




# A Chinese character hash function based on strokes for fingerprinting

Antoine Bossard

Graduate School of Science  
Kanagawa University  
Tsuchiya 2946, Hiratsuka, Kanagawa, Japan 259-1293  
 0000-0001-9381-9346

## Abstract

Character representation in computer systems is the main purpose of character encodings, such as Unicode. The representation of Chinese characters in computer systems is a long-standing issue. It is currently still not possible to easily represent, for instance to input, some Chinese characters in computers. In this research, we especially consider the issue of the Chinese characters that are not covered by the conventional encodings. In this paper, in continuation of our previous works on a universal character encoding for such characters, we describe a non-ambiguous hash function for *any* Chinese character. Unlike conventional approaches, this function is solely based on the character strokes, thus eliminating any sort of ambiguity. Given its sparsity and low collision rate, the proposed hash function can then be applied to fingerprinting, which can in turn be applied, for instance, to information retrieval. Simplicity and unambiguity are keys to our proposal. This work is then formally evaluated and compared to previous works so as to show its applicability, contribution and to measure its limits.

## 1 Introduction

Given the large number of glyphs involved, the representation of Chinese characters in computer systems has never been an easy task. Over the years, several solutions have been proposed, starting with characters hard-coded into ROM (still in use today, for instance with some LCD panels [6]) and later with logical encodings, for instance those proposed by the Japanese Industrial Standards Committee (JISC).

Conventional character encodings either follow the unifying approach, such as Unicode [13], or the non-unifying approach, such as encodings that are local to one writing system (e.g. Shift-JIS for the Japanese writing system). Both approaches suffering from limitations, mainly incapacity to include every known Chinese character, we introduced a few years ago a universal character encoding for Japanese (UCEJ) [4]. UCEJ relies on a three-dimensional structure to encode characters: to each character is assigned a coordinate on three dimensions. Simply stated, the first coordinate identifies the character radical, the second coordinate represents the number of strokes of the character, and the third coordinate is used to distinguish character variants.

In addition, the character decomposition operations as presented in [1, 2] are not directly applicable to character strokes given that strokes are not arranged according to easily identifiable patterns like characters.

Now, it needs to be clear that a conventional character encoding, such as Unicode and unlike UCEJ, cannot be used as a hashing function since it only supports a restricted number of characters (i.e. all the Chinese characters are not encoded – we say “covered” – such as many rather infrequent Japanese local characters, the *kokuji* characters [1]). Then, although UCEJ could be used for hashing and fingerprinting, the hash value that corresponds to a character cannot be easily calculated: in fact, the lookup function of UCEJ does not take into account the character variants.

In this paper and in continuation of our previous works on a universal character encoding for such characters, we describe a non-ambiguous hash function that is applicable to *any* Chinese character. Regarding the envisioned applications of this research, universal identification, that is in a unique, unambiguous manner for any Chinese character comes first. This issue is not trivial considering the numerous variants for one same character, variants which are often excluded from the conventional character encodings. So, fingerprinting a Chinese character is very interesting to refer to a character that is not covered by such a conventional encoding. Another fingerprinting method would be to rely on UCEJ [4] or decomposition operations [1], but these are not currently accepted standards, and a method based on decomposition operations can rapidly become ambiguous (e.g. the definition of the support set  $\tilde{R}$  of [1]). Hence, the proposed fingerprinting method: simple and unambiguous, so that even if it is not a standard it can be easily applied.

The rest of this paper is organised as follows: several recalls are made in Section 2, then the proposal is detailed in Section 3 and evaluated in Section 4, and the paper is concluded in Section 5.

## 2 Preliminaries

First, a short recall regarding hashing is made. Well-known to computer scientists, a hash function is used to calculate an index from a datum so that this datum can be used directly to refer to, for instance, the corresponding entry in a table. In the case of table indices, the calculated values are expected to fall within a range so that the table entries tend to be consecutive in memory, and this without any assumption on the sizes of the original data. Besides, it is highly desirable that the calculated indices for distinct data are distinct too, otherwise we say that collisions occur [10].

A fingerprint is another usage of such a function which realises a mapping between data and identifiers, such as indices. Applications of fingerprinting differ from those of conventional hashing functions as just recalled though: rather than calculating consecutive or near consecutive table indices, a fingerprint is typically used to identify with a more or less short value a datum of arbitrary size, which is thus comparable to the scientific applications of human fingerprints. Rabin’s fingerprinting algorithm is a classic example [5].

Second, essential properties of Chinese characters are recalled. A Chinese character has one radical, although there is sometimes a lack of consensus regarding the radical of particular characters (especially because of character simplifications [1]). It has at least one reading, but usually several especially when considering the various languages that involve Chinese characters. It is made of strokes, at least one, and there is a consensus that the highest number of strokes in a Chinese



Figure 1: *otodo*.

character, at least in Japanese, is 84: this is the *otodo* character (a.k.a. *taito*, *daito*), shown in Figure 1 [7]. Strokes are drawn in a certain order, which can fluctuate depending on the language considered [1]. And a character can have several, sometimes numerous variants [11]. Additional details can be found for instance in [12].

### 3 Methodology

The proposed hash function is based solely on character strokes, precisely, on the stroke number, the stroke types and the stroke writing order. It is essential to note that this approach induces no ambiguity at all in the definition of the function. For comparison, in previous researches we relied on the character radical property, and also on character decompositions, for character processing, which is more (the latter) or less (the former) ambiguous. Indeed, given any Chinese character, its number of strokes, the types of its strokes, and the writing order of its strokes is unambiguously defined. Even if the stroke order may differ in some cases from one language to another, for instance between Chinese and Japanese, it is well defined when considering one language. For instance, the Japanese government has formally established the stroke order of the Chinese characters used in Japanese [9].

In order to achieve a low collision rate, we rely on all the three aforementioned stroke properties: number, type and order. The Unicode consortium has defined 36 strokes for Chinese characters (code block 31C0–31EF) [13]; they are reproduced in Table 1, and to each is assigned a (unique) identifier. In this table, the columns are labelled “Id.” and “Str.” for “Identifier” and “Stroke”, respectively.

Table 1: The 36 strokes for Chinese characters (Unicode block 31C0–31EF). They are each assigned a unique identifier.

Id.	Str.	Id.	Str.	Id.	Str.	Id.	Str.	Id.	Str.	Id.	Str.
0	一	6	冫	12	ㄣ	18	ノ	24	冂	30	ㄣ
1	勹	7	フ	13	ㄥ	19	丿	25	丨	31	ㄣ
2	㇇	8	ㄥ	14	ㄣ	20	㇇	26	丿	32	乙
3	㇈	9	ㄣ	15	㇇	21	冂	27	ㄣ	33	ㄣ
4	ㄣ	10	丨	16	一	22	冂	28	ㄣ	34	ノ
5	ㄣ	11	ㄣ	17	丨	23	ㄣ	29	ㄣ	35	〇

Define  $S$  the set of these 36 character strokes, and  $k : S \rightarrow \{0, 1, \dots, 35\}$  the bijection between a stroke and its identifier. Let  $C$  be the set of Chinese characters; it is recalled that its cardinality is unknown. For a character  $c \in C$  of  $n \in \mathbb{N}^*$  strokes  $s_i \in S$  ( $0 \leq i \leq n-1$ ) and of stroke order that induced by the relation  $i < j \Rightarrow s_i$  written before  $s_j$  ( $0 \leq i, j \leq n-1$ ), we define the hash function  $h$  as follows:

$$\begin{aligned}
 h & : C \rightarrow \mathbb{N} \\
 c & \mapsto \sum_{i=0}^{n-1} 2^{6i} k(s_i)
 \end{aligned}$$

In other words, stroke identifiers are each represented with six bits, and the stroke number as well as the stroke order are directly induced by the concatenation of 6-bit sequences. The fingerprint can thus be conveniently represented with the octal notation: each stroke corresponds

to two octal digits. Examples of fingerprint calculations are given in Table 2; in this table, the stroke order is indicated from left to right and fingerprints are given in the octal notation, with the most significant digit on the left.

Table 2: Fingerprint calculation for sample Chinese characters.

Character	Stroke number	Stroke types & stroke order	Fingerprint (octal notation)
大 “large”	3	一, 丿, ㇇	17 22 20
水 “water”	4	丨, ㇇, 丿, ㇇	17 22 07 32
凧 “kite”	5	丿, ㇇, 丨, ㇇, 丨	21 06 21 10 22
迄 “until”	7	丿, 一, 乙, ㇇, ㇇, ㇇, ㇇	17 13 24 24 40 20 22

Once a fingerprint has been obtained, it can then be adjusted for hashing purposes (e.g. hash table), that is, to reduce the sparsity of the obtained fingerprints. This would be at the cost of an increased collision rate though. For example, hashing with folding by summing each stroke value, or division hashing by applying a modulo function to the obtained fingerprints.

## 4 Evaluation

### 4.1 Memory size requirements

First, let us compare the size of fingerprints versus the size of a character coordinate in UCEJ. To this end, we first recall that each character stroke is represented on 6 bits, and that the highest number of strokes in a Chinese character, at least in Japanese, is 84 is a consensus. So, a character of  $n$  strokes requires at most  $6n$  bits (“at most” because the last stroke may not require all the six bits, thus resulting in a few zeros at the MSB, in other words digits that can be discarded). So, an  $n$ -stroke character is expressed on at most  $6n/8 = 0.75n$  bytes. On the other hand, the coordinate of any character in UCEJ takes 10 bytes [4]: the required memory size does not depend on the character. And in the case of the refinement of UCEJ which takes into account stroke types and the stroke order, each character coordinate takes 38 bytes, again no matter the character [3]. This memory size requirement comparison is illustrated in Figure 2; because a conventional encoding such as Shift-JIS or Unicode only supports a fraction of the Chinese characters, it is not included in this comparison as it would be obviously largely unfair. Given that the vast majority of Chinese characters have at most 30 strokes (this is further detailed in Section 4.2 below), the memory size taken by a fingerprint remains reasonable compared to a UCEJ coordinate.

It is however critical to note that a UCEJ coordinate cannot be completely calculated from a character: as recalled in introduction, the UCEJ lookup function calculates from a character its X and Y coordinates only, thus not involving Z. This is a major drawback compared to the fingerprint calculation method proposed herein, and one reason for that lookup function not being a suitable hashing function.

### 4.2 Hash function sparsity

Next, we analyse the projected sparsity of the calculated fingerprints. Directly from above, we have that the fingerprint of a 1-stroke character is in the interval  $[0, 2^6 - 1]$  (since six bits per stroke), that of a 2-stroke character in the interval  $[2^6, 2^{12} - 1]$  (since twelve bits for the two

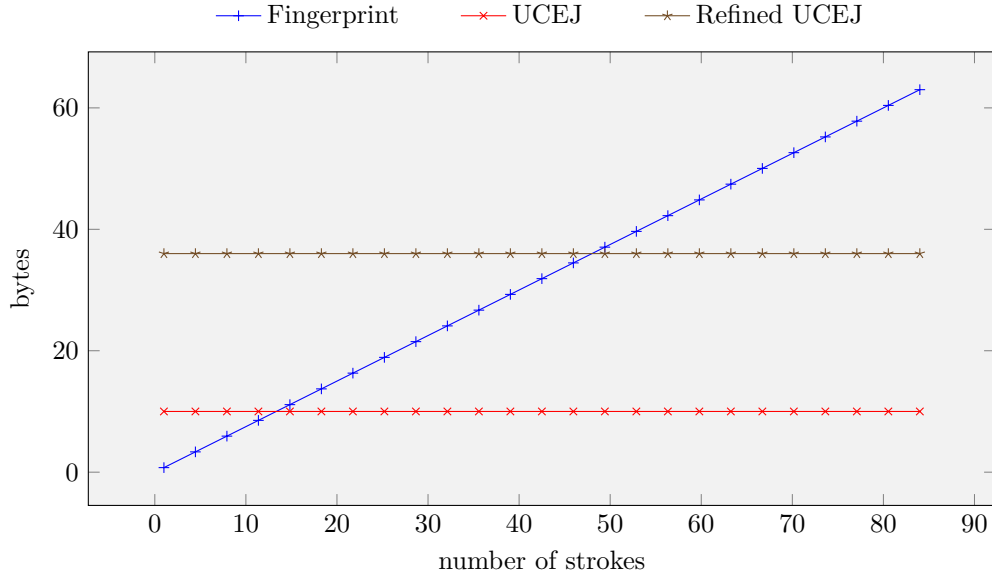


Figure 2: Memory size requirement of a fingerprint versus a UCEJ coordinate.

strokes) and so on. Because a character includes at most 84 strokes as recalled, a fingerprint consists in at most  $84 \times 6 = 504$  bits. Therefore, there are a total of  $2^{504}$  distinct fingerprints, which is of course significantly larger than the number of Chinese characters (even if only an estimation, several tens of thousands, of this character grand total is known). So, the character density in the range of the possible fingerprint values is globally low.

The distribution of the stroke number of the Chinese characters used in Japanese is illustrated in Figure 3. For reference, we have represented in the same plot the maximum number of bits required to represent the fingerprint of a character depending on the stroke number. These data have been extracted from the List of MJ Characters provided by the Japanese Character Information Technology Promotion Council [8]. This database includes in total 58 862 characters. Note that 84 has been considered as the highest stroke number as explained, but since the *otodo* character does not appear in the database, the number of occurrences therein is 0. Hence, although this database is rather exhaustive, the zero number of occurrences as soon as stroke number 65 is yet another indicator of the lacking support of the Chinese characters by computer systems.

It should be noted that the proposed fingerprinting algorithm is not perfect in the sense that it is possible – although rather rare – to find two distinct characters that induce the same fingerprint, for example 彳| *hiku* and 𠂔 *tomurau*, both of fingerprint 21 11 20 25 (octal notation). In other words, the described hashing function is not injective. In an attempt to further reduce the collision rate, additional character properties could be considered. Nonetheless, this would be at the cost of increased ambiguity in the function definition – it is recalled that we have completely eliminated such ambiguity with the approach proposed in this paper. Besides, in this search for a perfect hashing function, it will become clear that the successively established functions, defined at the beginning in a discrete manner, will inevitably evolve towards a continuous (i.e. non-discrete) function, which is problematic considering the hashing applications.

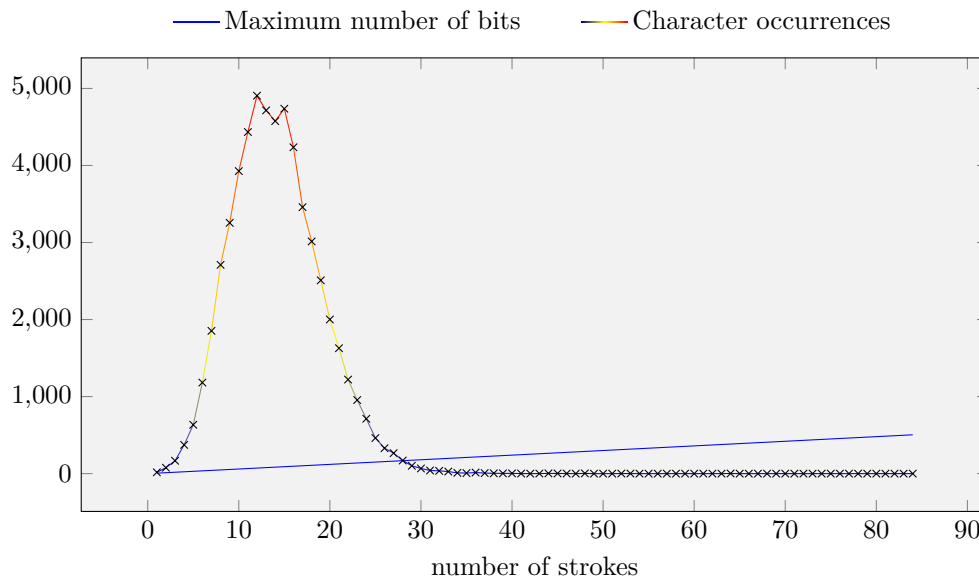


Figure 3: Distribution of the stroke number of the Chinese characters used in Japanese.

Finally, it is interesting to remark the following paradox regarding character density: the characters that have the greatest stroke number are those whose fingerprint occupies the greatest number of bits but which are the least “dense” characters. That is, when considering characters of at most  $n$  strokes, the number of representable such characters is  $2^{6n}$ , but at the same time as  $n$  increases, the number of  $n$ -stroke characters (i.e. character occurrences) decreases. This is clearly visible in Figure 3.

## 5 Conclusions

The representation of Chinese characters in computer systems is a long-standing issue: for instance, it is still not possible to represent some characters, albeit rather infrequent ones. We have been considering this problem for several years and recently defined a universal character encoding for Japanese (UCEJ) to address these issues. UCEJ still has some room for improvement, especially in the automatic calculation of a code point from a character. Directly related to this issue, we have proposed in this paper a non-ambiguous Chinese character hashing function for fingerprinting in order to facilitate the identification and processing in general in a computer system of these characters. The evaluation results show the applicability and contribution of the proposal.

Regarding future works, we are planning to realise a character fingerprint database in order to further experimentally evaluate the proposed hashing function and quantitatively measure its collision rate. It would also be interesting to try to further reduce the collision probability of the calculated fingerprints by refining the hashing function definition. As mentioned earlier, this involves several issues, such as the sparsity of the hash values, the collision rate, the simplicity and discreteness of the established function, issues which absolutely need to be considered simultaneously.

## Acknowledgements

The author is sincerely grateful to the three reviewers for their comments.

## References

- [1] Antoine Bossard. *Chinese Characters, Deciphered*. Kanagawa University Press, Yokohama, Japan, March 2018.
- [2] Antoine Bossard and Keiichi Kaneko. Chinese characters ontology and induced distance metrics. *International Journal of Computers and Their Applications*, 23(4):223–231, 2016.
- [3] Antoine Bossard and Keiichi Kaneko. Refining the unrestricted character encoding for Japanese. In *Proceedings of 34th International Conference on Computers and Their Applications (CATA; 18–20 March, Honolulu, HI, USA)*, volume 58 of *EPiC Series in Computing*, pages 292–300, 2019.
- [4] Antoine Bossard and Keiichi Kaneko. Unrestricted character encoding for Japanese. In *Databases and Information Systems X*, volume 315 of *Frontiers in Artificial Intelligence and Applications*, pages 161–175, January 2019.
- [5] Andrei Z. Broder. Some applications of Rabin’s fingerprinting method. In *Sequences II*, pages 143–152, 1993.
- [6] Hitachi, Tokyo, Japan. *HD44780U (LCD-II) (Dot Matrix Liquid Crystal Display Controller-/Driver)*, 1998. ADE-207-272(Z), ’99.9, Rev. 0.0.
- [7] Takehiro Ito. 辞書になかった最多画数の漢字「幽霊文字」の怪... 「タイト」さんをご存じないですか? *The Yomiuri Shimbun (online)*, November 2020. <https://www.yomiuri.co.jp/life/20201030-0YT8T50053/> (last accessed June 2021). In Japanese.
- [8] Japanese Character Information Technology Promotion Council (一般社団法人文字情報技術促進協議会). List of MJ characters (MJ文字情報一覧表). <https://moji.or.jp/mojikiban/mjlist/> (last accessed June 2021), May 2019. Version 006.01. In Japanese.
- [9] Japanese Ministry of Education, Science, Sports and Culture (文部省). 筆順指導の手びき, March 1958. First edition. In Japanese.
- [10] Donald E. Knuth. *The Art of Computer Science – Volume 3*. Addison-Wesley, Boston, MA, USA, second edition, 1998.
- [11] Kyoo-Kap Lee. Causes of variant forms as a result of structural changes to character components. *Journal of Chinese Writing Systems*, 1(1):29–35, 2017.
- [12] Ken Lunde. *CJKV Information Processing*. O’Reilly Media, Sebastopol, CA, USA, second edition, 2009.
- [13] 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/> (last accessed June 2021).