EPiC Series in Computing

Volume 58, 2019, Pages 292–300

Proceedings of 34th International Conference on Computers and Their Applications

Refining the Unrestricted Character Encoding for Japanese

Antoine Bossard[1] and Keiichi Kaneko[2]

[1] Graduate School of Science
Kanagawa University
2946 Tsuchiya, Hiratsuka, Kanagawa 259-1293, Japan
[2] Graduate School of Engineering
Tokyo University of Agriculture and Technology
2-24-16 Nakacho, Koganei, Tokyo 184-8588, Japan

Abstract

We have proposed in a previous work an unrestricted character encoding for Japanese (UCEJ). This encoding features an advanced structure, relying on three dimensions, in order to enhance the code usability, easier character lookup being one application. This is in comparison of, for instance, Unicode. In this paper, we propose several important refinements to the UCEJ encoding: first, the addition of the Latin and kana character sets as ubiquitous in Japanese, and second, the inclusion of character stroke order and stroke types into the code and the corresponding binary representation. We estimate the average and worst-case memory complexity of the proposed encoding, and conduct an experiment to measure the required memory size in practice, each time comparing the proposal to conventional encodings.

1 Introduction

The Japanese writing system relies on different character sets, mainly the Chinese character set and the kana set, the latter consisting of two independent subsets: hiragana and katakana, each made of about fifty characters (raised to about a hundred when including variants). The set of Chinese characters as used in Japanese is comparatively huge, as it includes thousands of characters. This set is the focus of this work.

The representation on computer systems of these numerous characters has always been a hot topic. As soon as the 60s, various character encodings targeting Japanese have been described over the years. Shift-JIS, JIS X 0213 [1], EUC-JP [2] and more recently Unicode [3] are examples of encodings that support Japanese. Yet, these conventional approaches suffer from critical issues: Japanese-specific (or almost) encodings such as Shift-JIS and EUC-JP severely restrict the number of encoded, and thus representable, characters. While this was definitely meaningful at a time when memory space and computer performance were limited, the memory pretext does not stand any more in the XXIst century. And while providing remarkable advantages, the more recent Unicode solution faces other critical issues: by adopting the unifying approach, that is, aiming at including glyphs for all the writing systems known to man, usability challenges arise. For example, it is difficult, and sometimes nearly impossible to locate a particular glyph inside the code. Some of the issues faced by the unifying approach have been summarised in [4].

In this paper, we extend the unrestricted character encoding for Japanese (UCEJ) that has been previously described by proposing several important refinements. First, because ubiquitous in Japanese writings, the kana character sets hiragana and katakana, as well as the Latin alphabet are added into the code. Second, we further discuss the inclusion of the stroke type and the stroke order information into the code. Third, we propose a binary representation for the characters of the UCEJ encoding that takes into account these refinements.

The rest of this paper is organised as follows. Important definitions of the UCEJ encoding are recalled in Section 2. The addition of the Latin alphabet and kana characters is discussed in Section 3. More details and examples regarding the stroke type and order are presented in Section 4. A corresponding binary representation is proposed in Section 5 and empirical evaluation is conducted in Section 6. Finally, this paper is concluded in Section 7.

2 Preliminaries

We have described in a previous work the UCEJ character encoding, whose aim is the unrestricted representation of the Chinese characters as used in Japanese [5]. The main idea is to map each character to a unique three-dimensional coordinate in a way that it is easy to locate any glyph inside the code, unlike with conventional approaches such as Unicode.

This code relies on the following essential properties of Chinese characters (see for instance [6] for additional details):

- Every character has one unique radical, even though there may be some debate regarding the assignment of a radical to a character.
- A character is made of strokes, written in a predefined order.
- A same character can sometimes have different writings; we speak of variants.

The three-dimensional space is then structured as follows:

Dimension 1 Say, the X axis. This dimension is used to distinguish character radicals.

Dimension 2 Say, the Y axis. This dimension is used to distinguish characters based on their stroke numbers.

Dimension 3 Say, the Z axis. This dimension is used to distinguish character variants.

These characteristics and definitions have then been used to derive an (almost perfect) hash function for the Chinese characters used in Japanese. They remain at the core of the research conducted in this paper.

3 Alphabet and Kana Support

The first important addition to the previously proposed character encoding for Japanese UCEJ is the representation of the Latin alphabet as well as completing the support of the Japanese writing system by including the kana characters (i.e., hiragana and katakana character sets).

The Latin alphabet is implemented with the 8-bit ASCII characters, thus resulting in 256 new glyphs added into the UCEJ code. The kana characters consist of 96 hiragana characters and of 96 katakana characters. Hence, in total, 448 new glyphs are added into the encoding.

Regarding the inclusion of these new glyphs in practice, we proceed as follows:

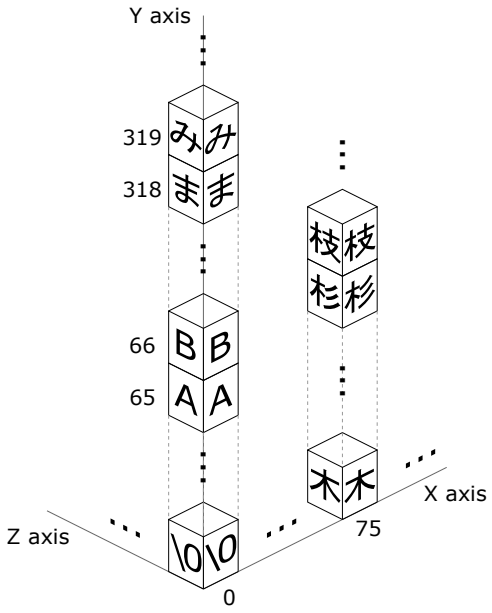

Figure 1: Excerpt of the proposed encoding with ASCII and kana glyphs newly added at coordinate X = 0.

- The ASCII and kana glyphs are located at coordinate X = 0.
- The Y coordinates of the ASCII glyphs match the ASCII code values. For instance, the Y coordinate of 'A' is 65. Hence, the Y coordinates of the ASCII glyphs span the range [0, 255].
- The Y coordinates of the hiragana characters are a mapping of the 96 Unicode values [12352, 12447] to the range [256, 351].
- The Y coordinates of the katakana characters are a mapping of the 96 Unicode values [12448, 12542] to the range [352, 447].
- The Z coordinates of the ASCII and kana characters are set to 0.

Hence, there are in total 448 characters in the UCEJ encoding that have an X coordinate equal to 0; their Y coordinates span the range [0, 447]. And all of these also have a Z coordinate equal to 0. See Figure 1. Some compatibility with ASCII (and UTF-8) is thus retained since the UCEJ coordinate of a glyph of ASCII code q is always of the form $(0, q, 0)$. As recalled in Section 2, the X axis of the UCEJ encoding is used to represent radicals. The case X = 0 is thus a special case: the X coordinate 0 does not specifies the characters of radical ID 0, but instead ASCII and kana glyphs. The 214 radicals thus span the [1, 214] range of the X axis. Finally, for this special "radical" X = 0, the characters are not ordered on the Y axis according to their stroke numbers as it is the case for radicals X $\neq$ 0. Such ordering is effectively not applicable since the characters at X = 0 include Latin letters. Instead, as detailed previously, ASCII code and kana conventional ordering (as for instance in Unicode) is used.

Besides, since this work focuses on Chinese characters as used in Japanese, it can be noted that for the sake of simplicity: 1. we abide by the Unicode classification of the kana characters,

Table 1: Stroke types and the corresponding decimal digit values. Stroke names are given in Japanese.

Value	Stroke	Name
0	丶	*ten*
1	一	*yoko*
2	丨	*tate*
3	丿	*hidari harai*
4	乀	*migi harai*
5	𠂉	*hane*
6	亅 (including 亅, 乚, 乀)	*hane, kagi*
7	𠃌 (including 乛)	*ore*

thus skipping the archaic *yi*, *ye* and *wu* kana characters, 2. the historic kana characters such as hentaigana are not represented, and 3. the half-width versions of the kana characters have been ignored. These skipped characters could be added with minimal effort in the event they are necessary.

4 Stroke Type and Order

As recalled previously, the UCEJ character encoding relies on three coordinates X, Y, Z to identify each character, with the Y coordinate being used to distinguish (and order) characters of a same radical. This particular coordinate is expressed with a real number that has a certain number of decimals in addition to its integral part. The latter is used to express the number of strokes of the character. The former, decimals, were originally introduced to distinguish characters that have the same radical and the same number of strokes. In an attempt at further increasing the accuracy of the code hash function (e.g., for character lookup in the code), it was then hinted that decimals could be used to express additional character information: stroke types and order.

For the sake of clarity, we shall consider hereinafter the eight basic stroke types that have been identified for instance by Coulmas [7]. A more extensive list of character strokes can be found in [6]. A similar representation could be easily derived if considering more than eight stroke types. The basic eight stroke types are given in Table 1 along with the value that will be used as a decimal to represent this stroke type.

Next, given that the highest number of strokes in a Chinese character as used in Japanese is 84 [5], it was derived that 84 decimals would suffice to represent the stroke order (i.e., character writing sequence). And in the exceptional event that two characters that are not variants (otherwise they would have the same X and Y coordinates but differ on the Z coordinate) have the same radical, stroke number, stroke types and stroke order, 6 additional decimals were appended to the right of the 84 decimals. These extra 6 decimals are used to mutually distinguish such exceptional characters. The character pair 弓丨 *hiku*, 弔 *tomurau* is an example of such exceptional characters that are not variants and have the same radical, stroke number, stroke types and stroke order. It was argued in our previous work that 6 extra decimals are sufficient given the estimated upper bound on the maximum number of Chinese characters. It is recalled that the stroke number does not include the radical strokes. The stroke order is given right-to-left: the value of the first stroke type is the rightmost decimal of the 84-decimal block.

Table 2: Details of the Y coordinate: stroke number (radical strokes excluded), stroke type and stroke order representation. The leading zeros of the integral part (i.e., stroke number) are given for clarity.

Stroke number	**.**	**Stroke order**	**Distinctive decimals**
≤ 3 digits	.	84 decimals	6 decimals

Examples:	$d_2^i d_1^i d_0^i$	.	$d_{89}^m \dots d_7^m d_6^m$	$d_5^m \dots d_1^m d_0^m$
技 *waza*	004	.	0...000004721	000000
使 *tsukau*	006	.	0...000431721	000000
雪 *yuki*	003	.	0...000000117	000000
雪 *yuki*	003	.	0...000000117	000000
引 *hiku*	001	.	0...000000002	000000
弔 *tomurau*	001	.	0...000000002	000001

One should note that even though the highest number of strokes of a character in Japanese is as explained 84, we are slightly pessimistic in our approach since radical strokes are excluded from representation with decimals. So, it is safe to assume that we need less than 84 decimals to represent character strokes, radical ones being excluded. Of course, we could subtract from 84 the number of radical strokes of the 84-stroke character *otodo*, but there would be no guarantee that the obtained stroke number remains the highest.

Formally, the Y coordinate of a character is as follows: $d_2^i d_1^i d_0^i . d_{89}^m \dots d_1^m d_0^m$, with d_a^i ($a \in \{0, 1, 2\}$) one digit of the *i*ntegral part, and d_a^m ($0 \leq a < 90$) the a-th decimal from the right of the *m*antissa. The first stroke value is thus the decimal d_6^m, the second one d_7^m and so on. The integral part of the Y coordinate consists of at most 3 digits because of the ASCII and kana characters (i.e., the characters of coordinate $X = 0$): it is recalled that these characters span the range [0, 447] on the Y axis. This representation is clarified in Table 2 together with concrete character examples. It should be noted that the two characters 雪, 雪 *yuki* are variants; since they have the same stroke number, types and order, they have the same Y coordinate (but distinct Z coordinates). Finally, if the integral part of the Y coordinate is equal to 0, it means that the character is a radical. Hence, all the bits of its Y coordinate are set to 0.

5 Binary Representation

The proposed character encoding relies on 3 coordinates for each character. Because the Y coordinate of each character consists in numerous decimals (90, precisely), it is not possible to use the conventional IEEE 754 floating-point representation to represent Y coordinates. So, we describe in this section a fixed-point binary representation for the characters of the proposed UCEJ encoding.

The X coordinate is an integer value in the range $[0, 214]$ since it is used to distinguish character radicals (there are 214) and ASCII-kana glyphs (which consume 1 radical only: their X coordinates are 0). Hence, given that there are 215 possible values to cover, 1 byte suffices to represent the X coordinate of one character.

Regarding the Y coordinate, as described previously, it is a real number. Its integral part is in the range $[0, 447]$ (because of the ASCII-kana glyphs, 84 being the highest stroke number

Table 3: Binary representation of a character's coordinates: 38 bytes are needed (before eventual compression). Examples are given in hexadecimal for conciseness and readability.

X	**Y**	**Z**
1 byte	36 bytes	1 byte

Sample characters:

技 *waza*	0x40	0x0 04 00 ... 00 00 00 00 9D 10 00 00	0x00
使 *tsukau*	0x09	0x0 06 00 ... 00 00 00 23 3D 10 00 00	0x00
雪 *yuki*	0xAD	0x0 03 00 ... 00 00 00 00 04 F0 00 00	0x00
雪 *yuki*	0xAD	0x0 03 00 ... 00 00 00 00 04 F0 00 00	0x01
引 *hiku*	0x39	0x0 01 00 ... 00 00 00 00 00 20 00 00	0x00
弔 *tomurau*	0x39	0x0 01 00 ... 00 00 00 00 00 20 00 01	0x00

for Chinese characters as used in Japanese). The mantissa is made of 90 decimals. Hence, the integral part can be represented on 9 bits. For the mantissa, each decimal of the 84-decimal block is an integer in the range $[0, 7]$, so 3 bits are enough to represent one decimal. Therefore, considering that the 6 distinctive decimals can be represented with 20 bits, a total of $84 \times 3 + 20 = 272$ bits are required to represent the mantissa of the Y coordinate of one character. As a result, the integral part and the mantissa together require $9 + 272 = 281$ bits, and thus $\lceil 281/8 \rceil = 36$ bytes per character.

The Z coordinate is an integer value with 255 as upper bound as discussed previously. Hence, 1 byte suffices to represent the Z coordinate of one character.

From this discussion, in total for one character (i.e., representing the all three coordinates X, Y and Z), $1 + 36 + 1 = 38$ bytes are required to represent one single character according to the proposed UCEJ encoding (i.e., in raw format). This binary representation scheme is summarised in Table 3. In this table, the integral part, stroke order and type part, and the distinctive decimals part are highlighted with different colours. In addition, as the integral part is on 9 bits, since using the hexadecimal notation, the leading 3 bits are ignored.

Obviously, and as shown in Table 3, long sequences of 0 bits will be present for the Y coordinate of most characters since in average, as per our database which is based on the IPA *mojikiban* database [8] version 005.01 (58,861 characters imported, including 1,986 non-Unicode characters), a character has $530025/58861 \approx 9$ strokes (excluding radical strokes and ASCII-kana characters), where 530,025 is the sum of the stroke numbers (excluding radical strokes) of all the 58,861 characters in our database. Thus, by using a compression algorithm such as run-length encoding (RLE) [9], the actual number of bytes taken to store one character can be significantly reduced. Precisely, since in average a character has 9 strokes, $84 - 9 = 75$ decimals of the Y coordinate will be a sequence of $75 \times 3 = 225$ bits set to 0, which corresponds to approximatively 28 bytes set to 0. These 28 bytes can thus be compressed with RLE to, say, only 2 bytes. In average, one character can thus be stored on approximatively $38 - 28 + 2 = 12$ bytes, which is a reduction of about 70% of the original uncompressed character size in the proposed encoding.

For reference and comparison purposes, the memory size taken by one glyph (character) with mainstream encodings is given in Table 4. Because the number of bytes to encode one character with conventional encodings is low, compression on a character basis would have no or even a negative impact; it is thus marked not applicable, and the byte sizes for these

Table 4: Comparison of the memory size taken to store one character: the proposed UCEJ character encoding versus mainstream character encodings.

Encoding	Compression	Size	Remark
UCEJ	none	38 bytes	largely pessimistic
UCEJ	RLE	12 bytes	estimated average
ASCII [10]	n/a	1 byte	Japanese not supported
Shift-JIS [1]	n/a	1 to 2 bytes	Japanese standard
EUC-JP [2]	n/a	1 to 3 bytes	standard for Japanese
JIS X 0213 [1]	n/a	2 bytes	Japanese standard
UTF-8 [3]	n/a	1 to 4 bytes	unifying encoding
UTF-16 [3]	n/a	2 or 4 bytes	unifying encoding

conventional encodings are given when uncompressed.

Thus, in average, the proposed UCEJ encoding requires from 3 to 6 times more memory space than existing encodings. Of course, the UCEJ encoding features many other advantages as recalled and as described in [5] compared with these related works. Because the proposed encoding is targeting textual documents, this memory overhead remains affordable in practice given the low size in average of textual documents, well within range of current storage hardware capacities.

6 Experimentation

As experiment, we consider the following 8-character Japanese sentence (translation given aside):

本日は晴天なり. “Today is sunny.”

which consists of 4 Chinese characters (‘本’ *hon*, ‘日’ *jitsu*, ‘晴’ *sei*, ‘天’ *ten*), 3 hiragana characters (‘は’ *ha*, ‘な’ *na*, ‘り’ *ri*) and 1 ASCII character (‘.’). This trial sentence is successively encoded with the proposed UCEJ encoding (see Table 5) and conventional encodings. The character coordinates are calculated automatically from our implementation of the UCEJ database and code structure. The respective memory sizes are then measured and summarised in Table 6. Since the trial sentence is short, compression is only applied to the UCEJ result. The resulting file size would increase for the other encodings since uncompressed files are made of a few kilobytes only.

An about 32% size reduction was obtained with compression of the UCEJ file. We did not achieve as much as 70% in size reduction as estimated previously because the file size is only a few dozens of kilobytes, about 200kB when compressed, with thus a significant part taken by the file headers. The enhanced structure of the UCEJ encoding is at the cost of approximatively 10 times more memory space than Unicode.

7 Conclusions

In this paper, we have introduced several refinements to the original UCEJ proposal [5]. First, the Latin alphabet, based on the ASCII code, and kana characters have been included in UCEJ in order to increase the usability of this encoding, as both sets (Latin and kana) are ubiquitous

Table 5: Using UCEJ to encode the real-world example string '本日は晴天なり.'

	(X, Y, Z)	**Bytes** (decimal)
本	(75, 10, 0)	`75, 0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0, 0`
日	(72, 1, 0)	`72, 0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0`
は	(0, 303, 0)	`0, 1,47,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0`
晴	(72, 364, 0)	`72, 0,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,122,37,16,0,0, 0`
天	(37, 8, 0)	`37, 0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,16,0,0, 0`
な	(0, 298, 0)	`0, 1,42,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0`
り	(0, 330, 0)	`0, 1,74,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0`
.	(0, 46, 0)	`0, 0,46,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0, 0`

Table 6: Memory sizes (in bytes) taken by the 8-character trial sentence.

Encoding	**Compression**	**Size**
UCEJ	none	$8 \times 38 = 304$
UCEJ	zip-ppmd	208
UCEJ	zip-deflate	217
UCEJ	zip-lzma	226
UCEJ	zip-bzip2	239
UTF-8	n/a	22
UTF-16	n/a	18

in Japanese writings. Then, we have discussed in detail how to take into consideration the stroke order and stroke types of a character into the code. We have discussed the worst-case and average memory size complexity induced by the proposed encoding for character string manipulation. These two proposals have then been empirically evaluated and compared to conventional character encodings. Even though the advanced code structure of UCEJ requires more memory space than that of, say, Unicode, we were able to confirm that compression algorithms can significantly reduce the memory space taken. Future works include the inclusion of the stroke order and stroke type information inside the database on which can be based UCEJ. Then, it would be valuable to further refine the proposed binary representation aiming at lowering the required memory size to store characters, for instance by using a less pessimistic approach regarding the number of decimals of the Y coordinate.

8 Acknowledgements

The authors sincerely thank the reviewers for their insightful comments and suggestions. This research project is partly supported by The Telecommunications Advancement Foundation (Tokyo, Japan).

References

[1] Japanese Industrial Standards Committee (JISC). *7-bit and 8-bit Coded Character Sets for Information Interchange* (７ビット及び８ビットの情報交換用符号化文字集合, in Japanese), 1969.

[2] Ken Lunde. *CJKV Information Processing*. O'Reilly Media, Sebastopol, CA, USA, 2009.

[3] The Unicode Consortium. *The Unicode Standard 5.0*. Addison-Wesley, Boston, MA, USA, 2007. More recent versions accessible online at http://www.unicode.org/versions/latest/.

[4] Antoine Bossard. On Unicode and Chinese characters. In *Science Journal of Kanagawa University*, Vol. 28, No. 2, pp. 235–237, 2017.

[5] Antoine Bossard and Keiichi Kaneko. Proposal of an Unrestricted Character Encoding for Japanese, In *Proceedings of the 13th International Baltic Conference on Databases and Information Systems*, *Communications in Computer and Information Science*, Vol. 838, pp. 189–201, Trakai, Lithuania, July 2018.

[6] Antoine Bossard. *Chinese Characters, Deciphered*. Kanagawa University Press, Yokohama, Kanagawa, Japan, 2018.

[7] Florian Coulmas. *The Writing Systems of the World*. Basil Blackwell, Oxford, England, 1989.

[8] Information-technology Promotion Agency (Japan). *Mojikiban Database* (文字情報基盤文字情報一覧表, in Japanese), 2016. http://mojikiban.ipa.go.jp/. Last accessed February 2018.

[9] A. H. Robinson and Colin Cherry. Results of a prototype television bandwidth compression scheme. In *Proceedings of the IEEE*, Vol. 55, No. 3, pp. 356–364, 1967.

[10] Charles E. Mackenzie. *Coded Character Sets, History and Development*. Addison-Wesley, Boston, MA, USA, 1980.