Computer Processing of Kanji Fonts: A Survey

ROBERT T. MYERS

Abstract. Existing research on computer processing of Chinese character (*kanji*) fonts is surveyed. Literature in the areas of dot-pattern-based processing, vector-type processing, character composition grammars, and automatic placement of character components is reviewed.

0. Introduction

The majority of output devices attached to small Japanese computers, primarily display screens and printers, support *kanji* output. In almost all cases, the output is performed based on pre-stored, dot patterns at some predetermined resolution. This means that usually only one font is available. Storing more fonts requires more storage. These fonts may be manipulated in various ways, such as enlargement or slanting, but in many cases such manipulation gives poor results. Even if adequate storage is available for more fonts, the work involved in digitizing each is tremendous.

0.1. Processing Levels

A variety of research has been done to address these problems. The first level of research occurs at the dot-pattern level. Dot patterns may be used at low resolutions (defined as 24x24 or less), high resolutions (up to 60x60), and ultra high resolutions such as used in computer typesetting (often up to 100x100 or greater). As shown in the table below, the memory requirements and quality attainable change predictably based on the resolution chosen. Typical research here includes ways to store dot patterns more efficiently and involves various compaction algorithms. Other research proposes smoothing algorithms for use in enlarging dot patterns. Finally, some research has occurred on how to change from one font to another while staying purely at the dot pattern level.

Storing fonts as vectors solves many of the problems of dot patterns. In particular, enlargement and other manipulation can be done at higher quality levels. Storage requirements for vector representations are also usually less than those for dot patterns. Research in this areas has centered on algorithms for efficiently vectorizing fonts and on various possible shapes for the vectors. There are two basic approaches: the **skeleton** method stores only the conceptual center line of a stroke, which is then fleshed out at display time; the **outline** method stores the actual character outline which is simply filled in when displayed.

A vectorized approach requires independent vectorization of each stroke in each character in each font. However, clearly Chinese characters are basically composed of various combination of a small number of "components"[1]. By isolating the information about the location of the strokes within each component from that concerning the identity and location of the components that compose a given character, a greater degree of flexibility is attained while simulataneously reducing storage requirements. The terminology we shall use is "component definition" for the data concerning the location of each stroke within the component, and "character composition definition" for the data giving which components are to be placed in which location in the character space to create a complete character. This overall approach is what I call the **composition approach**.

[1]We use the term "component" in the rough sense of a radical or *bushu*.

Under this approach, new fonts can be defined relatively easily, in theory, by redefining the physical shape of each unit, with those new physical shapes then being automatically incorporated into the characters based on the character composition data. The problem with this approach, as we shall see, is that the physical shape of each unit is not in practice independent of the surrounding units within one character or of the that character's space which it occupies. Nor, to the extent that the character composition data places each unit in one fixed location within the overall character space, is enough flexibility gained to do the subtle repositioning might be necessary due to the font changes made.

The latter problem could in principle be solved partially by algorithms which could correctly place, automatically, the components within the character space. In addition, this would reduce the amount of work required to manually specify those locations. This is the **automatic placement** problem. We review a couple of papers on this topic. The other half of the problem requires the ability to perform subtle redesign of each component based on its placement within the overall character. To our knowledge, no research has been done here.

In any case, a subsidiary problem is the representation of the character composition definition information, as well as the identification of the appropriate universe of components. This is what I call the **character composition grammar**" problem. Since such information is required in doing pattern recognition of Chinese characters, we introduce some research from that field.

All of the approaches above have assumed that the component definition information was provided in physical location form. One can also imagine algorithms which could correctly position individual strokes, automatically, within a component. This would eliminate the work required to physically redefine each component for a new font. Perhaps we should be satisifed with being able to correctly place components within characters, but solving this problem, for which no research has been done as far as is known, would provide an even higher level of flexibility and efficiency in defining new fonts.

We present the levels of processing and their characteristics in the following table.

Level	Storage Requirements	Processing Requirement	Technical Difficulty	Flexibility	Quality
Dot Pattern	High	Low	Low	Low	
Low Res	Somewhat Lower				Low
High Res	Medium				Medium
Ultra High Res	Very High				High
Vectors	Medium	Medium	Medium	Medium	Very High
Skeletons	Lower				Slightly lower
Outlines	Higher				Very High
Composition					
Fixed Placement	Low	High	High	High	Medium[2]
Auto Placement	Lowest	Highest	Highest	Highest	Low[3]

Table 1. *Characteristics of Various Levels of Kanji Processing*

0.2. Introduction to Research

Research in these topics has been going on for over twenty years. Many of the older papers are actually more interesting and in many respects the work done more recently does not seem to make any major contribution. We will jump ahead here and introduce the Sakai, Nagao and Terai[39] paper published in 1969, which outlines the issues with admirable prescient clarity. We quote:

> "It is critically important to think about the ways to store *kanji* in computers or draw them as output which are not only the most efficient but also preserve their special characteristics. The same applies equally to the input of *kanji*. The first step in addressing these issues is to think about a grammar system for machines which produce *kanji*....this approach should be well suited to handle characters other than *kanji*, such as *hangul*, as well."

They proceed to develop a comprehensive, clean grammar to describe *kanji* shapes, introduce the concept of "complexity"[4] of a subelement, and in general to lay the framework for a mathematical approach for font design. Their work is introduced in more detail in Section 4 below.

[2]In principle, very high quality should be achievable, but we have marked this Medium in view of the fact that current algorithms for adjustment of component shape based on the surrounding components and so on are not well developed.

[3]Again, we mark this low only due to the lack of success to date in developing good automatic placement algorithms.

[4]*omomi.*

Nomura and Koike[29] start with a good overview of all methods of generating Japanese characters, both analog (including using type) and digital. Under digital methods, they distinguish between the dot matrix approach, the line-dot approach, the "stroke" approach (including both the skeleton and outline methods), the "partial pattern synthesis" method, and the "component unit synthesis method." The "partial pattern synthesis method" apparently refers to the work of Sakai *et al* referred to above, and the authors point out that its problem is that there is no provision to modify the shapes of partial patterns based on their location in a character or relation to other partial patterns in particular characters. The final, "component unit synthesis method", can provide the best results in terms both of quality and storage space (see our table above), but the "synthesis algorithms are complex.". Indeed they are, as we shall see in Sections 4-5 below. The remainder of their paper is devoted to compression algorithms which we review in Section 2.

1. Dot Pattern Level Research

In general, the dot pattern level research can be broken down into attempts to compact dot pattern data, research on smoothing algorithms for expansion of dot pattern data, and other research including experiments with changing character font at the dot pattern level. We will address each in turn.

1.1. Compacting Dot Pattern Data

Work in compaction or compression has been done by Arai, Kato and Yasuda[5,6], Fukizaka[9], Furumaru, Shimamura and Kimura[10], Mori[19,20], Sahashi and Horiguchi[35], Takagi and Tsuda[46], Taniguchi and Mori[49], Tomita, Iwata, Ueno and Onishi[51], and others. However, the problem of compaction has been eased recently with the wide availability of low cost memory. In addition, other approaches, such as vector or shape-based algorithms, naturally accomplish a high degree of compression while providing other benefits as well.

1.1.1. Mori

In general compaction approaches try to improve on standard statistical compression techniques by taking advantage of the special characteristics of *kanji* dot patterns, in particular the prevalence of vertical and horizontal strokes. The work of Mori[19] seems representative. He introduces an enhanced run-length encoding scheme which is based on the fact that *kanji* have white space on the right side. This reduces the area to be encoded to 62% and permits a 16% improvement in compression efficiency over standard run-length methods, from a factor of 1.9 to 2.2. He then proposes a different, transition coding approach, which takes advantage of the presence of vertical strokes in kanji. This gains additional improvements in efficiency. He also develops a model for expected compression factors for different dot pattern sizes and verifies it empirically. Its interesting to note that using this type of coding, memory requirements can be made to increase only linearly with the number of rows or columns in the pattern.

1.1.2. Nomura and Koike

Nomura and Koike, introduced earlier, give an overview of several compression approaches. With pure run length encoding, one may consider the characters either one line at a time (where the values for which runs are detected are either black or white dots), or two lines at a time (in which case, the run values are all the combinations of a black and white dot on the top and bottom line of the two being considered). They derive optimal compression, at 73% of the original requirement, by encoding two rows at a time and encoding runs of a maximum of 4.

They also investigated both simple and compound Huffman coding. The coding was based on dividing the dot pattern up into either 2x2 or 4x1 subgroups, and assigning a code to each of the possible 2^4 values. The compression achieved was 65%. On the other hand, compound Huffman codes, where the code for each value is based on the preceding value, gave a compression ratio of nearly two to one.

1.1.3. Compression Summary

This is by no means a complete review of the compression literature, but there seem to be some contradictions between Mori's numbers and those of Nomura et al. In any case, when coding dot patterns it is probably reasonable to assume that using reasonable techniques compression ratios somewhat greater than two to one can be readily achieved. Of course, these numbers are completely different from those involved in representing characters as vectors or shapes, where, as will be discussed below, compression ratios of ten and twenty to one (over the corresponding naive bit pattern representation) can be achieved.

To my knowledge no one has addressed the question of what the theoretical maximum compression achievable should be. One interesting approach is to note that twelve-by-twelve dot characters are probably the smallest distinguishable by humans. This implies that it should be representable in 144 bits. That corresponds to a compression ratio of about four to one in the case of Nomura's 28x24 bit patterns, or slightly more than twice as good as he achieved with compound Huffman coding.

1.2. Enlargement Algorithms

Okawa, Watanabe, Kazama and Ito[31], with Watanabe, Kazama, Ito and Kameyama[53] propose a simple method which has the problem of introducing mud at intersections. Tanaka, Otsuda and Okada[] have proposed a method for high-quality enlargement and reduction of dot pattern kanji data. They point out that such an algorithm should be able to maintain character balance, should preserve relative line thicknsses, require a minimum of external parameter data, and not result in jaggies (in the case of enlargement). They name their methods RALTH (Reduction to Arbitrary size with a Line Thinning method) and MACS (Magnification to Arbitrary size with a Contour Smoothing method). This method applies various masks to the character to detect Mincho decorations, to find corners prior to line thinning, to smooth the lines of enlarged characters and the like. It appears to give excellent results. However, it seems also to most suited for relatively low-resolution characters (they show results in the 12 dot to 48 dot range).

1.3. Font Conversion Algorithms

Shiono, Sanada and Tezuka[43] have proposed an algorithm for transforming Mincho dot pattern fonts into both square and round Gothic equivalents. However, their approach appears to be quite mechanistic and primitive. Essentially it involves first removing the decorations or *uroko* from the Mincho font; thinning the character to a skeleton, and then placing meat back on the skeletion. Unfortunately the resulting characters appear to be almost unusable. They themselves note that the resulting font should really be called "pseudo-Gothic", and that research remains needed to remove certain "unnatural" results from their method. They also note that the approach seems ill suited to process *kana*.

1.4. Other Dot Pattern Research

1.4.1. Triangular Pin Printers?

Shiono[42] has proposed the use of a dot matrix printer with triangular pins. Each pin in a traditional printer is divided into four triangular sub-sections. The results he shows are most convincing in the case of uroko, which are largely triangular in shape. However slanted lines drawn using the triangular pins look strange and wavy. Most importantly, he seems not to realize that a 48-by-48 rectangular dot printer could probably be built just as easily as a 24-by-24 dot printer with each pin divided into his triangular sections. The former would probably produce much better results.

1.4.2. Digitization Assistance Systems

Production systems using high-resolution *kanji* dot patterns typically transmit an image photographically to a computer where it is digitized and then "cleaned up" by a human being. To the extent the computer can help the human being clean up the digitized character, by being aware of concepts of lines and *uroko*, the faster the cleanup. This is the goal of the system proposed by Nakano and Mori[28]. However, their system provides no help for slanted or curved lines.

2. Vector-based Approaches

Vector-based processing of kanji fonts involves defining the outline or skeletal shape of the character and then filling in the outline or filling out the skeleton. One major issue is the type of shapes to be used in defining the outline or shape.

2.1. Yamazaki and Imura

Yamazaki and Imura[54] analyze the results of approximating curves by straight lines only, by straight lines plus arcs, and finally by straight lines and "arcs considering a farther point". Their work is distinguished by a rigorous consideration of the error involved in various approximation approaches and the time required to both fit the approximating curves and to reproduce the curve. All of their methods yield compression factors over pure bit patterns that approach 95%. The method involving consideration of further points gives much better results all else being equal, but requires more time to fit.

2.2. Sakamoto and Takagi

Sakamoto and Takagi[39] have drawn *kana* and achieved excellent results using a modified cubic spline as the basic form. Their approach involves defining the skeleton of the character (the so-called "ductal" method[5]) and determining stroke width by specifiying a radius at each control point of the cubic spline.

[5] *hone-kaku niku-tsuke hou.*

Specific contributions made by Sakamoto and Takagi are a modified cubic spline where the time or distance parameter ($0 \le t \le 1$) is replaced by a factor d defined as $d=t^{2/3}$. (See also Sakamoto and Takagi[38].) This apparently gives smoother, more natural lines. They also experimented with various approaches to interpolating line thickness between the thicknesses given at the control points[6]. They claim that using the first derivative of the derived cubic spline produces better results than a simple linear interpolation. Personally I can't see any difference. As regards the shape of the "drop" used to trace the skeleton, they point out that although a true ellipse (oriented with its longer axis perpendicular to stroke direction) gives subtly better results, it requires too much computation, and use of an elongated octagon appears to work fine. This observation confirms the work of Knuth in Metafont.

2.3. Sasahara

Sasahara[40] has reported vector-based font work of the skeleton variety. He studies and documents the unsurprising fact that a component such as *ito-hen* appears in subtly different locations depending on the character. We place his work here due to the fact that he apparently does not use the concept of a component, but describes each character in terms of individual strokes. He uses a total of 42 stroke types. He reports that storage requirements for his vector information are about 25% of those for the corresponding dot pattern representation. He implements *uroko* by pre-storing many dot patterns at high resolutions and then tacking them onto the end of his vector-generated strokes. Unfortunately he does not tell us what the mathematical form of the strokes he uses is. His approach can draw tapering lines, but only by discretely changing the line width from one to, say, two. He reports that he was able to vectorize 30 characters per day. My subjective evaluation of his results are the characters are usable if you don't look at them too closely.

2.4. Other Work and Vector Summary

In Zhang, Sanada and Tezuka[58] to be discussed below, the authors use a sine curve and get good results for the three strokes in the character "*san*, three".

An interesting topic of research is methods to digitize the vectors. For instance, it is well known that the digitization algorithms must take special note of the resolution to avoid producing lines of unequal widths. What special issues are raised when digitizing kanji? What about the problem of adjusting digitization algorithms to the peculiarities of one maker's laser printer engine or dot matrix printer's dot characteristics?

3. Composition

Below we introduce work in composing characters from components, but with no particular emphasis on a grammar of character composition or on automatic placement. Naturally, all these systems have some internal scheme for representing character composition which in some sense is a grammar, but we defer explicit treatment of composition grammars to the next section.

[6]Note that MetaFont's `penwidth` operator corresponds to interpolation based on the difference between two cublic splines.

3.1. TEX and MetaFont

MetaFont is Knuth's system for typeface design[17]. It provides a rich toolbox of functionality to define and digitize characters. In terms of its basic capabilties it seems quite suited for defining *kanji* fonts. However some of the font management and coding appraoches it uses may need to be changed or replaced to handle *kanji*.

3.1.2. Japanese Versions of TEX

MetaFont's sister system TEX is for typesetting, and is related to MetaFont in that it uses fonts generated by the latter. TEX also is not exatly suited to *kanji*. For instance, the internal structure to represent a single character allows only 256 fonts and 256 characters per font, although Knuth proposes an extension to handle larger character sets.[7]

Although not directly related to the topic of this survey, we will review the work of Saito[36], who has built a Japanized version of TEX which he dubs jTEX. His version does not involve MetaFont-generated fonts, but rather simply dot patterns. To solve the 256 character-per-font limitation, he simply divides Japanese into a lot of little fonts. But this means each character must be preceded by a TEX command invoking the appropriate font and the character within it. Saito's solution is to build a series of macros where the escape character preceding Japanese text triggers complex processing of the following characters with the result that the proper TEX commands are generated. This approach works, but I can't imagine it is very fast, and Saito confirms this is a major pending problem[8]. Fixing this will require changes to TEX itself. This involves redefining Japanese as one single font, and thus will also require changes to device drivers. Other reamining issues include the preparation of real Japanese fonts to replace the current dot patterns. Currently, when Japanese and English are mixed in jTEX, the English looks much better than the Japanese. Finally, Saito mentions the desirability of preparing standard TEX macro packages for Japanese processing corresponding to those available in English such as LATEX.

3.1.3. Fujiwara's MetaFont Characters

Fujiwara[8] has built some characters using MetaFont. This can be considered a vector-based approach, with the difference that she defines a number of element types and combines them, and of course the element shapes are not given by pure coordinates but rather indirectly in the form of a MetaFont program. The total number of elements used is 28. These are combined into a higher level of predetermined shapes called radicals[9]. Whereas the elements are defined separately for each font (she has worked on Mincho, Kaisho and Gothic), the radical definitions are independent of font.

[7] TEX: The Program, p. 57.
[8] p. 7.
[9] *bushu*.

These characters are somewhat surprising in appearance. Due to the nice MetaFont cubic splines, and the dynamic treatment of *hane* and *mage*, at first glance they appear quite well done. Closer inspection shows major problems in balance. Even the author indicates more work needs to be done before the fonts should actually be used.[10] Two of the specific problems are first, that the system has no way to dynamically adjust the thickness of lines depending on how complicated the character is. Therefore simple characters look too flimsy and complex characters look overblown. There is no way to change the relationship of elements that consitute a radical, something she points out needs to be done, for example, for *nin-ben* depending on its location and size within a character. The major problem with this approach is the colossal amount of work that needs to be done to define the entire universe of characters. Her system appears relatively well structured and parameterized. In particular, she isolates data structures corresponding to our character composition data and component definition data as described in the Introduction. However, it seems to be the case as well that introducing new fonts would require rewriting the character definitions.

As might be expected, *kana* are handled differently. Each *kana* is simply defined in its entirety for each font.

Fujiwara raises the problem of kerning. Especially in the case of *kana*, the distance between characters need to be adjusted. However, whereas MetaFont defines kerning in terms of a two-dimensional table whose axes are left-hand character and right-hand character, clearly this is infeasible with a 5000-character character set. Fujiwara addressed this problem by simply leaving extra space around some of the *kana* which usually need to kern away from their neighbors.

3.1.4. Other MetaFont Work and Summary

Hobby and Guoan[13] have also used MetaFont to build Chinese characters. I have not reviewed their work.

It is questionable whether MetaFont is the right tool for building *kanji* fonts. Since it is in some sense a general programming language, of course it is feasible, but the work to write the program describing each character is likely to be immense. It will never be able to produce fonts in real time. Its design, including the definition of a font itself and the way kerning is handled, is not suited to Japanese. There is no real reason to use MetaFont if the ony benefit gained is that of a cubic spline digitization engine.

3.2. Sanada Team Work

Work done by the Sanada team has reached the state of producing characters which border on usefulness. In actuality the Sanada approach can be characterized as pure component composition without automatic placement. Each character is represented in terms of strokes whose exact location is given in a character database. To draw a particular stroke, a subroutine library was created with instructions for drawing each stroke type at an arbitrary location and in arbitrary size.

[10] p. 67.

3.2.1. Kaisho

In Zhang, Sanada and Tezuka[59], they apply this approach to drawing Kaisho or "square-styled brush-written" characters. To achieve their results, they were forced to define a total of 46 stroke types, each taking between six and twelve parameters. To a large extent, their thinking here was dominated by the fact that they were attempting to create hand-written characters as opposed to printed typefaces. For example, they use algorithms which calculate the actual speed, direction and pressure of the brush on the paper and derive the resulting mark on the paper mathematically. In this connection, they chose to break down a single stroke into a finer gradation of elements they denote "segments", which include the brush start, brush end, brush turning point, and so on.

A point of interest in their work is the use of a sine-based function to describe stroke shape.

3.3.2. Reisho

In Zhang, Sanada and Tezuka[58], the same sort of approach is extended in a straightforward way to the drawing of Reisho.

3.3.3. Automatic Parameter Definition

Due to the large number of parameters required to define each stroke, it would be ideal to devise a way to automatically calculate parameters directly from actual hand-written examples of characters. This is the topic of the paper Uchio, Zhang, Sanada and Tezuka[52]. They succeeded in developing a system where characters were drawn on a tablet and the parameters such as starting and ending point, degree of curvature, and thickness were detected automatically. Characters drawn by their automated system based on these paraemters bear a startling resemblance to the original characters from which the parameters were dervied.

3.3.4. A *Shodo* Expert System?

In Kitagawa, Inoue, Sanada and Tezuka[16], the researchers widen their sights to what they call a Shodo expert system. Here, for the first time they make explicit a representation of the underlying character structure. This collection of data they grandly call a character structure knowledge base. For unknown reasons, they choose to structure it as a pseudo-AI frame-and slot database.

Other work done by this team includes Shiono, Sanada and Tezuka[43], where Mincho dot patterns are translated into Gothic or pseudo-Gothic; this work was described above.

3.4. Sugita *et al*

Sugita, Sakamoto, Shima and Tsukamoto[45] have built a system which generates character shapes from components. Their method of representing the character composition data is called by them "Method of Borrowed Subpatterns"[11] and by me the spaghetti method. When describing any character, they can refer to any other character in the database or any subset of its strokes and bring those strokes, suitably translated or scaled, into the current character. This approach seems to sacrifice structure of the character composition data for a small improvement in storage efficiency, and would probably make it quite difficult, for instance, to modify their database for pre-World War II character shapes or for modern People's Republic of China shapes. It would be interesting to know how fast they can encode, or vectorize, characters. Their results look quite acceptable at 24x24 resolution. Speed of composition on their proprietary hardware was 100 characters per second. In Tomimoto, Ota and Sugita[50] they describe the application of their technology to a word processor which also incorporated character recognition features.

3.5. Hasegawa

Hasegawa[12] described font composition work about 12 years ago. He solves some of the problems encountered by others by introducing a large number of components, namely 280 (plus another 490 for doing English letters and *kana*, for a total of 770). He also introduces "compound basic patterns", which are combinations of basic patterns which may be used in turn in building characters. Of these there are 700 for *kanji* alone and another 190 for other characters. Each character can be described by an average of 2.5 basic patterns and compound basic patterns. Some of the basic patterns are "pre-shrunk" into the position they will occupy in the final character; others are shrunk or moved around when placed into the character. His data occupies only 1/8 of the corresponding naive dot pattern representation for 32x32 dot characters (which is what he produced). His composition speed, at 3000 characters per second, is quite good due to the use of some special hardware. It would be nice to see examples of his characters, if he actually built any, and to know how long it took him to code each character.

3.6. Summary

It is not too surprising that by specifying enough information, quite good results can be obtained from the component composition method. Within the framework of specifying each component's location exactly, which is what defines the component composition method at this level, it would be quite interesting to do a more theoretical study on the proper number of components, compound components if they are to be used, and stroke types. In the work introduced above, the number of components ranged from under 100 to over 1000, and the number of strokes from less than 10 to more than 50. Standard character dictionaries use about 300 radicals, of which some are certainly best treated as combinations of others, and to which addditional components should certainly be added for most efficient representation.

4. Character Composition Grammars

A variety of research has been done in how to represent the underlying shape of *kanji* and use that information in pattern recognition.

[11] *bubun pattern shakuyou hou.*

A major distinction in the types of grammars can be made based on how much information they contain. In general, the less information embedded in the grammar the more general and flexible its application. At one extreme, we have grammars which specify exact coordinates and other parameters for each stroke, and amount to glorified vector representations. On the other end of the spectrum are grammars which describe the charac - ters shape in complete geometric generality (and thus presumably would be used in conjunction with an automatic placement algorithm). All grammars share the characteristic that they take note of and use common components within characters. Another distinction between grammars is the number, and level of complexity, of the "atomic" elements they use.

4.1. Rankin

Rankin[33] and Rankin,Sillars and Hsu[34] first addressed the problems of defining and representing the shapes of Chinese characters over a decade ago. I have not reviewed their work.

4.2. Sakai

We have already introduced the work of Sakai et al in the Introduction. They develop a simple algorithm to denote the ways elements can be combined. It is worth listing their basic operators: left/right, left/right touching, top/bottom, top/bottom touching, piercing, enclosing, crowning, embedded to top and right (*nyuu*), embedded to bottom and left (*tasuki*), and embedded to right and bottom (*tare*).

They then discuss criteria for choosing the components to be combined by these operators and end up choosing 250, which are related but not identical with the classical *bushu* of Sino-Japanese dictionaries.

We will discuss the work of Sakai related to automatic placement in that section below.

4.3. Nagahashi and Agui

Work coming from this team includes Agui[3], Agui, Nakajima and Nagahashi[4], Nagahashi[22], Nagahashi and Agui[23,24,25,26], and Nagahashi, Nakatuyama and Nishizuka[27]. I am at a loss to interprest this body of research. They note, for instance that components of a character are included within a box. The whole area occupied by the character can be divided into a number of "physical areas", within which "logical areas" containing the component can be placed by one of several predefined rules. A character's structure can be described by a tree showing the interrelationships of its components. However, I fail to understand why this tree must be defined by a complex generative grammar with an arbitrary number of subscripted production rules. A distinction of questionable usefulness is made between "main blocks", "cross blocks" and "neighbor blocks".

They have tried generating some characters. However, these attempts are flawed, first, by the lack of any attempt to automatically position elements. For instance in one of their papers, they make the unremarkable ovservatio nthat with different direct specifications of how the blocks inside the character space are to be allocated, the resulting characters come out looking different. As far as I can tell, they use no curves, also contributing to the ugliness of their characters.

5. Automatic Component Placement

5.1. Sakai

We have already referred to this work in the introduction as well as in the section above on composition grammars. Eighteen years ago, these gentlemen were already developing mathematical approaches, using their NEAC 2200, to automatically determine the proper placement of character components. They first define a default space allocation for each operator. For instance, a crown always takes the same amount of space on the top. But for the vertical and horizontal combination operators, they asssign a "weight" to each of the two components, based on whether it is in turn an operator itself, or if an element, based on its belonging to one of three classes they define. The partitioning then takes place depending on the ratio of the weights of the two components being combined.

The problem remains that sometimes components cannot be gotten to touch each other properly even when using the "touching" operators, due to the definition of how the element sits within its box. To solve this, the researchers introduce a special trimming operation.

They summarize by noting that their approach cannot handle about ten characters well, including that for *tame*. The characters are "slightly unnatural". No way exists to specify the dot present in the character for *inu*, for example. Finally, they note that their methodology could equally well be applied to *hangul* (probably with fewer operators).

5.2. Shigegaki *et al*

Shigegaki, Kurose and Yamada[41] present an admirable attempt at solving the problem of automated location of strokes and components. They suggest an iterative approach where the strokes and components are laid out essentially at random and then their position is modified until certain conditions are satisfied. This method appeals to the intuition that the relationship between components is complex and may not be able to be represented deterministically. The conditions driving their iteration are essentially the ratio between the "weight" of a component and the area it occupies.

They also introduce the idea of separate horizontal and vertical weights of a component. The former are used to allocated space between side-by-side characters, and so on. However, the characters produced by their tests clearly need more work.

In a related paper Kurose, Shigegaki and Yamada[18] the authors combine the above system with a general framework for representing character shapes as well as actually generating various sorts of brush strokes to create the characters.

6. Summary

6.1. Terminology

The first impression from reviewing the research is that a more standardized terminology is needed. For instance, the term subpattern is used both to describe a 2x2 pixel area for Huffman coding, as well as for a radical in a character. The terms kanji unit, subpattern, component, radical, segment, area and so on should be clarified. We prefer our term "component"[12] to refer to a part of a character approximating the level of complexity of the tradition radical. The term "segment"[13] introduced by Sanada *et al* to refer to one element of a stroke (such as the brush hitting the paper, moving along, turning, or leaving the paper), also seems reasonable.

6.2. Dot Pattern Research

The need for additional research in this area is waning. Cheap memory is reducing the problems of memory requirements. Complicated and clumsy attempts to process characters already reduced to a dot pattern representation are best replaced by processing at a higher level of abstraction, namely vectors or shapes. To the extent processing power slowed down the popularization of vector and shape approaches in the past, this is clearly now changing. It will not be unlikely for computers in the future to have a specialized, dedicated font or character slave microprocessor.

6.3. Vectors

Vectors are the wave of the short-term future. The technology is now well understood, as is used in PostScript[1,2], for example. The combination of low memory costs and the low memory requirements for vector representations of fonts means that several font styles and sizes can be included into most output devices. With today's microprocessors, CPU cycle time requirements are no longer really a bottleneck, especially when algorithms for caching calculated characters and the like are considered.

Due to the unsolved problems in the composition approach, even when a large amount of data is pre-specified, commercial vector-based systems are likely to vectorize each character independently. The advantages of composition lie in its potential for reduced storage, which is less and less important, and in its flexibility, which is not an immediate demand.

The best results in terms of output aesthetics and memory requirements are obtained by cubic splines, possibly modified. However, the classic proponent of their use, MetaFont, is too clumsy to use for designing *kanji* fonts. Although commercial systems are available for vectorization, vectorizing an entire *kanji* font still requires too much work. The next stage is likely to be support systems to ease the vectorization task, including the ability to make local vector modifications and minor stylistic changes in characters easily. The problems involved in doing professional-level character composition, much less automatic component placement, mean that these approaches are not likely to be useful commercially in the near future.

[12] probably in Japanese *kousei bubun*.

[13] *segumento*.

6.4. Composition

The shape, or composition, based approach is likely to be dominant in the medium to long-term future. The reason for this lies in the ultimate flexibility it offers to create, automatically, a wide variety of fonts. Since the intermediate output of the shape approach is most likely vectors, it also offers all the advantages of that methodology.

6.5. Composition Grammars

The first problem here is that of the grammar to represent the character shapes. Nowhere in the research has a definitive grammar been proposed, nor have the criteria for what consitutes a good grammar even been set forth. The problem can be defined as consisting of the definition of the elements of the grammar and the operators that combine them. The operators consist tautologically of composition in the vertical, horizontal, and inside/outside dimensions, and possibly a few others, including those to place dots such as occur in the upper right of the character "*inu*, dog". An interesting research topic is the universe of components as well as the possibility of defining distinct levels of components and interrelationships between them. The proper choice will optimize space requirements, provide the modularity needed to modify the representations and use them in different applications, and most importantly, preserve the information needed by a variety of applications to produce the highest-quality characters. For instance, use of the representation data in pattern recognition may require fairly high level components at some level in the description, since humans are likely to abbreviate such high level components as one unit. Although if the proper grammar is chosen, in theory characters will need to be encoded only once, it is still a consideration that the grammar should be easily codable.

6.6. Automatic Placement etc.

The real problem is that of drawing the characters based on the abstract shape representation. This includes calculation of where within the character space each component should be placed, and encompasses the problem, as yet unaddressed in the literature, of inter-component kerning. It also includes the issue of modifications to components based on their neighbors. Nor has research been done on the nature of parameters which might control such automatic placement or other aspects of character generation, beyond the simplest physical parameters such as line thickness. The simple algorithms that have been proposed so far may represent less than one percent of the complexity of those needed to create publication-quality fonts. This is the major research topic for the future.

Bibliography

Journal Abbreviations

IECEJ	Inst. Electronics Comm. Engrs. Japan
IP	Information Processing
IPSJ	Information Processing Society Japan
TIECEJ	Trans. Inst. Electronics Comm. Engrs. Japan
PNCIECEJ	Proc. National Convention IECEJ
PNCIPSJ	Proc. National Convention IPSJ

*Titles of papers marked with an asterisk have been translated from Japanese by the author. Other English titles are those given by the original author.

[1] Adobe Systems Incorporated, *PostScript Language Reference Manual*, 1985, Addison-Wesley.

[2] Adobe Systems Incorporated, *PostScript Language Tutorial and Cookbook*, 1985, Addison-Wesley.

[3] Agui, T., "Representation of Hand-written Chinese Characters by Relationship of Position of Sub-patterns"*, *TIECEJ*, 60-D, No. 12, Dec., 1976, pp. 1109-1116.

[4] Agui, T., Nakajima, M. and Nagahashi, H., "A Description Method of Handprinted Chinese Characters by Relative Locations among Partial Patterns" (*Bubun pattern no ichi kankei wo riyou shita tegaki kanji no hyougen hou*), *TIECEJ*, J60-D, No. 12, December 1977, No. 1109-1116.

[5] Arai, Kato and Yasuda, "An Improvement in Kanji Data Compression by the Stroke Method" (*Stroke hou ni yoru kanji data asshuku no ichi kairyo*), *PNCIECEJ*, 1975, 973.

[6] Arai, Yasuda, and Kato, "Several Methods for Pixel Kanji Data Compression" (*Gasokei kanji data asshuku no ni, san no houhou*), *Trans. Inst. Electronics Comm. Engrs. Japan*, Vol. 57-D, No. 6, July 1981.

[7] Casey R. and Nagy G., "Recognition of Printed Chinese Characters", *IEEE Trans. EC*, Feb. 1966.

[8] Fujiwara, K., "Development of Japanese Fonts with METAFONT" (*Metafont ni yoru nihongo font no kaihatsu*), paper published privately.

[9] Fukizaka, "A Method for Compression of Kanji Pattern Data"*, *PNCIECEJ*, 1978, 5-139.

[10] Furumaru, Shimamura and Kimura, "A Method for Data Compression of Pixel Kanji Patterns" (*Gasokei kanji pattern ni okeru data asshuku no ichi houhou*), *PNCIECEJ*, 1976, 5-51.

[11] Hasegawa, "A System for Generating Kanji Patterns"* (*Kanji pattern hassei system*), *Image Electronics*, Vol. 3, No. 4, 1974.

[12] Hasegawa, J., "KANJI Input and Output System by Method of Unit Pattern Construction" (*Pattern Gousei ni yoru kanji nyuushutsuryoku shori*(, *IP* , Vol. 16, No. 9, Sept. 1975, pp. 808-817.

[13] Hobby, J. and Guoan, G., "A Chinese Metafont", *TUGboat*, Vol. 5, No. 2, 1984.

[14] Izumi, Y., Harashima, H., and Miyakawa H., "Absorption of Variations in Handprinted Chinese Character Using Hierarchical Dictionary Modified by Strokes" (*Stroke ni motozuku jisho no henkei wo mochiita tegaki moji no hendou kyuushuu*), *TIECEJ*, Vol. J69-D, No. 2, Feb. 1986.

[15] Kida, Hozaka, Tominaga, "Method for Generating Kanji Diagrams" (*Kanji zukei no hassei houshiki*,) *PNCIECEJ*, 1978, 5-139.

[16] Kitagawa, K., Inoue, T., Sanada, H. and Tezuka, Y., "A Suggestion on Systems to Generate Computer-based Brush-type Chinese Characters" (*Keisanki saiyou mohitsu jitai kanji sekkei system ni tsuite no ichi kosatsu*), *PRL*, 85-61.

[17] Knuth, D. E., *Computers and Typesetting*, Volumes A to E, Addison-Wesley, 1983—1987.

[18] Kurose, H., Shigegaki, and Yamada, "A System for Creation of Arbitrary Fonts from Basic Kanji Fonts" (*Kanji kihon shotai kara kakushu shotai no sakusei system*), *PNCIPSJ (1982 1H)*, pp. 1025-1026.

[19] Mori, K., "Considerations on Data Compression for Dotted Kanji Patterns" (*Gasokei kanji pattern no data asshuku ni kansuru ni, san no koan*), *TIECEJ* J64-D, No. 7, July 1981, 617-624.

[20] Mori, K., "Size Transformation for the Dot Represented Kanji Character Pattern" (*Dot Kanji Pattern Matrix no Jisuu Henkan Hou*), *TIECEJ*, J60-D, No. 10, Oct. 1977, pp. 801-808.

[21] Nakagawa, M. and Takahashi, N., "Structuring a Character Dictionary by Sub-Patterns" (*Subpattern no dounyuu ni yoru moji jisho no kouzouka*), *JDP*, Vol. 5, No. 4, March 5, 1986.

[22] Nagahashi, H., "On the Coding of Patterns for Hand-written Kanji" (*Tegaki kanji pattern no fugouka ni tsuite*), *TIECEJ*, J61-D, No. 11, Nov., 1978, pp. 803-810.

[23] Nagahashi, H. and Agui, T., "A Coding Method of Handprinted Chinese Characters" (*Tegaki kanji pattern no fugouka ni tsuite*), *TIECEJ*,, J61-D, No. 10, October, 1978, 803-810.

[24] Nagahashi, H. and Agui, T., "A Pattern Generation Method of Chinese Characters" (*Kanji pattern no seisei hou ni tsuite*), *PRL*, 80-15..

[25] Nagahashi, H. and Agui, T., Representation and Coding of Kanji Patterns* (*Kanji pattern no hyougen to fugouka ni tsuite*), *PRL*, 80-14, pp. 49-56.

[26] Nagahashi, H.and Agui, T., "Relation between Description and Coding of Chinese Characters", Kanji pattern no hyougen to fugouka ni tsuite, *TIECEJ*,, J64, No. 4, April 1981, pp. 316-323.

[27] Nagahashi, H., Nakatuyama M. and Nishizuka, N., "A Study of Pattern Generation of Chinese Charactors and Data Structure of the Patterns" (*Shojun ni yoru kanji pattern no seisei to sono data kouzou ni tsuite*), *PRL*, 82-13, 31-38.

[28] Nakano, H. and Mori K., "Kanji Pattern Processings Based on the Separation of Vertical or Horizontal Lines" (*Chokusen bunri hou ni yoru kanji pattern no shori*), *TIECEJ*,, Vol. J62-D, No. 12, Dec. 1979, pp. 796-803.

[29] Nomura, S. and Koike, H., "Japanese Character Generating Method" (*Nihongo moji hassei houshiki*), *IP*, Vol. 21, No. 11, Nov. 1980, pp. 1136-1143.

[30] Ogawa, H. and Tezuka, Y., "Hierarchical Representation and Recognition of Chinese Characters" (*Kanji no kaisou hyougen to sono ninshiki*), *TIECEJ*, 57-D, 12, Dec. 1974, pp. 700-707.

[31] Okawa, Watanabe, Kazama and Ito, "Study of Kanji Pattern Enlargement Methods" (*Kanji pattern kakudai houshiki no kentou*), *National Convention of Institue of Image Processing*, Discussion Paper, May 2, 1976.

[32] Okishi, Uchibori and Watanabe,"Kanji Font Generation on a Standard Terminal" (*Han'you tanmatsu sochi ni yoru kanji font no sakusei*), *IE*, 79-4.

[33] Rankin, B. K., *A Linguistic Study of the Formation of Chinese Characters*, Ph.D. Thesis, 1965.

[34] Rankin, B. K., Sillars, W. A., Hsu, R. W., *On the Pictorial Structure of Chinese Characters* , NBS Technical Note, Jan. 1965.

[35] Sahashi, Horiguchi, "A Proprosal for Compression of Kanji Data"* (*Kanji data no asshuku ni kansuru ichi koan*), *PNCIEJCE*, 1975, 5-143.

[36] Saito, Y., "Japanese TeX" (*Nihongo TEX*), *Japanese Document Processing*, 10-3, Jan. 21, 1987.

[37]Sakai Toshiyuki, Nagao Makoto, Terai Hidekazu, "A Description of Chinese Characters Using Sub-Patterns" (*Bubun pattern ni yoru kanji no gousei*), *IP*, Vol. 10,No. 5, Sept. 1969, pp. 285-293.

[38] Sakamoto, M. and Takagi, M., "A cubic spline with a curve deviation nearly proportional to its chord length" (*Gen ni taisuru ko no fukurami no hi ga hobo ittei to naru 3jigen spline kyokusen*), *PRL*, 84-33.

[39] Sakamoto, M. and Takagi, M., "Generation of High-Quality Mincho-face Hiragana and Katakana by Computer" (*Kouhinshitsu minchoutai hiragana, katakana no keisanki ni yoru seisei*), *TIECEJ*,, J-68D, 4, April, 1985.

[40] Sasahara, "On the Feasibility of a Printer Supporting Many Fonts" (*Houfu na moji shu wo motsu printer no jitsugensei ni tsuite*), *Japanese Input Methods* 19-4.

[41] Shigegaki, M., Kurose, H. and Yamada, H., "On Balancing Algorithgms for Kanji Character Shape Skeletons" (*Kanji shotai no honegumi balance ka algorithm ni tsuite*), *PPNCIPSJ (1H 1982)*, pp. 1023-1024.

[42] Shiono, M., "High-quality Kanji Dot Printer For Japanese Language Word Processor using Triangular Dots" (*Sankaku dot wo mochiita waapuro you kouhinshitsu kanji printer*), *JDP*, 9-2, Nov. 12, 1986.

[43] Shiono, M., Sanada, H., and Tezuka, Y., "Transformation of Minchou-type KANJI pattern into Gothic-type" (*Insatsu kanji pattern no minchoutai kara gothic tai e nojitai henkan hou*), *IT* ,48-2..

[44] Suenaga, Optimization of Kanji Display through the Stroke Method* (*Stroke hou ni yoru kanji hyouji no saitekika*), *PNCIECEJ*, 1975, 972.

[45] Sugita, T., Sakamoto, H., Shima, K., and Tsukamoto, K., "Multi-Font Kanji Synthesizer" (*Multi-font kanji goseiki*).

[46] Takagi, M. and Tsuda, K., "Study of Kanji Pattern Data Compression using Two-dimensional Forecasts" (*2-jigen yosoku ni yoru kanji pattern no data asshuku no kentou*), *Proc. National Conference*, IECE, 1976, 5-52.

[47] Takahashi, N., "Expectations for Information Processing of Japanese Language"* (*Nihongo jouhou shori e no kitai*), *IP* , Vol. 20, No. 1, 1979, pp. 44-49.

[48] Takamura, Kubo andKitamura, "Kanji Pattern Generation Based on Third-degree Curve Interpolation"* (*3-ji kyokusen hokan houshiki ni yoru kanji pattern no hassei*), *PNCIECEJ*, 1977, 5-52.

[49] Taniguchi and Mori, K., "A Method for Compression of Pixel Kanji Patterns" (*Gasokei kanji pattern asshuku no ichi houhou*), *TIECEJ*, Vol. J57-D, No. 6.

[50] Tomimoto, T., Ohta H. and Sugita T., "Structure and Development Process of the Hand-Writing Word Processor"* (*Tegaki waapuro no kouzou to kaihatsu katei*), *PRU* , 86-77.

[51] Tomita, Iwata, Ueno, Onishi, "A Method for Compression of Kanji Patterns" (*Kanji pattern asshuku no ichi houhou*), *PNCIECEJ*, 1975, 974.

[52] Uchio, F., Zhang, X., Sanada, H. and Tezuka, Y., "A Semi-automatic System for Determining Feature Parameters for Generating Square-styled Brush Characters" (*Kanji kaisho mohitsu tai seisei parameter no han jidou kettei system*), *PRL*, 84-35, pp. 37-44.

[53] Watanabe, H., Kazama, N., Ito, H. and Kameyama, M., "Study of Kanji Dot Pattern Enlargement Methods" (*Moji pattern kakudai houshiki ni kansuru ichi kentou*), *PNCIECEJ*, 1977, 5-52.

[54] Yamasaki, I.and Imura, H., "Curve-Fitting for Character Outlines with Circular Arcs and Straight Lines" (*Moji rinkakusen no enko to chokusen to ni yoru kinji*), *IP*, Vol. 26, No. 4, pp. 726-732.

[55] Yamashita, Higuchi, Yamada and Haneshita, "Categorization of Hand-written Chinese Characters by Method of Combination of Hierarchical Strokes" (*Kouzouka senyou seigouhou ni yoru tegaki kanji no daibunrui - shuuhen bunpu no riyou*), *PRL*, 82-12, June, 1982.

[56] Yasaku, K., *The Mincho Typeface*, 1976, Heibonsha.

[57] Yasuda Yoshiaki, "Present State and Future Prospects for Information Processing of Chinese Characters"* (*Kanji jouhou shori no genjoy to tenbou (I, II)*), *TTIECEJ*, Vol. J58-D, Nos. 7-8, July-Aug., 1978.

[58] Zhang, X., Sanada, H. and Tezuka, Y., "Forming REISHO of Chinese Character with a Computer" (*Keisanki ni yoru kanrei no seisei*), *PRL*, 84-34, pp. 29-36.

[59] Zhang, X., Sanada, H. and Tezuka, Y., "Generation of "Kaisho" Characters by Computer" (*Kanji kaisho mohitsu jitai no keisanki ni yori seisei*), *TIECEJ*, Vol. J67-D, No. 5, May, 1984.