, about 21,000 codes in all.

2. Big-endian vs little-endian.

An immediate problem is one which arises whenever multi-byte entities are to be handled on a byte-addressed computer. Is an entity addressed at its most-significant end (big endian) or at its least significant end (little endian)? If the ASCII text ‘abc’ is written to a file, the bytes in big-endian will be 00 61 00 62 00 63, while the little-endian form will be 61 00 62 00 63 00. More importantly, the bytes 00 01 02 03 04 05 06 07 in the file may be read as either 0001, 0203, 0405, 0607 or as 0100, 0302, 0504 0706, depending on the computer. To overcome this problem a Byte Order Mark U+FEFF may be prepended to the Unicode file. A “big-endian” file will start with the two bytes FE and FF, while a little-endian file will start with FF FE.

To illustrate, the bytes of a file with the letters “code” are, in the two forms –

	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

Figure 1. Comparison of ASCII data “promoted” to big- and little-endian UCS-2

3. Compressors and Unicode

In principle, a Unicode file is a sequence of bytes and can be compressed by any standard lossless compressor. In fact a Unicode file has properties which often degrade the compression from what might be expected. It is easiest to consider some of the main classes of compressor and how they might behave. We assume ASCII data, expanded to 16-bit UCS-2 symbols.

1. **Finite-context statistical compressors**. These compressors, exemplified by traditional PPM compressors [1, 2, 3], predict a symbol from a context of say the previous 4 symbols. That means 4 *bytes*, which is Unicode 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 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.

2. **Unbounded-context statistical compressors**. These compressors, especially PPM*[4] and Burrows-Wheeler block sorting[2, 5], 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 doubling the context order as needed. The operation may be slowed, but compression should remain good.
3. **LZ-77 Compressors**. In these we have a sliding window of recent text (or its equivalent) and emit pointers into the window giving {phrase_displacement, phrase_length} couples. With a byte-oriented compressor, the 16-bit symbols mean that only half the expected number of symbols are covered by both the displacement and the length. Both of these effects reduce the compression.

3.1 The Unicode file test Suite

Unicode files have been obtained via the Web from a directory at Duke University[1] , from a Tamil library at the University of Singapore[2], the Unicode Consortium Home Page[3], and a Unicode resource Page maintained by Ado Nishimura[4].

File Source	File Name	UTF-8 bytes	UCS-2 chars	fraction ASCII	
Corpus	bib	111,261	111,261		
Corpus	paper1	53,161	53,161		
Corpus	progc	39,611	39,611		
Duke	banviet	1,757	1,585	92%	Vietnamese poem
Duke	calblurb	2,761	1,694	68%	description of Unicode browser
Duke	jpndoc	19,653	18,118	96%	Japanese and English
Duke	russmnvr	1,904	1,430	67%	Russian and English
Duke	sample6	2,506	1,443	63%	Korean and English
Duke	unilang	2,037	1,678	86%	Various languages
Tamil	Elangovan	6,676	5,746	79%	Poems, Tamil & English
Ado	kanji-ga	30,669	19,525	54%	Japanese
Ado	kokkyou-no	6,643	5,949	65%	Windows NT & Japanese
Tamil	Purananuru	56,676	25,809	40%	Traditional Tamil poems
	Unicode1.jap	3,354	1,985	45%	Japanese, description of Unicode

Table 1. Files of the test suite.

The selected files mostly satisfy two of the three criteria : a reasonable size, a good amount of non-ASCII text, and good coverage of the Unicode coding space. Others

[1] http://www.lang.duke.edu/unichtm/unichtm.htm
[2] http://mirage.irdu.nus.sg/tamilweb/
[3] http://www.unicode.org/
[4] http://www.ado.sig.or.jp/~ado/unicode/u-link-e.html

have been simulated by expanding text files from the Calgary Corpus, to provide a control against known text files.The test files are shown in Table 1, together with their sizes, the fraction of the text which is ASCII (7-bit) codes, and some commentary.

3.2 Test compressors

These tests used 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. The chosen compressors are –

- **GZIP** The Unix compressor released by the Free Software Foundation, as a good example of an LZ-77 class compressor.
- **BZIP** Julian Seward's implementation of the Burrows-Wheeler block sorting compressor [7], also released by the Free Software Foundation.
- **COMP-2** A PPM compressor described by Mark Nelson[6].
- **Compress** The well-known and traditional Unix compressor, implementing the LZW derivative of an LZ-78 compressor.
- **LZU** This is a specially developed LZ-77 compressor, designed to operate in both 8-bit (ASCII or UTF-8) and 16-bit (Unicode, 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 LZU compressor is described in Appendix 2 to this paper.

3.3 File Handling

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 UCS-2 files (both native and promoted ASCII) are written in both big-endian and little-endian forms, each with an appropriate Byte Order Mark.
- The UTF-8 original files are compressed as bytes.
- The big-endian and little-endian files are compressed as bytes, with no regard to the 16-bit structure.
- The UCS-2 files are split into two component files, one of the more-significant bytes of each character and one of the less-significant bytes. The two components are individually compressed with a byte compressor and the total size of the two compressed files combined. (For ASCII files one component is identical with the original and the other is all-zero.)

We are concerned not only with the relative performance of the compressors on each file – the differences here are well known. We are more interested in how a given compressor behaves on different versions of the same file.

Data compression results are traditionally given in output bits per input byte, or bits

per byte. A recent tendency has been to give performance in bits per character (bpc). With Unicode there is no longer 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 6.

4. Compression tests

In this section we test the compressors and the chosen files. A point which must remembered for all compressors is that many of the files are quite small, with little scope to develop good context information. The compression for these files is usually poorer than for larger files with more available context. ASCII and UTF-8 files are regarded as equivalent.

4.1 Statistical compressors (BZIP unbounded context, PPM finite context)

- The three ASCII files behave as expected from the earlier discussion. BZIP gives very similar results for each of the four versions of most files, while the PPM compressor (COMP-2) gives markedly poorer performance on most UCS-2 files.

	BZIP				PPM order 4			
	8-bit UTF-8	Big Endian	Little Endian	Split	8-bit UTF-8	Big Endian	Little Endian	Split
bib	1.95	1.95	1.95	1.95	2.02	2.72	2.72	2.03
paper1	2.46	2.47	2.47	2.47	2.51	3.04	3.04	2.52
progc	2.50	2.50	2.51	2.51	2.60	3.08	3.08	2.60
banviet	5.10	5.09	5.12	5.43	5.09	5.85	5.91	5.19
calblurb	7.11	7.15	7.27	7.74	7.93	7.86	7.87	7.99
jpndoc	2.88	2.92	2.92	3.00	3.03	3.33	3.34	3.07
russmnvr	4.77	4.92	4.95	5.25	5.40	5.83	5.87	5.33
sample6	6.61	6.63	6.78	7.44	7.46	7.61	7.70	7.76
unilang	5.59	5.63	5.67	5.65	6.16	6.60	6.66	5.84
Elangovan	4.43	4.73	4.94	5.23	4.98	5.44	5.60	5.56
kanji-ga	5.73	6.26	6.68	7.67	5.94	6.84	7.10	8.04
kokkyou-no	4.09	4.35	4.55	4.74	4.33	5.01	5.19	4.92
Purananuru	2.31	2.30	2.29	2.86	2.61	2.46	2.45	3.03
Unicode1.jap	6.75	7.70	8.12	8.64	7.06	8.50	8.72	8.97
Average	4.33	4.36	4.40	4.60	4.69	5.10	5.13	4.70

Table 2. Comparison of Statistical Compressors

- PPM order 4 is always worse on the big-endian and little-endian files because it is working at about order 2. Table 2 repeats the earlier result for the ASCII files, but adding the performance for compression at low orders, confirming the expected change in performance to about order-2 compression.

	8 bit Order 1	8 bit Order 2	Order 4		
			8-bit	Big Endian	Little Endian
bib	3.46	2.65	2.02	2.72	2.72
paper1	3.81	2.93	2.51	3.04	3.04
progc	3.85	2.94	2.60	3.08	3.08

Table 3. Comparison of PPM compressor

- Splitting the file into components breaks up the context information, making the split file always inferior to the UTF-8 file, and often worse than the UCS-2 files. More surprising is the difference between the two UCS-2 files, with the big-endian version being generally better. This probably arises from changes in the context structure as successive bytes are swapped between the two versions and from the non-recognition of the essential 16-bit structure of the files. Even though all three formats (UTF-8, big-end and little-end) have identical logical structure, the byte-wise operation of the compressors breaks up the structure in different ways.

In summary the performance with BZIP (unbounded context) is particularly good, and little different for any of the standard Unicode representations. Finite-context PPM shows a definite decrease performance for UCS-2 files, and the split files are usually inferior.

4.2 Dictionary compressors (Compress – LZ-78/LZW; GZIP – LZ-77)

For both of these compressors the UCS-2 files give rather poorer compression because their phrase lengths and dictionary size are effectively halved.Where the text is dominated by alternating blocks of single alphabets the split files give extremely good performance, but the split files quickly deteriorate as soon as the characters spread across the Unicode space. Both compressors are hampered by the splitting of the UCS-2 codes into bytes during compression.

	Compress				GZIP			
	8-bit UTF-8	Big Endian	Little Endian	Split	8-bit UTF-8	Big Endian	Little Endian	Split
bib	3.35	4.12	4.12	3.39	2.52	3.23	3.23	2.53
paper1	3.77	4.70	4.69	3.83	2.80	3.33	3.33	2.81
progc	3.87	4.84	4.83	3.93	2.68	3.20	3.20	2.70
banviet	6.38	7.64	7.64	6.73	5.36	6.38	6.40	5.69
calblurb	9.74	9.83	9.77	9.38	7.97	8.09	8.07	7.96
jpndoc	4.20	4.97	4.95	4.20	3.28	3.69	3.69	3.29
russmnvr	6.91	8.00	7.93	6.98	5.64	6.23	6.19	5.58
sample6	9.42	9.46	9.49	9.21	7.66	7.82	7.81	7.78
unilang	7.39	8.45	8.37	7.13	6.06	6.70	6.72	5.91
Elangovan	7.86	8.90	9.12	9.07	5.21	6.05	6.11	5.75
kanji-ga	6.17	7.72	7.80	6.70	6.32	7.28	7.39	7.90
kokkyou-no	6.00	7.05	7.19	6.52	4.26	5.05	5.12	4.97
Purananuru	4.09	3.93	3.93	3.97	3.54	3.41	3.37	3.63
Unicode1.jap	8.79	10.28	10.49	10.09	7.29	8.44	8.69	8.96
Average	6.11	6.89	6.87	6.09	4.89	5.41	5.40	4.92

Table 4. Comparison of Dictionary Compressors (Ziv Lempel)

4.3 Simple LZ-77 compressor (LZU-8 8 bit mode; LZU-16 16 bit mode)

This test is included only to compare the performance of two very similar LZ-77 compressors, one operating in conventional 8-bit mode, and the other in a 16-bit mode more suited to UCS-2 Unicode.

	LZU-8				LZU-16			
	8-bit UTF-8	Big Endian	Little Endian	Split	8-bit UTF-8	Big Endian	Little Endian	Split
bib	3.25	4.39	4.39	3.25	4.07	3.41	3.61	4.07
paper1	3.30	4.33	4.33	3.30	4.28	3.46	3.70	4.27
progc	3.16	4.18	4.18	3.16	4.27	3.30	3.64	4.25
banviet	6.52	7.77	7.85	6.94	8.70	6.43	8.67	9.14
calblurb	9.52	9.93	9.86	9.66	12.66	8.26	9.98	11.45
jpndoc	3.75	4.54	4.54	3.83	4.99	3.73	4.21	4.96
russmnvr	6.75	7.81	7.82	6.77	8.96	6.16	8.73	9.03
sample6	9.07	9.58	9.59	9.77	11.81	8.26	9.94	11.66
unilang	7.26	8.04	8.10	6.75	8.85	6.25	8.78	8.56
Elangovan	6.02	7.02	7.19	6.75	7.90	6.32	7.36	8.39
kanji-ga	7.42	8.72	9.07	9.37	9.16	8.10	8.86	11.08
kokkyou-no	4.91	5.89	6.02	5.78	6.35	5.07	6.48	7.47
Purananuru	4.34	4.11	4.15	4.34	4.33	3.45	3.60	5.03
Unicode1.jap	8.77	10.32	10.78	11.36	10.85	9.86	11.11	12.69
Average	5.84	6.73	6.74	5.94	7.62	5.47	8.38	7.49

Table 5. LZU compressor, operating in 8-bit and 16-bit modes.

The only sensible comparison here is between LZU-8 compressing the 8-bit/UTF-8 files and LZU-16 compressing the UCS-2 files. As before, the split files are mostly unsatisfactory, as are the UCS-2 files with LZU-8. One might think that 8-bit LZU-8 and big-endian LZU-16 should give identical results on ASCII files, which they do for minimum phrase lengths of 3 symbols. However LZU-16 gives best overall results with a phrase length of 2 characters, with a slight degradation in ASCII file compression.

The big-endian file with LZU-16 usually gives the best compression, showing the advantage of compressing UCS-2 files with a compressor which recognises their 16-bit structure. LZU-16 is really designed to operate in big-endian mode, with 5- and 8-bit literals for ASCII and other simple alphabets. With little-endian files all literals must be emitted in 16-bit form and the performance suffers accordingly.

5. Comparisons

The split files were a generally unfortunate and forgettable experiment. Except where the data consists of large blocks of single alphabets (not ideographs which spread across many Unicode blocks) their results are mostly inferior to other formats.

Otherwise the results are generally as expected, with the unbounded context statistical compressor performing nearly as well on 16-bit UCS-2 as on UTF-8. Other compressors give reduced performance on the UCS-2 files. Tests with PPM* (unbounded context PPM, and not stated here) paralleled the results with BZIP.

The most serious reduction in compression occurs with the finite-context statistical compressor, although it is still better than some of the dictionary compressors. It is simply not worthwhile using a finite-context compressor on UCS-2; comparable compression is obtained with a good dictionary compressor, and much faster.

A pleasing result is the performance of LZU-16 on UCS-2 files, compared with the similar LZU-8 on UTF-8 data. We can combine the previous results for GZIP and LZU compression

	GZIP				LZU		
	8-bit UTF-8	Big Endian	Little Endian	Ratio avg:UTF8	8-bit UTF-8	16-bit big-end	Ratio 16bit:8bit
bib	2.52	3.23	3.23	128%	3.25	3.41	105%
paper1	2.80	3.33	3.33	119%	3.30	3.46	105%
progc	2.68	3.20	3.20	119%	3.16	3.30	105%
banviet	5.36	6.38	6.40	119%	6.52	6.43	99%
calblurb	7.97	8.09	8.07	101%	9.52	8.26	87%
jpndoc	3.28	3.69	3.69	112%	3.75	3.73	100%
russmnvr	5.64	6.23	6.19	110%	6.75	6.16	91%
sample6	7.66	7.82	7.81	102%	9.07	8.26	91%
unilang	6.06	6.70	6.72	111%	7.26	6.25	86%
Elangovan	5.21	6.05	6.11	117%	6.02	6.32	105%
kanji-ga	6.32	7.28	7.39	116%	7.42	8.10	109%
kokkyou-no	4.26	5.05	5.12	119%	4.91	5.07	103%
Purananuru	3.54	3.41	3.37	96%	4.34	3.45	80%
Unicode1.jap	7.29	8.44	8.69	118%	8.77	9.86	113%
Average	4.89	5.41	5.40	113%	5.84	5.47	98%

Table 6. Results for GZIP and LZU Compression

With GZIP, UCS-2 files compress on average to 113% the size of the UTF-8 files. But UCS-2 files with LZU-16 now compress to 98% the size of the UTF-8 files with LZU-8.

As LZU-8 and LZU-16 differ only in the recognition of 16-bit codings, it seems that having a good dictionary compressor recognise UCS-2 files and emit 16-bit codes may improve its Unicode performance by perhaps 13%. Alternatively, a compressor such as GZIP should recognise UCS-2 files and convert them to UTF-8 for compression.

6. UTF-8 Compression

As it is often necessary to compress information which is already in UTF-8 format, we investigate the compressibility of UTF-8 files for the three standard compressors, BZIP, GZIP and Compress. The results are now shown in bits per byte. (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 of 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). The “Purananuru” file illustrates another feature of Unicode. About 60% of the file is text

from the U+B0xx block, occupying 3 bytes per symbol. If all symbols, 1-byte and 3-byte, compress to 3 bits per symbol (or 1 bit/byte for the 3-byte codings), the resultant compression is 3.0×40%+1.0×60%=1.8 bpc, in line with what is observed. Thus files with only a few simple alphabets may be expected to compress very well.

File Name	UTF-8 bytes	BZIP bytes	GZIP bytes	Compress bytes	BZIP bit/byte	GZIP bit/byte	Compress bit/byte
bib	111,261	27,097	35,063	46,529	1.95	2.52	3.35
paper1	53,161	16,360	18,577	25,081	2.46	2.80	3.77
progc	39,611	12,379	13,275	19,144	2.50	2.68	3.87
banviet	1,757	967	1,017	1,263	4.40	4.63	5.75
calblurb	2,761	1,506	1,688	2,062	4.36	4.89	5.98
jpndoc	19,653	6,525	7,438	9,510	2.66	3.03	3.87
russmnvr	1,904	853	1,008	1,235	3.58	4.24	5.19
sample6	2,506	1,192	1,382	1,699	3.81	4.41	5.42
unilang	2,037	1,173	1,272	1,549	4.61	5.00	6.08
Elangovan	6,676	3,184	3,741	4,312	3.82	4.48	5.17
kanji-ga	30,669	13,984	15,431	19,174	3.65	4.03	5.00
kkkyou-no	6,643	3,043	3,171	4,586	3.66	3.82	5.52
Purananuru	56,676	7,453	11,418	13,194	1.05	1.61	1.86
Unicode1.jap	3,354	1,676	1,808	2,181	4.00	4.31	5.20

Table 7. Compression of UTF-8 Files

7. Conclusions

The results of this paper confirm that 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 LZ 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.

i. If a byte compressor detects that it is compressing a UCS-2 file, it could preprocess it into UTF-8 format and compress that UTF-8 data.

ii. 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 recognise 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.

References

1. Bell, T.C., Cleary, J. G., and Witten,I. H., "*Text Compression*", Prentice Hall, New Jersey, 1990
2. Burrows M. , Wheeler, D.J. (1994) "A Block-sorting Lossless Data Compression Algorithm", SRC Research Report 124, Digital Systems Research Center, Palo Alto. gatekeeper.dec.com/pub/DEC/SRC/research-reports/SRC-124.ps.Z
3. Cleary, J.G. , Teahan, W.T., Witten, I.H, "Unbounded length contexts for PPM", Data Compression Conference, DCC-95, pp 52–61
4. Cleary, J.G. Witten, I.H. (1984) "Data compression using adaptive coding and partial string matching", IEEE Trans Communications, COM-32, vol 4, pp 396–402.
5. Fenwick, P.M. "The Burrows–Wheeler Transform for Block Sorting Text Compression — Principles and Improvements", *The Computer Journal*, Vol 39 No 9 (1996), pp 731–740.
6. M. Nelson. "Arithmetic coding and statistical modelling", *Dr Dobbs Journal*, Feb 1991. Anonymous FTP from `wuarchive.wustl.edu/systems/msdos/msdos/ddjmag/ddj9102.zip`
7. Seward, J. "The BZIP compressor" posted to comp.compression.research newsgroup, (1996).
8. Witten, I., Neal, R., and Cleary, J., "Arithmetic coding for data compression", *Communications of the ACM*, Vol 30 (1987), pp 520-540.
9. —, *The Unicode Standard , Version 2.0*, The Unicode Consortium, Addison-Wesley

Appendix 1. UTF-8 Coding

UTF-8 coding gives a way of representing UCS-2 characters (16-bit) and UCS-4 characters (32 bit) within an 8-bit code stream. ASCII characters are represented unchanged, while others are packed into groups of bytes.

data bits	Input bit pattern	coding into successive bytes		
7	0…0 abc defg	0abc defg		
11	0…0 abc defg hijk	110a bcde	10fg hijk	
16	0…0 abcd efgh ijkl mnop	1110 abcd	10ef ghij	10kl mnop

Figure A1. UCS-2 and UTF-8 coding.

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). Only 2-byte and 3-byte codes are used for UCS-2 characters. (UTF-8 can also handle 32-bit UCS-4 codes and some extended alphabets.)

Appendix 2. The LZU compressor

The compressor was designed as one which could operate in both 8-bit and 16-bit modes, to gain some idea of possible benefit from a compressor specifically designed for Unicode. It has a conventional sliding window buffer holding the previous 8,192 characters. The window is scanned by a fast string matcher, with a hash table leading to lists linking starts of similar phrases. In 8-bit mode single bytes are read into the low-order byte of the 16-bit characters in the buffer, while in 16-bit mode two bytes are read for each character, in big-endian order. All string searching is done on the 16-bit quantities (C `unsigned short`). There is no claim that LZU is an especially good LZ-77 compressor; its sole purpose is operate in both 8- and 16-bit modes.

Output coding consists of control flags followed by either phrase codes or various literal codes —

- `0 d dddd dddd dddd L…L` A **phrase**, with a 0 flag, a 13 bit displacement and then the length using an Elias γ code (or Levenstein code).
- `10 x xxxx` A "**5-bit literal**" code. This code is used where the more-significant bits are the same as for the previous character. It is especially useful when handling monocase ASCII letters or equivalent.
- `110 xxxx xxxx` An "**8-bit literal**" code, similar to the 5-bit code. This is the "last resort" coding for the 8-bit mode; in the 16-bit mode it is used when the character belongs to the previous Unicode block of 256 characters.
- `111 xxxx xxxx xxxx xxxx` A "**16-bit literal**" code, used for general Unicode characters.

As noted earlier, the 16-bit compression suffers if the the sliding window buffer uses big-endian byte alignment but the compressor is used on little-endian files. Compression would also improve in 16-bit mode if the flags coding was reallocated, with a shorter code for 16-bit literals.

In retrospect, the compressor should have been designed to default to little-endian files, instead of the big-endian files which it now prefers. A production design should of course read the byte order mark and adjust automatically to the file characteristics.

Appendix 3 — Unicode Standard 2.0 Code Blocks

The Unicode Standard 2.0 code blocks are presented here. Detailed information on the standard is in Reference 9.

Code Range	Name	Code Range	Name
U+0000-U+007F	C0 Controls and Basic Latin	U+2190-U+21FF	Arrows
U+0080-U+00FF	C1 Controls & Latin-1 Supplement	U+2200-U+22FF	Mathematical Operators
U+0100-U+017F	Latin Extended-A	U+2300-U+23FF	Miscellaneous Technical
U+0180-U+024F	Latin Extended-B	U+2400-U+243F	Control Pictures
U+0250-U+02AF	IPA Extensions	U+2440-U+245F	Optical Character Recognition
U+02B0-U+02FF	Spacing Modifier Letters	U+2460-U+24FF	Enclosed Alphanumerics
U+0300-U+036F	Combining Diacritical Marks	U+2500-U+257F	Box Drawing
U+0370-U+03FF	Greek	U+2580-U+259F	Block Elements
U+0400-U+04FF	Cyrillic	U+25A0-U+25FF	Geometric Shapes
U+0530-U+058F	Armenian	U+2600-U+26FF	Miscellaneous Symbols
U+0590-U+05FF	Hebrew	U+2700-U+27BF	Dingbats
U+0600-U+06FF	Arabic	U+3000-U+303F	CJK Symbols and Punctuation
U+0900-U+097F	Devanagari	U+3040-U+309F	Hiragana
U+0980-U+09FF	Bengali	U+30A0-U+30FF	Katakana
U+0A00-U+0A7F	Gurmukhi	U+3100-U+312F	Bopomofo
U+0A80-U+0AFF	Gujarati	U+3130-U+318F	Hangul Compatibility Jamo
U+0B00-U+0B7F	Oriya	U+3190-U+319F	Kanbun
U+0B80-U+0BFF	Tamil	U+3200-U+32FF	Enclosed CJK Letters and Months
U+0C00-U+0C7F	Telugu	U+3300-U+33FF	CJK Compatibility
U+0C80-U+0CFF	Kannada	U+4E00-U+9FA5	CJK Ideographs
U+0D00-U+0D7F	Malayalam	U+AC00-U+D7A3	Hangul Syllables
U+0E00-U+0E7F	Thai	U+D800-U+DB7F	High Surrogates
U+0E80-U+0EFF	Lao	U+DB80-U+DBFF	High Private Use Surrogates
U+0F00-U+0FBF	Tibetan	U+DC00-U+DFFF	Low Surrogates
U+10A0-U+10FF	Georgian	U+E000-U+F8FF	Private Use Area
U+1100-U+11FF	Hangul Jamo	U+F900-U+FAFF	CJK Compatibility Ideographs
U+1E00-U+1EFF	Latin Extended Additional	U+FB00-U+FB4F	Alphabetic Presentation Forms
U+1F00-U+1FFF	Greek Extended	U+FB50-U+FDFF	Arabic Presentation Forms-A
U+2000-U+206F	General Punctuation	U+FE20-U+FE2F	Combining Half Marks
U+2070-U+209F	Superscripts and Subscripts	U+FE30-U+FE4F	CJK Compatibility Forms
U+20A0-U+20CF	Currency Symbols	U+FE50-U+FE6F	Small Form Variants
U+20D0-U+20FF	Combining Diacritical Marks	U+FE70-U+FEFF	Arabic Presentation Forms-B
U+2100-U+214F	Letterlike Symbols	U+FF00-U+FFEF	Halfwidth and Fullwidth Forms
U+2150-U+218F	Number Forms	U+FFF0-U+FFFF	Specials

Figure A2. Unicode Symbol Allocation