Behav Res (2015) 47:826–836
DOI 10.3758/s13428-014-0505-8

RadicalLocator: A software tool for identifying the radicals in Chinese characters

Lili Yu · Erik D. Reichle · Mathew Jones · Simon P. Liversedge

Published online: 29 August 2014

Abstract This article describes a new software tool called *RadicalLocator* that can be used to automatically identify (e.g., for visual inspection) individual target radicals (i.e., groups of strokes) in written Chinese characters. We first briefly clarify why this software is useful for research purposes and discuss the factors that make this pattern recognition task so difficult. We then describe how the software can be downloaded and installed, and used to identify the radicals in characters for the purposes of, for example, selecting materials for psycholinguistic experiments. Finally, we discuss several known limitations of the software and heuristics for addressing them.

Keywords Charaters · Chinese · Radicals · Software tool

During the last decade, there has been a rapid increase in the number of psycholinguistic experiments investigating the cognitive processes that are involved in reading Chinese text. (For a review of recent experiments that have used eye movement measures to examine the reading of Chinese text, see Zang, Liversedge, Bai, and Yan 2011). This recent interest undoubtedly reflects an increased appreciation that the Chinese language and writing system have many unique properties that make them ideal for testing specific hypotheses about the perceptual, cognitive, and motor processes that are involved in reading.[1] Perhaps the most salient of these properties is the simple fact that written Chinese is nonalphabetic; rather than being comprised of a set of more basic component letters (as is English and other alphabetic languages), the written forms of Chinese words are instead composed of one or more square-shaped characters. Each of these characters (which number in the thousands) is in turn composed of 1–36 strokes (i.e., simple line segments that historically correspond to the individual brush strokes used to write the characters) that are often arranged or "configured" into one or more radicals (i.e., groups of strokes that occur in many characters).

These features and many others of the Chinese writing system (e.g., individual words are not demarcated by blank spaces) make Chinese written text useful for the purposes of psycholinguistic research. For example, because Chinese is written without spaces between words, experiments using Chinese have been extremely useful for gaining a better understanding of how the perceptual segmentation of words influences lexical processing and saccade targeting during reading (e.g., Bai, Yan, Liversedge, Zang, & Rayner 2008). Similarly, because individual Chinese characters vary in terms of their complexity (i.e., number of strokes) but are the same size, visual-complexity effects can be examined independent of word-length effects in Chinese (e.g., by using single-character words that vary in terms of how many strokes they contain; see Liversedge et al., in press). Finally, some characteristics of the Chinese orthography (e.g., characters are

E. D. Reichle · M. Jones · S. P. Liversedge
University of Southampton, Southampton, UK

L. Yu (✉)
Tianjin Normal University (Balitai Campus),
No. 57 (Zeng 1) Wujiayao Street,
Hexi District, Tianjin City, China 300374
e-mail: lily.yu.96@gmail.com

[1] Spoken Chinese actually consists of many different dialects (with Mandarin and Cantonese being the primary two) that share a common writing system. However, the writing system itself also consists of two variants—the historically older and more complex characters that are still being used in Hong Kong and Taiwan, and the more recent, simplified characters that are being used on the mainland of China. The examples used in this article are based on the latter, simplified characters. (For an accessible introduction to the history and properties of the Chinese language and writing system, see Chang & Chang, 1978.)

comprised of radicals) are themselves unique to the language, and for this reason issues pertaining to how these aspects of the language are represented and processed are fundamental to our understanding of reading and language.

As we have indicated, radicals are important constituent parts of a significant number of Chinese characters. They represent the structure within the character and can have a particular function in relation to conveying aspects of the phonology or meaning of the character itself, and sometimes of the word within which it appears. Figure 1 shows two examples of single-character words that are each composed of exactly two radicals—one that conveys information about the meaning of the word and a second that conveys information about its pronunciation. Such examples might be called "simple" radicals, because their boundaries are unambiguous and they cannot be decomposed into smaller clusters of strokes. However, as N. Wang (1997) indicated, there are also more complex radicals. For example, Fig. 2 also shows two examples of characters that are each composed of two radicals; however, whereas the first character is composed of two simple radicals, the second contains a complex radical that is itself the character in the first example (and thus composed of two simple radicals). As this example illustrates, even N. Wang's definition of "radical" is somewhat ambiguous, because some configurations of strokes can be classified as either radicals or characters, depending on the context in which they occur.

Due to their functional characteristics, the properties of radicals have been manipulated in a number of experiments to investigate the cognitive processes underlying reading (e.g., Chen, Allport, & Marshall 1996; Feldman & Siok, 1999; Leck, Weekes, & Chen 1995; Lee, Tsai, Huang, Hung, & Tzeng 2006; Liu, Inhoff, Ye, & Wu 2002; Yan, Zhou, Shu, & Kliegl 2012; Zhou & Marslen-Wilson, 1999). For example, some studies have demonstrated that knowledge of radicals is important for the learning of Chinese as a second language (e.g., Taft & Chung, 1999; M. Wang, Perfetti, & Liu 2003). Similarly, how often a radical occurs in printed text (i.e., its *token frequency*) affects the time required to identify the character in which it is embedded (e.g., Feldman & Siok, 1997; Taft & Zhu, 1997; Tsang & Chen, 2009). And because radicals can appear at different positions within different characters, the positional frequency of radicals (i.e., their *type frequency*) is also known to influence the processing times associated with the characters and words within which they appear (Ding, Peng, & Taft 2004; Taft, Zhu, & Peng 1999; Wu, Mo, Tsang, & Chen 2012).

On the basis of these studies, there is already ample evidence that radicals play an important functional role in Chinese reading, and that their properties influence the efficiency with which they are processed during normal reading and other tasks used to study reading (e.g., lexical decision and word naming). It is therefore critically important to be able to identify—and thereby to quantify and control—the characteristics of radicals that are used in Chinese psycholinguistic experiments. To be clear, it is vital that we be able to identify the radicals in Chinese characters for the purposes of controlling their properties in any experiment concerned with character and word identification. This is true for experiments that aim to actively manipulate linguistic variables to demonstrate their effects on language processing, as well as for those studies that do not, but that nonetheless require that the extraneous effects of radicals be controlled.

To illustrate this, let us consider a situation that arose in relation to an experiment that we wished to undertake in our laboratory. Although the purposes of this study are not relevant here, the design of the study necessitated the selection of characters that contained specific target radicals from a very

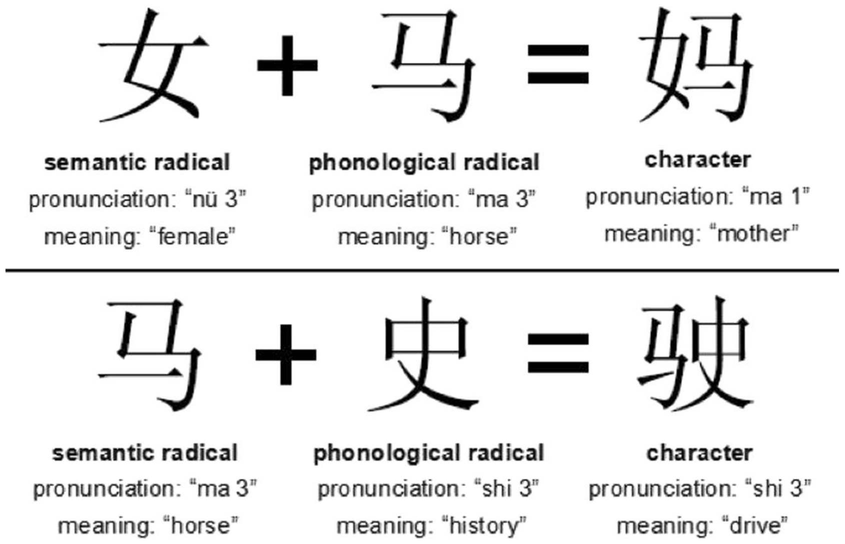


Fig. 1 Two examples showing how single-character words (meaning "mother" in the top panel and "drive" in the bottom panel) are composed of two simple radicals, with one radical conveying semantic information and the other conveying phonological information

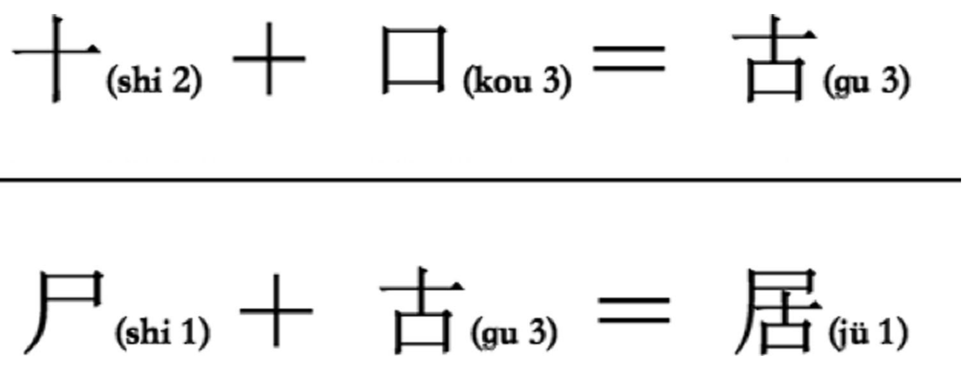

十 (shi 2) + 口 (kou 3) = 古 (gu 3)
尸 (shi 1) + 古 (gu 3) = 居 (jü 1)

Fig. 2 Two examples showing characters composed of radicals of varying complexity. In the top panel, the character 古 is composed of two simple radicals, 十 and 口. In the bottom panel, the character 居 is composed of a simple radical, 尸, and a complex radical, 古, with the latter being the character shown in the top panel

large corpus of characters—a task that essentially amounted to the identification (i.e., detection) of a large number of radicals in specific locations in characters within an even larger number of characters. Although this task might appear to be easy (but tedious), several aspects of the Chinese writing system actually made the task exceedingly difficult.

The first is that, although individual radicals can be described as "configurations" of spatially proximal strokes, these strokes are not necessarily overlapping or even spatially contiguous, but can instead be separated by blank spaces. And depending upon a variety of factors (e.g., the exact character, the font being used, and the writing style and/or skill of a person who is writing the character), the individual strokes that make up a given radical may or may not intersect with or touch the strokes of other radicals within the same character, thereby making it even more difficult to use the spatial contiguity of strokes to localize individual radicals. Furthermore, the precise size (i.e., spatial extent) of any given radical can vary from character to character, depending on the number and nature (e.g., size) of the other radicals that co-occur in the character. This is due to the fact that individual characters are always rendered to occupy approximately the same amount of two-dimensional space, which therefore means that a given radical may be rendered bigger or smaller, depending on both the number and complexity of the other radicals in the character. Finally, depending on the character in which it occurs, the same radical may occupy different spatial locations (e.g., the left side vs. the bottom half of a character), and if the radicals appear in different locations in different characters, they may or may not be the same shape. Figure 3 provides some examples illustrating how these intrinsic properties of the writing system make the task of localizing individual radicals so difficult.

For the reasons just articulated, and because commercial software cannot handle the complexity associated with recognizing individual radicals in corpora of characters, it has been difficult to determine radical positional frequencies. To understand the limitations of commercial software, consider the two options that are currently available for determining how often target radicals occur at different locations within characters. The first involves using radicals (called "Bu-Shou") that are listed in dictionaries for the purposes of indexing (i.e., locating) the characters containing those radicals. Although the number of characters containing a given radical might be used as a proxy measure of the radical's type frequency, any given character is only indexed by a single Bu-Shou radical, thus making it impossible to estimate the type frequency of some radicals. For example, because the character 驶 is only indexed by the radical 马, the frequency of the other radical 史 cannot be estimated.

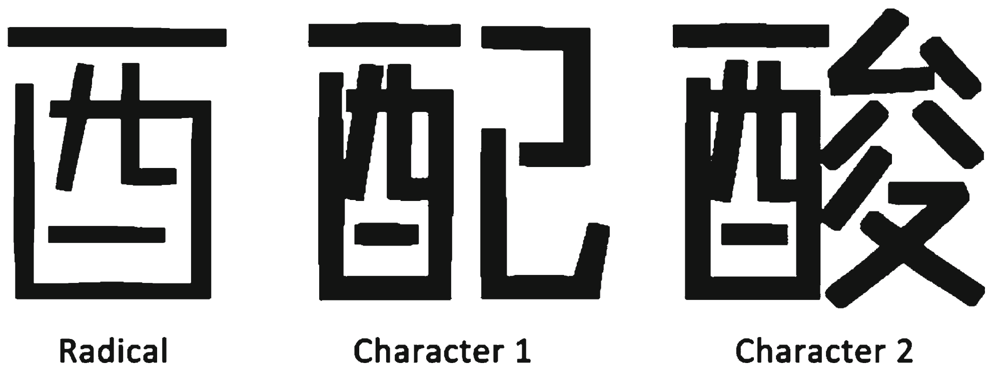


Fig. 3 Example showing a radical (left) composed of non-spatially-contiguous strokes. The size of the radical can also change when it is embedded within a character, as the two examples (center and right) illustrate. The right example also shows how strokes from different radicals within a character can be spatially adjoined, further reducing the demarcation of those radicals. (Note that these examples are rendered in the Mini Jian Yayi font)

A second method for determining how often target radicals occur at different locations involves using techniques that are commonly available in word processing. For example, although one might utilize a commonly used alphabetic system (called "Pinyin") to locate and thereby estimate the type frequency of phonetic radicals, this method is inherently limited, because many characters share the same radical but are pronounced differently (e.g., the characters 妈 and 驶 in Fig. 1). Similarly, although one might utilize Unicode characters (which represent the individual graphemes that are used in a wide variety of languages) to locate specific target radicals, this method is also inherently limited, because a single Unicode character is sometimes used to represent different radicals. For example, the same Unicode character is used to represent the radicals 亻 and 人 (which both mean "human"). Moreover, Unicode also contains both simplified and traditional characters, which makes the task of locating radicals of one type or the other even more difficult.

Because there is currently no accurate way to automatically locate target radicals in characters, and because the manual identification of target radicals for any sizeable corpus of characters (e.g., the type of corpus required for a well-designed psycholinguistic study) would have been prohibitively time consuming and prone to errors (e.g., missing characters because of fatigue), we decided that it was necessary to develop our own software tool to help us perform the task. This software, *RadicalLocator*, provides a useful tool that can be used to perform this function. Not only does the tool allow us to identify characters that contain a radical, but it also allows us to then visually inspect the characters in which a radical appears to determine the radical's locations. Furthermore, by delimiting the corpus to either a list of commonly used characters or a representative text, it is alternatively possible to obtain frequency metrics that are specific to a particular experimental item set (i.e., type frequency counts) or that are representative of language more generally (i.e., token frequency counts). Because this tool has proved useful in this capacity in our research, we have decided to make it freely available to other researchers.

The purpose of this article, therefore, is to describe this software tool and to make it available to the larger psycholinguistic community.[2] The remainder of this article will do exactly that—first describing how to access the software, and then providing an overview of how the software can be installed and used to identify individual radicals in large corpora of Chinese characters.

General overview of RadicalLocator

The main function of this software tool is to automatically identify specific target radicals in corpora of Chinese characters. It does this by matching an image of a specific target radical to a corpus of character images. To do this efficiently, the program first segments each of the characters into a (potentially large) number of two-dimensional "blobs," with each such blob ideally corresponding to an individual stroke in the radical. The software then calculates a number of statistics by analyzing each blob's features. These statistics include, for example, each blob's center of gravity, the degree to which the blob fills a rectangle bounding the blob, and the blob's location and size with respect to other blobs. To determine a match in a target radical and a particular character, the program attempts to match the radical (using the previously mentioned statistics) to each possible permutation of a set of blobs in the character (with the number of blobs at issue being set equal to the number in the target radical). A weighted summation of all the differences in these statistics is calculated, to generate an overall "error" measure between the radical and each character, and the value corresponding to the permutation having the lowest overall error measure is then selected as a possible match. Because the weights used to calculate this error measure are user-configurable (as will be described below), the interpretation of the error score will depend on the criteria used to locate target radicals within a corpus of characters.[3]

Because the program uses different permutations of strokes to construct the blobs, the degree to which individual blobs are spatially distinct from one another is a critical variable that determines how well the program will perform. To understand why this is true, let us return to our previous example in Fig. 3, in which both Characters 1 and 2 contain the radical 酉. In Character 1, the left and right radicals are spatially separated, so that the character is decomposed into distinct blobs that the software can use to localize the two radicals. However, in Character 2 the software will have more difficulty locating the two radicals, because two of the strokes from the right radical are spatially adjoined with strokes of the left radical. As this example illustrates, separating the blobs corresponding to different radicals is one of the key computational challenges that is faced by the software in localizing the individual radicals within characters. To help solve this problem, the software performs an "erosion" operation on the images of radicals and characters to reduce the length and width of individual strokes, thereby separating strokes that would otherwise connect the radicals within a given character. This erosion tool will be described in detail later this article.

Downloading, installing, and starting RadicalLocator

The software, brief instructions for its installation and use, and a 3,500-character corpus of Chinese characters can be

[2] The software is freely available but will not be supported because of resource limitations. It is therefore offered only as an "as is," standalone tool.

[3] Interested users can also examine the RadicalLocator source code to see exactly how the error score is calculated.

downloaded from the second author’s personal website: www.erikdreichle.com/programs.html. The software is written in the C# and C++ programming languages and must be run in a Windows operating system environment. The software also requires either a 32- or 64-bit version of the Visual Studio 2010 SP1 Redistributable and the .NET Framework 4.0. As will be discussed below, the software actually consists of two related tools: (1) RadicalLocator, which is used to locate specific target radicals within corpora of characters, and (2) RadicalProcessor, which is used to assess the effects of the erosion parameters that control the search process in RadicalLocator. Both of these software tools will be described in detail below.

After the software has been downloaded, both it and the file containing the images of Chinese characters (i.e., the corpus of interest) must be placed in a common working directory (e.g., a folder on the computer’s virtual desktop). (The image files can be in .bmp, .gif, .jpg, .png, or .tiff formats.) After this has been done, one can start the program by simply double-clicking on the program icon to open a *graphical user interface* (GUI), or a window that allows the user to run and interact with the program. The opening window of the GUI is shown in Fig. 4a.

As Fig. 4a shows, the right panel is labeled “Reference Image” and is used to display the image of a selected target radical, taken from the working dictionary that contains images of all of the target radicals of interest. In a similar manner, the left panel of the GUI is labeled “Dictionary Image” and is used to display the character images from the working dictionary that has been selected for input. The figure shows an example in which the “Reference Image” panel displays the image of the target radical 马 and the “Dictionary Image” panel displays the character 冯.

The top bar of the GUI contains three buttons representing menus: File, Option, and Help. The Help menu shows the version of the software and copyright information; the other menus will be discussed below.

The bottom bar of the GUI also contains three indicators that provide information about the running status of the software. From left to right, these indicators are (1) an error measure, indicating how well the target radical matches the current character; (2) a Boolean measure indicating whether the error measure exceeds some threshold (set by the user) indicating that the current character contains the target radical; and (3) a unique identifier image number for each of the character images in the dictionary. In the example shown, the error measure is 0.5464, which (because it is less than the threshold of 0.6 that was set by the user) results in the signal FOUND being displayed, indicating that the character 冯 (which is the 276th of the 3,500 contained in the dictionary) does contain the radical 马. (An error measure exceeding the threshold would result in a “——” signal being displayed, indicating that the character does not contain the target radical.)

Creating a new project

The File menu includes three options that can be used to (1) create a new project (“New Project Wizard”), (2) export data to a file (“Export Data”), or (3) exit the program (“Exit”). Each of these options will be discussed in turn.

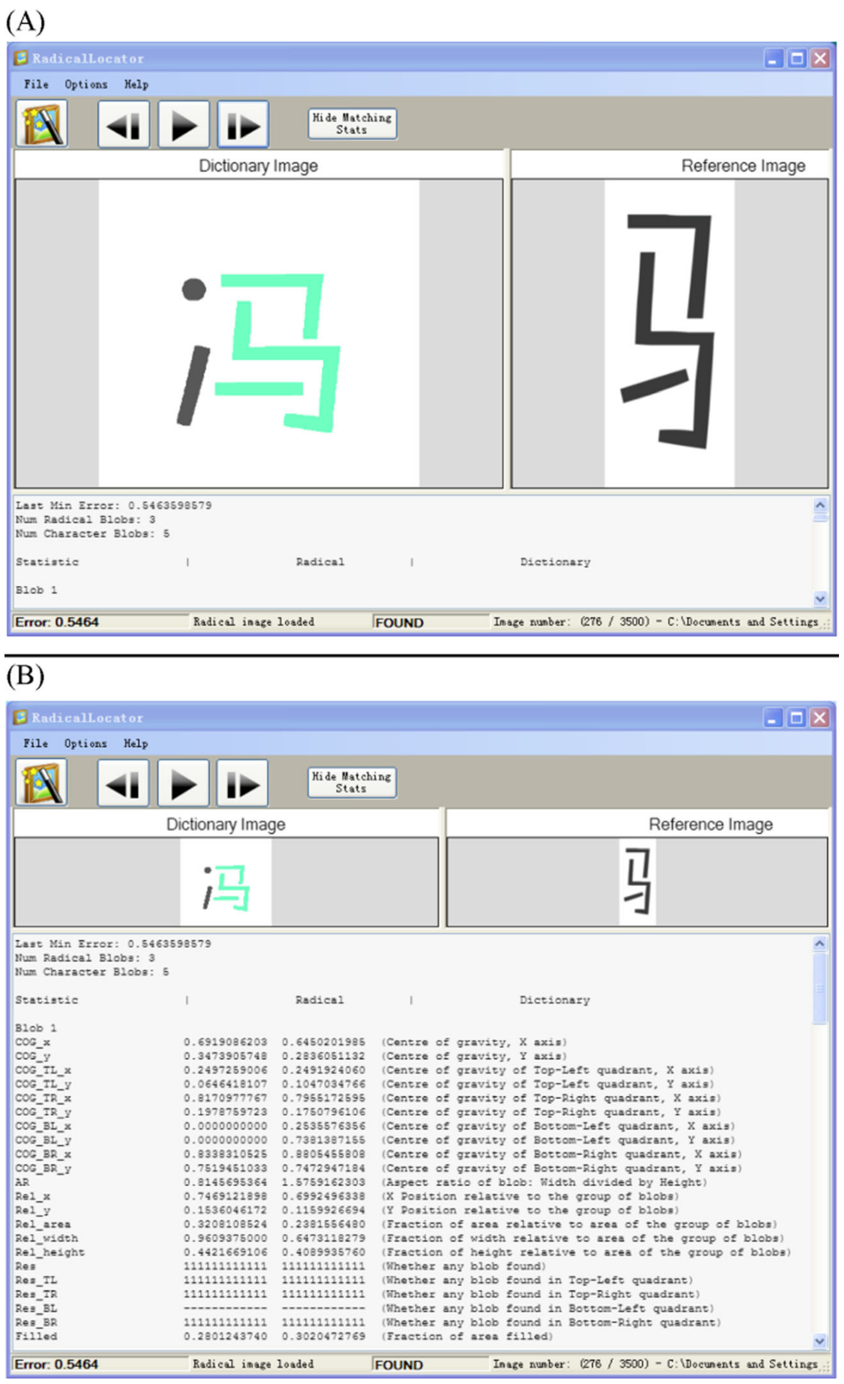
The first step to create a new project is to configure the input file. To do this, click “New Project Wizard” to open a window that prompts the user to specify the directory and file containing the dictionary (i.e., corpus) of character images. After this has been done, click “Next” to open the window that prompts the user to specify the name(s) of the file(s) containing the image file(s) of the target radical(s). Any number of target radicals can be added by simply clicking the “Add” button to append additional images to the list contained in the “Image Filenames” box. After this has been done, click “Next” to specify the directory and filename to which the data will be exported. Although the data are exported automatically upon reaching the end of the dictionary (i.e., after comparing the target radical to every possible character), this menu option can also be used to export the data prior to that point (e.g., if the user is only interested in a subset of the dictionary). Finally, click “Finish” to complete the configuration of the new project.

Software tools and search parameters

Returning to the opening window (see Fig. 4a), the Options menu allows the user to set parameters that control the manner in which RadicalLocator searches for target radicals within characters, and contains tools for using the results of such searches. These options include (1) “Create Dictionary from Images,” (2) “Set Decision Params,” (3) “Set Decision Weights,” (4) “Set Erode Params,” and (5) “Verify Results.” The first four of these options will be discussed in this section; the fifth will be discussed in the next section.

One can generate image files for a new dictionary of characters. This can be done by clicking “Create Dictionary from Images” to specify the input file(s) that will be used to create the dictionary, such as the images in the example shown in Fig. 5, and the output file containing the results. The input can be drawn from either a single file or multiple files, by clicking either “Single File as Input” or “All Files in Directory as Input” and then specifying the input file(s). All of the parameters in the bottom dialog box can be adjusted by either clicking the scrolling triangles or typing the desired values directly into the boxes. The file names for the single character images that are generated in this manner will be generated automatically.

One can also adjust the values of the parameters that control the search-decision parameters, or how closely a given “blob” within a character in the dictionary has to match the target radical in order for RadicalLocator to tag the character



```
Last Min Error: 0.5463598579
Num Radical Blobs: 3
Num Character Blobs: 5

Statistic            |            Radical            |            Dictionary

Blob 1
```

```
Error: 0.5464    Radical image loaded    FOUND    Image number: (276 / 3500) - C:\Documents and Settings
```

```
Last Min Error: 0.5463598579
Num Radical Blobs: 3
Num Character Blobs: 5

Statistic            |            Radical            |            Dictionary

Blob 1
COG_x                0.6919086203  0.6450201985  (Centre of gravity, X axis)
COG_y                0.3473905748  0.2836051132  (Centre of gravity, Y axis)
COG_TL_x             0.2497259006  0.2491924060  (Centre of gravity of Top-Left quadrant, X axis)
COG_TL_y             0.0646418107  0.1047034766  (Centre of gravity of Top-Left quadrant, Y axis)
COG_TR_x             0.8170977767  0.7956172595  (Centre of gravity of Top-Right quadrant, X axis)
COG_TR_y             0.1978759723  0.1750796106  (Centre of gravity of Top-Right quadrant, Y axis)
COG_BL_x             0.0000000000  0.2535576356  (Centre of gravity of Bottom-Left quadrant, X axis)
COG_BL_y             0.0000000000  0.7381387155  (Centre of gravity of Bottom-Left quadrant, Y axis)
COG_BR_x             0.8338310525  0.8805455808  (Centre of gravity of Bottom-Right quadrant, X axis)
COG_BR_y             0.7519451033  0.7472947184  (Centre of gravity of Bottom-Right quadrant, Y axis)
AR                   0.8145695364  1.5759162303  (Aspect ratio of blob: Width divided by Height)
Rel_x                0.7469121898  0.6992496338  (X Position relative to the group of blobs)
Rel_y                0.1536046172  0.1159926694  (Y Position relative to the group of blobs)
Rel_area             0.3208108524  0.2381556480  (Fraction of area relative to area of the group of blobs)
Rel_width            0.9609375000  0.6473118279  (Fraction of width relative to area of the group of blobs)
Rel_height           0.4421669106  0.4089935760  (Fraction of height relative to area of the group of blobs)
Res                  111111111111  111111111111  (Whether any blob found)
Res_TL               111111111111  111111111111  (Whether any blob found in Top-Left quadrant)
Res_TR               111111111111  111111111111  (Whether any blob found in Top-Right quadrant)
Res_BL               ------------  ------------  (Whether any blob found in Bottom-Left quadrant)
Res_BR               111111111111  111111111111  (Whether any blob found in Bottom-Right quadrant)
Filled               0.2801243740  0.3020472769  (Fraction of area filled)
```

```
Error: 0.5464    Radical image loaded    FOUND    Image number: (276 / 3500) - C:\Documents and Settings
```

Fig. 4 The opening window of the RadicalLocator GUI. Panel A shows that the window contains a "Reference Image" displaying a target radical, a "Dictionary Image" displaying a character that (in this example) contains the radical, and a window indicating the program status. Panel B shows the same window after the bar separating the top and bottom sections has been moved up to display more information about the program status

as containing the radical. To do this, click "Set Decision Params" to open a window showing the maximum value of the error measure between a target radical and a character (see Fig. 4a) that must not be exceeded in order for the software to tag the character as containing the radical. The default value of the threshold parameter is 1, but one can make the criterion for detecting a radical within a character more or less conservative by, respectively, decreasing or increasing the value of this parameter. Larger values of the threshold increase the numbers of both hits (i.e., characters that are correctly identified as containing a target radical) and false alarms (i.e., characters that are incorrectly identified as containing a target radical), whereas smaller values of the threshold have the opposite effect (see the Appendix). Finally, the box labeled "Use

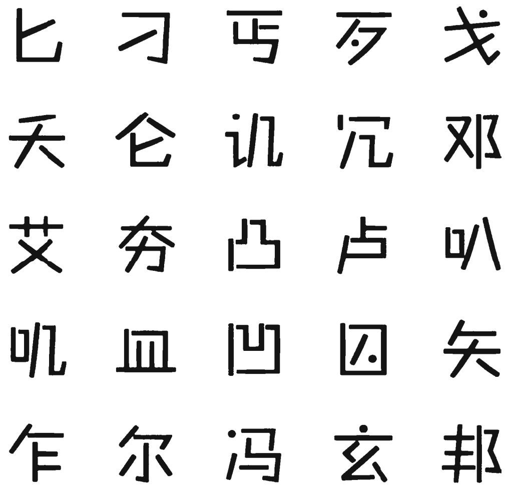


Fig. 5 Example input image containing 25 different characters

Special Parameters" can be checked in those rare instances when the characters being searched contain only a single blob; in such instances, one can set the values of special parameters to further control how the radical search is completed. (Note that this special mode of search is automatically rendered ineffective whenever the characters being searched contains more than a single blob.)

The values of the aforementioned search-decision parameters can be adjusted by clicking "Set Decision Weights" to open a window showing the values of parameter weights that can be applied to the various blob features (e.g., width), allowing them to play a greater or lesser role in the final error measure that is generated for a particular character. Table 1 lists these parameters and provides brief descriptions of their functions.

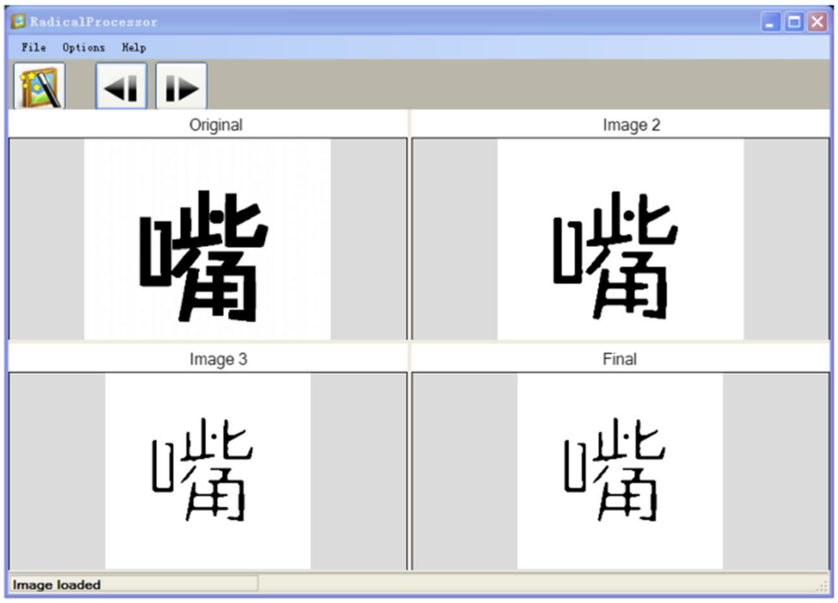
It is also possible to display additional information in a dialog box about, for example, how well a target radical matches each particular blob within a character, by clicking the "Show Matching Status" button shown in Fig. 4a. (After the button is clicked, its label changes to "Hide Matching Status.") The dialog box can also be enlarged (as is shown in Fig. 4b) by moving the bar separating the box from the rest of the display upward. This allows for more easy inspection of the information contained in the dialog box. As Fig. 4b shows, the dialog box provides detailed information about each of the parameters used to match character blobs to a target radical. For a radical to be considered a match, the values should be similar. If there is a large difference in these values, however, it could mean that the weights were not specified well for this character set, or that the program has matched the wrong blob (if a matching blob even exists).

Finally, as we indicated previously, RadicalLocator uses a heuristic based on the "erosion" of line segments to help it separate radicals that might otherwise be connected (and thus treated as a single radical), due to strokes that overlap or are spatially adjoined. It is possible to specify exactly how this should be done by clicking "Set Erode Params" to open a window labeled "Erode Parameters." Two variations of the

Table 1 The search parameters and brief descriptions of their functions

Parameters	Descriptions of Functions
Blob parameters	**These shape and position parameters are applied irrespective of the number of blobs in the reference radical.**
Blob	Overall weight applied to all parameters except blob shape and position, including the blob's center of gravity and the centers of gravity of the four quadrants
Position	Blob position, in relation to other blobs
Width	Blob width, as percentage of entire radical width
Height	Blob height, as percentage of entire radical height
Area	Blob area, relative to total area of all blobs in radical
Single-blob parameters	**These special weights are applied only if there is one blob in the reference radical.**
Blob single	Overall weight applied to all blob parameters except shape and position
Fill single	Percentage of bounding rectangle filled by blob
AR single	Aspect ratio of blob (i.e., width divided by height)
Sobel parameters	**If a horizontal or vertical Sobel operator is applied to an image, then the image is reduced to a one-dimensional image of averaged pixel values representing the presence of any horizontal or vertical lines in the blob**
Sobel mean	Mean value of Sobel image
Sobel percentage	Percentage of pixels greater than mean in Sobel image
Sobel L98	98% of pixel values are less than this threshold in Sobel image
Sobel num peaks	Number of peaks identified in Sobel image (i.e., number of horizontal or vertical lines)
Sobel peak pos	Position of maximum value in Sobel image (e.g., top, middle or bottom)
Sobel correlation	Correlation between Sobel image and one-dimensional image of reference radical (i.e., measure of how well the Sobel images match each other)

(A)



(B)

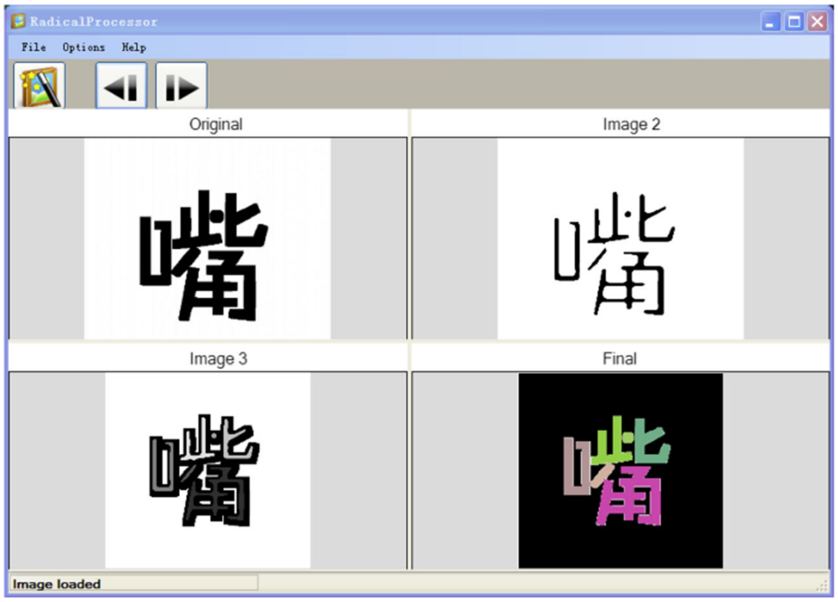


Fig. 6 Two examples illustrating the effects of erosion. Panels A and B, respectively, show the effects of standard and watershed erosion on a single character

erosion tool can then be used to help separate individual radicals within a character: (1) standard erosion and (2) watershed erosion. These two variants can be selected with radio buttons, and the parameters controlling the methods can be adjusted by clicking the scrolling triangles. Each variant will now be discussed in turn.

Figure 6 shows the GUI window of the supplementary RadicalProcessor software, which can be used to visually inspect the effects of the erode operator on individual characters. As the figure shows, the window contains four quadrants (panels) that display the progressive effects of erosion on a character, starting with the original (i.e., unmodified) character and ending with the final (i.e., fully eroded) character through two successive stages.

Figure 6a shows how standard erosion can be applied to a single character. The erosion operator is applied exactly twice, in conjunction with two Gaussian-blurring stages and two threshold operations. With each iteration, the lines of the character strokes become progressively thinner and shorter, allowing the radicals to “emerge” from the overall configuration of strokes that comprise the character. The panel labeled “Original” shows the original image of character, before any of the erosion operations. The panels labeled "Image 2" and "Image 3" show images of the character after the first and second applications of the erosion operation, and the panel labeled "Final" shows the final image. Because the erosion operation is applied exactly twice, the image in this last panel will be the same as the image in the "Image 3" panel.

However, if the watershed erosion method is instead selected, an extra, "watershed" operation will be applied to the image in the "Image 3" panel, producing the image shown in the "Final" panel of Fig. 6b. This watershed operation is a marker-based object segmentation image-processing algorithm that involves two stages. The first attempts to mark the objects to be segmented (i.e., the centers of the individual strokes to be separated in the radical), and the second attempts to determine the boundaries between these objects. This second stage is done by using morphological operators (e.g., erosion or dilation) to flood-fill the region around each marker whilst avoiding the filling of neighboring regions. This process is performed until only a one-pixel border around all of the marked regions remains unfilled, corresponding to the boundaries. The erosion operator is then used to identify the markers, and the resulting boundaries are used to separate the individual strokes.

Although erosion provides a useful tool for separating radicals that would otherwise be treated as a single radical, due to spatially adjoined strokes, the operation is not without its own limitations. For example, prior to using erosion, it is important to explore the effects of the erosion parameters on several characters, to avoid, for example, using parameter values that might cause entire strokes within the characters to disappear. This should be done using the RadicalProcessor tool on characters with a large (e.g., more than 20) number of strokes, because such characters are the most susceptible to this problem, and thus the most diagnostic of parameter values that are too extreme. In our experience, it has often proved better to use whatever minimal erosion is necessary to separate the radicals of characters in a given corpus, knowing in advance that such parameters may not be able to separate all radicals within a corpus of characters. For example, the software can have difficulty separating radicals that are overlapping or spatially adjoined (e.g., the examples shown in Figs. 2 and 7).

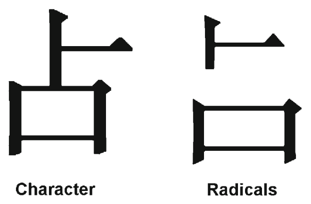


Fig. 7 Example showing a character composed of two (deliberately) connected radicals that cannot be decomposed by using erosion

Running RadicalLocator

To run the program, return to the main GUI (see Fig. 4a). At the top of this GUI are three arrow buttons that (going from left to right) are used to (1) move back to (i.e., display) the previous character, (2) run the program, and (3) jump ahead to (i.e., display) the next character. Clicking the central arrow button therefore starts the program, causing it to run and causing the "Play" icon to change into a black square that, if clicked, will stop the program from running. Finally, to exit the program, click "Exit" to close all of the RadicalLocator windows and any file containing search results.

Verifying results

A file containing the results of a search will be automatically created after the RadicalLocator program has finished running. The file will be formatted as a standard text file (i.e., .txt file extension), which allows it to be examined using Microsoft Office Excel or any other spreadsheet or text editor software. This allows one to, for example, sort the results on the basis of error-matching scores, so that the characters providing the best potential match to a target (i.e., the ones having the smallest error values) can be quickly identified. At the top of the file is a header containing the file name, the creation time of the results file, the file name containing the target radical, the total number of characters in the corpus file, and the number of matches (i.e., the total number of radicals that exceeded the threshold for matching a character in the corpus). The remainder of the file is organized into columns, as follows: (1) search trial number (i.e., the number of the character in the corpus), (2) whether or not the match was successful (coded as "1" for matches and "0" for mismatches), (3) the value of the match error measure, and (4) the filename for the images of the characters. Because the search process is not perfect, however, it is important to verify the search results by visual inspection, as will be described next.

To verify the results, return to the main GUI and click "Verify Results" using the results file that was generated by the search. After this has been done, the program will open the window labeled "Verify Results," which allows the user to verify search results by scrolling through the list of potential matches, examining the character associated with each potential match, and then selecting those characters by checking the boxes labeled "Found" (as appropriate). After the correctly identified characters have been verified (selected) in this manner, click "OK" to save the updated results. The user will then be prompted to specify a filename and directory for the file containing the new results (i.e., whatever changes have been made to the original results file).

Conclusions

In our experience, RadicalLocator has proven to be a useful software tool that can be used to automatically identify target radicals in large corpora of written Chinese characters. Although the software was developed with this specific application in mind, the software might also be useful in other image scanning and matching applications. For example, one application would be the identification of orthographically similar characters for the purposes of developing materials for a priming experiment using Chinese characters. As this example illustrates, the software provides a basic tool, but the utility of this tool is limited mainly by the goals and/or creativity of its user.

However, as we already discussed in relation to erosion, there are still two known limitations of this software. First, because the software operates by calculating potential matches between a given target radical and a large number of character blobs, the calculations that are performed by the software are computationally expensive, making it relatively slow. For example, it takes approximately 20 min to locate one target radical in a dictionary of 3,500 characters. Second, as was indicated in our discussion of erosion, it may be difficult for the software to locate radicals that are spatially contiguous with other radicals (see Figs. 2 and 7).

Although the first limitation is difficult to remedy (we can only advise users to run RadicalLocator on modern computers that have fast microprocessors), we have developed several heuristics that are useful for minimizing the adverse consequences of the second limitation. First, because the location of a target radical is one of the parameters that the software uses (in addition to blob height, etc.) to perform its search, it is often useful to render separate images of the target radical—with one apiece for various possible within-character locations in which the radical might occur. Second, it is better to use a font that minimizes the amount of stroke overlap between radicals. On the basis of our experiences, we recommend using Mini Jian Yayi, Mini Jian Xihei, or Mini Jian Texidengxian. And finally, for instances involving connected radicals like the ones illustrated in Fig. 7, it is better to render an image of the complex radical (i.e., one containing both radicals) and to simply use this complex image to locate the radical of interest. Using these three heuristics, it is possible to minimize the problems associated with finding radicals that—for various reasons—might overlap with other radicals.

Appendix

To evaluate the accuracy of RadicalLocator, we examined the hit rates, false-alarm rates, and miss rates that resulted from searching for ten target radicals in a corpus of 100 characters using watershed erosion. (We deliberately used a small number of radicals and characters so that we could count the number of times that each radical occurred in the corpus by hand, thereby allowing us to accurately evaluate the software's performance.) We conducted three searches for each radical, using threshold values corresponding to the minimal error value between a given radical and all of the characters in the corpus (θ in Table 1), along with two other threshold values: θ+1 and θ+2. The values of the thresholds were set for each radical in this way because the radicals varied in terms of their distinctiveness, and thus differed in terms of how readily they could be discriminated from other radicals/characters (e.g., more distinctive radicals are less visually

Table 2 Hit rates, false-alarm rates, and miss rates resulting from ten target-radical searches using the indicated threshold values

Target Radicals	# of Characters Containing Target Radical	Thresholds	Hits	Misses	False Alarms
豆	1	θ (=0.3)	0	1	0
		θ+ 1 (=1.3)	1	0	5
		θ+ 2 (=2.3)	1	0	30
立	2	θ (=0.8)	0	2	0
		θ+ 1 (=1.8)	2	0	15
		θ+ 2 (=2.3)	2	0	52
艹	11	θ (=1.4)	0	11	0
		θ+ 1 (=2.4)	9	2	1
		θ+ 2 (=3.4)	11	0	0
爫	4	θ (=0.3)	0	4	0
		θ+ 1 (=1.3)	2	2	5
		θ+ 2 (=2.3)	3	1	45
田	4	θ (=0.2)	0	4	0
		θ+ 1 (=1.2)	3	1	0
		θ+ 2 (=2.2)	3	1	9
更	2	θ (=1.2)	0	2	0
		θ+ 1 (=2.2)	1	1	2
		θ+ 2 (=3.2)	2	0	18
口	19	θ (=1.3)	0	19	0
		θ+ 1 (=2.3)	8	11	1
		θ+ 2 (=3.3)	16	3	17
日	14	θ (=2.1)	0	14	0
		θ+ 1 (=3.1)	8	6	8
		θ+ 2 (=4.1)	13	1	57
木	10	θ (=0.2)	0	10	0
		θ+ 1 (=1.2)	9	1	12
		θ+ 2 (=2.2)	10	0	77
寸	2	θ (=0.3)	0	2	0
		θ+ 1 (=1.3)	2	0	56
		θ N 2 (=2.3)	2	0	91

θ= minimum error between target radical and the 100 characters in the corpus similar to other radicals/characters, and therefore produce smaller error scores than do less distinctive radicals). Our rationale for using this particular test was that it is objective and provides an estimate of the both the effort and accuracy that one might expect when using the software, as will be explained below.

Table 2 shows the results of this test. Each row corresponds to one of the ten radicals and one of the three threshold values used to identify possible characters containing the radicals. Each row also shows the number of characters in the corpus that actually contained the radical, the number of characters containing the radical that the software identified (i.e., the number of hits), the number of characters containing the radical that the software failed to identify (i.e., the number of misses), and the number of characters incorrectly identified as containing the radical (i.e., the number of false alarms).

To evaluate the test results in Table 1, imagine that a user of our software had adopted a simple heuristic of using threshold values equal to $\theta+1$ to conduct their search. Using this threshold value, our test indicated that the mean expected hit and miss rates would be .75 and .25, respectively. This fairly high level of accuracy would, of course, be offset by the fact that, using this heuristic, .53 of the characters identified as containing the target radicals would not. Although this false-alarm rate might appear unacceptably high, it can be reduced (by accepting a lower hit rate) in applications for which it is not critical to identify all of the characters containing a specific radical. Furthermore, the simple heuristic used for the purposes of our test becomes increasingly efficient with the size of the character corpus, because it requires the user to visually inspect and validate only a relatively small pool of candidate characters. The efficiency of this heuristic can be appreciated if it is contrasted with the alternative method for locating the target radicals—visually inspecting all of the characters in the corpus for each target radical of interest.

References

Bai, X., Yan, G., Liversedge, S. P., Zang, C., & Rayner, K. (2008). Reading spaced and unspaced Chinese text: Evidence from eye movements. *Journal of Experimental Psychology: Human Perception and Performance, 34,* 1277–1287. doi:10.1037/0096-1523.34.5.1277

Chang, R., & Chang, M. S. (1978). *Speaking of Chinese*. New York: Norton.

Chen, Y. P., Allport, D. A., & Marshall, J. C. (1996). What are the functional orthographic units in Chinese word recognition: The stroke or the stroke pattern? *Quarterly Journal of Experimental Psychology, 49A,* 1024–1043.

Ding, G., Peng, D., & Taft, M. (2004). The nature of the mental representation of radicals in Chinese: A priming study. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 30,* 530–539. doi:10.1037/0278-7393.30.2.530

Feldman, L. B., & Siok, W. W. T. (1997). The role of component function in visual recognition of Chinese characters. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 23,* 776–781. doi:10.1037/0278-7393.23.3.776

Feldman, L. B., & Siok, W. W. (1999). Semantic radicals contribute to the visual identification of Chinese characters. *Journal of Memory and Language, 40,* 559–576.

Leck, K. J., Weekes, B. S., & Chen, M. J. (1995). Visual and phonological pathways to the lexicon: Evidence from Chinese readers. *Memory & Cognition, 23,* 468–476.

Lee, C. Y., Tsai, J. L., Huang, H. W., Hung, D. L., & Tzeng, O. J. (2006). The temporal signatures of semantic and phonological activations for Chinese sublexical processing: An event-related potential study. *Brain Research, 1121,* 150–159.

Liu, W., Inhoff, A. W., Ye, Y., & Wu, C. (2002). Use of parafoveally visible characters during the reading of Chinese sentences. *Journal of Experimental Psychology: Human Perception and Performance, 28,* 1213–1227. doi:10.1037/0096-1523.28.5.1213

Liversedge, S. P., Zang, C., Zhang, M, Bai, X., Yan, G., & Drieghe, D. (in press). The effect of visual complexity and word frequency on eye movements during Chinese reading. *Visual Cognition*.

Taft, M., & Chung, K. (1999). Using radicals in teaching Chinese characters to second language learners. *Psychologia, 42,* 243–251.

Taft, M., & Zhu, X. (1997). Submorphemic processing in reading Chinese. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 23,* 761–775. doi:10.1037/0278-7393.23.3.761

Taft, M., Zhu, X., & Peng, D. (1999). Positional specificity of radicals in Chinese character recognition. *Journal of Memory and Language, 40,* 498–519.

Tsang, Y. K., & Chen, H. C. (2009). Do position-general radicals have a role to play in processing Chinese characters? *Language and Cognitive Processes, 24,* 947–966.

Wang, N. (1997). The structure and the meaning of Chinese character and word. In D. Peng, H. Shu, & H. C. Chen (Eds.), *Cognitive research on Chinese language* (pp. 35–49). Shangdong: Shangdong Educational Publishers.

Wang, M., Perfetti, C. A., & Liu, Y. (2003). Alphabetic readers quickly acquire orthographic structure in learning to read Chinese. *Scientific Studies of Reading, 7,* 183–208.

Wu, Y., Mo, D., Tsang, Y. K., & Chen, H. C. (2012). ERPs reveal sublexical processing in Chinese character recognition. *Neuroscience Letters, 514,* 164–168.

Yan, M., Zhou, W., Shu, H., & Kliegl, R. (2012). Lexical and sublexical semantic preview benefits in Chinese reading. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 38,* 1069–1075.

Zang, C., Liversedge, S. P., Bai, X., & Yan, G. (2011). Eye movements during Chinese reading. In S. P. Liversedge, I. D. Gilchrist, & S. Everling (Eds.), *Oxford handbook on eye movements* (pp. 961–978). Oxford, UK: Oxford University Press.

Zhou, X., & Marslen-Wilson, W. (1999). The nature of sublexical processing in reading Chinese characters. *Journal of Experimental Psychology: Learning, Memory, and Cognition, 25,* 819–837. doi:10.1037/0278-7393.25.4.819