Computer-Assisted Language Comparison in Practice

Tutorials on Computational Approaches to the History and Diversity of Languages

Computing Distances Between Japanese Kanji from Stroke Order and Pen Movements

Alessandro Mantelli
Ca' Foscari University
Venice

KanjiVG provides vector drawings of Japanese kanji in which individual strokes are stored separately and in their prescribed order. This information makes it possible to compare characters not only as static images, but also as sequences of drawing movements. This article introduces a straightforward way to calculate distances between kanji using their SVG paths. Each stroke is represented by a small set of points, and the movements between successive strokes are recorded. A sequence-alignment procedure then compares the resulting representations, allowing characters with different stroke counts to be compared. The method is illustrated with a short JavaScript implementation applied to six characters from a fixed KanjiVG release. In this small set, each character's nearest neighbor is its intended visually similar partner (未/末, 木/本, 日/目), and the optimal alignments identify the strokes that distinguish the members of each pair. The procedure is intended as an accessible starting point for exploring stroke-based character similarity, rather than as a validated model of human visual perception.

1 Introduction

Japanese kanji can resemble one another in ways that are difficult to describe using only their number of strokes or their component structure. For example, 未 and 末 have the same number of strokes and a similar overall arrangement, but the relative lengths and positions of their horizontal strokes distinguish the two characters. Other pairs, such as 木 and 本, differ in stroke count while sharing much of their general shape.

A computer can compare characters as images, but image comparison does not necessarily make use of information about how they are written. The open KanjiVG project (Apel, n.d.) provides a useful alternative: it represents individual characters as Scalable Vector Graphics (SVG), with separate paths for the strokes. These paths retain information about stroke shapes, positions, and drawing order.

Computer-Assisted Language Comparison in Practice
Volume 9, Number 2
URL: https://calc.hypotheses.org/9633 DOI: 10.15475/calcip.2026.2.5
Published under a Creative Commons Attributions 4.0 LICENSE
Published on 09/11/2026

Measuring similarities between kanji can have several practical applications. For example, it may help identify visually similar characters in dictionary lookup systems, find potentially confusing kanji, or group characters for educational purposes. While stroke geometry provides information about the visible shape of a character, pen-up movements also describe how the pen moves between successive strokes during writing. This additional information could be useful in applications concerned with the writing process, such as stroke-order learning tools or guided tracing activities, where the movements between strokes may be relevant even though they leave no visible marks.
This article presents a small computational experiment: how can the information in KanjiVG be converted into a numerical distance between two kanji? The proposed procedure compares both the strokes themselves and the movements required to move from one stroke to the next. The aim is not to propose a new general-purpose recognition system. Rather, it is to demonstrate how a publicly available writing resource can be used for a transparent and reproducible comparison of characters.

2 Background

An SVG drawing describes shapes as vector paths. A path contains drawing instructions specifying straight or curved segments, which a browser can render at different sizes. In a raster image, the final arrangement of pixels is the main source of information. In a stroke-based SVG resource, the individual paths can also be examined independently.
KanjiVG was developed to represent the shapes and stroke order of Japanese characters (Apel, n.d.). Its SVG files contain paths corresponding to individual strokes, accompanied by identifiers and, in many files, additional structural annotations. Importantly, the order of the stroke paths can be used as an ordered sequence: the first path represents the first stroke, the second path the second stroke, and so on. This suggests two complementary ways of comparing characters. First, strokes may be compared by their positions and shapes. Second, the movement from the end of one stroke to the beginning of the next can be compared by considering the distance and direction from the endpoint of the current stroke path to the starting point of the next. This movement, known as a *pen-up movement*, is not drawn but can be inferred from the SVG paths.

Stroke order has also been used in previous studies of kanji similarity. Apel and Quint (2004) described an early approach to representing kanji through ordered strokes and SVG paths. Yencken and Baldwin (2008) later used Apel and Quint's stroke data to investigate different measures of kanji similarity. They found that this approach worked well for identifying very similar characters, while a method based on radicals and general shape better reflected human similarity judgements. The method presented here

also compares ordered strokes, but uses their SVG geometry and pen-up movements rather than stroke types alone. This allows strokes of the same type to be distinguished by their length, position and direction. The comparison also preserves the original stroke order.

3 Materials and Methods

3.1 Materials: KanjiVG SVG Files

The input consists of individual SVG character files from the KanjiVG repository, release r20260714. Each character is identified by its Unicode code point, and its strokes are expressed as SVG `<path>` elements. The examples in this article use six ordinary, non-variant character files: 未 (0672a.svg), 末 (0672b.svg), 木 (06728.svg), 本 (0672c.svg), 日 (065e5.svg) and 目 (076ee.svg). All six files use a viewBox of 0 0 109 109, which defines a common drawing area with coordinates ranging from 0 to 109 on both axes. For a character with four strokes, for example, the relevant information can be represented schematically as shown in Listing 1.

```
<svg viewBox="0 0 109 109">
  <path id="stroke1" d="..." />
  <path id="stroke2" d="..." />
  <path id="stroke3" d="..." />
  <path id="stroke4" d="..." />
</svg>
```

Listing 1: Schematic illustration of an SVG containing four stroke paths; this is not an excerpt from an actual KanjiVG file.

In the actual files, the stroke paths are contained in a group whose identifier begins with `kvg:StrokePaths`, while the stroke numbers shown in the drawings are `<text>` elements in a separate group beginning with `kvg:StrokeNumbers`. The central part of each path is the `d` attribute. Instead of interpreting its drawing commands manually, JavaScript can use the browser's built-in methods to obtain the path length and the coordinates of points along it. To ensure that these methods return usable values, the SVG should be inserted into the page before processing.

3.2 Methods: Representing Strokes and Movements

The procedure has three steps. **First, sample each stroke.** Twelve points are sampled at equal intervals along the length of every SVG path, including its start and end. Their coordinates are divided by the width and height of the SVG viewBox, so that all points

are expressed in a common coordinate system. The sequence of sampled points retains an approximation of the stroke's position, direction, and shape.

Second, record pen-up movements. For each stroke after the first, the difference between its starting point and the ending point of the preceding stroke is calculated. This produces a two-dimensional vector. The first stroke has no preceding movement, so its movement vector is set to zero. These vectors are calculated from each original character sequence and are not recalculated when strokes are inserted or deleted during alignment.

Third, compare two stroke sequences. Each character is represented as an ordered sequence of strokes. To compare two characters, the algorithm identifies which strokes can be paired while preserving their original order. For example, 木 has four strokes and 本 has five. Their first four strokes can be paired, while the additional horizontal stroke of 本 remains unmatched. Each possible pairing is evaluated by comparing the geometry of the strokes and the movements between them. The combination producing the lowest overall cost is selected. This procedure is known as *sequence alignment* (Wagner and Fischer 1974).

The following figure illustrates the two comparison settings used in this study. On the left, only the geometry of the drawn strokes is compared. On the right, the comparison also includes the pen-up movements between consecutive strokes.

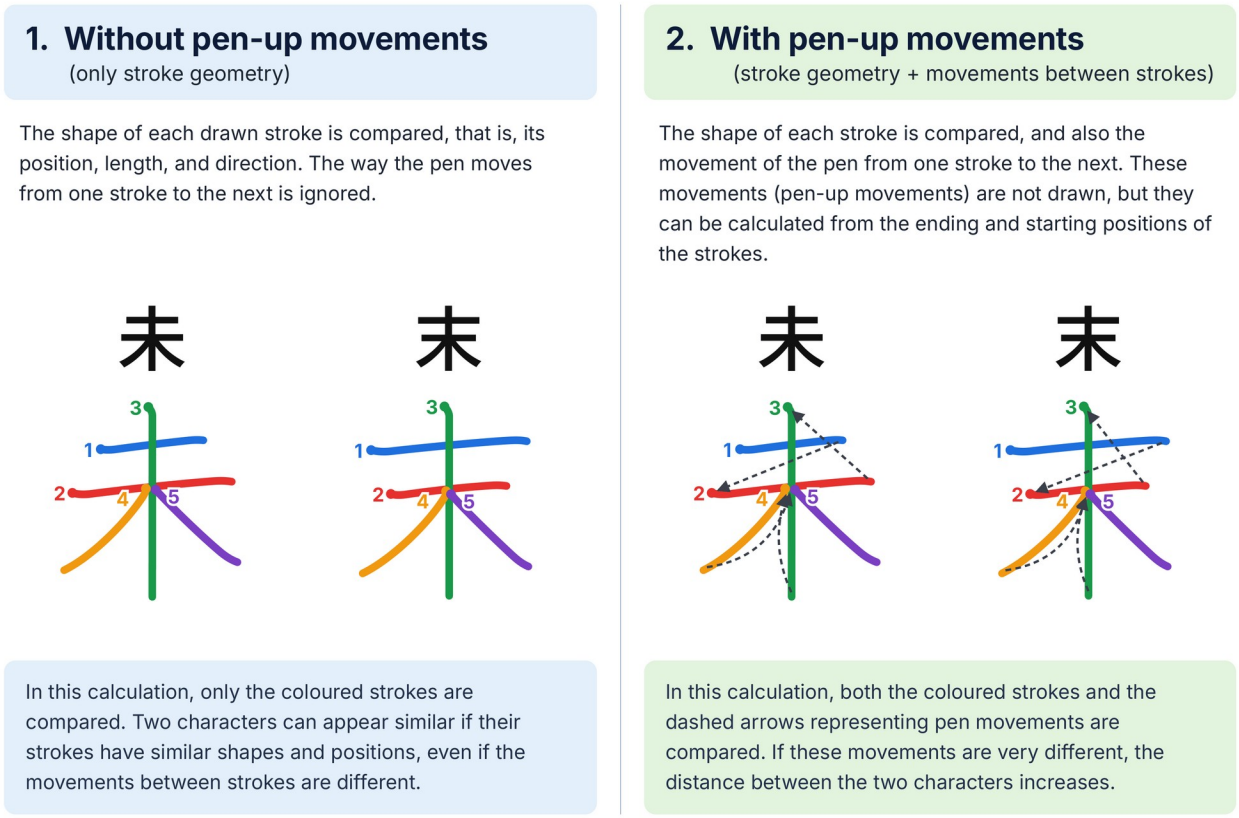


Figure 1. Schematic illustration of the two comparison settings used in this study. **Left:** only the geometry of the drawn strokes is compared. **Right:** the comparison also includes pen-up movements, shown as dashed arrows between consecutive strokes.

The JavaScript code shown in Listing 2 extracts the necessary information from an SVG element. It selects the paths in the stroke-path group and assumes that they occur in stroke order in the document, as they do in the KanjiVG files used here.

```
function strokesFromSVG(svg, samples = 12) {
  const box = svg.viewBox.baseVal;
  const paths = [...svg.querySelectorAll('g[id^="kvg:StrokePaths"] path')];

  const strokes = paths.map(path => {
    const length = path.getTotalLength();
    return Array.from({ length: samples }, (_, i) => {
      const p = path.getPointAtLength(length * i / (samples - 1));
      return [(p.x - box.x) / box.width,
              (p.y - box.y) / box.height];
    });
  });

  return strokes.map((points, i) => {
    const previous = strokes[i - 1];
    const move = i === 0 ? [0, 0] : [
      points[0][0] - previous.at(-1)[0],
      points[0][1] - previous.at(-1)[1]
    ];
    return { points, move };
  });
}
```

Listing 2: Converting an SVG into a sequence of sampled strokes and pen-up movements. For SVGs with transforms on stroke groups, the coordinates must first be transformed into a common coordinate system; in the six files used here, the stroke paths carry no transforms.

Listing 3 shows how the distances can be calculated. The function `distance()` compares two two-dimensional points or vectors. The `strokeCost()` function combines the geometric and movement components, while `characterDistance()` finds the least expensive alignment.

The procedure produces a numerical value representing the difference between two characters based on their stroke geometry and pen-up movements. A value of zero indicates that their sampled representations are identical, while higher values indicate greater differences. The comparison is symmetric, meaning that comparing character A with character B produces the same result as comparing B with A. However, the calculated distance does not necessarily reflect how similar two characters appear to a

human reader. It should therefore be understood as a measure of dissimilarity based on the SVG data rather than as an exact measure of visual similarity.

```
function distance(a, b) {
  return Math.hypot(a[0] - b[0], a[1] - b[1]);
}

function strokeCost(a, b) {
  const shape = a.points.reduce((sum, point, i) =>
    sum + distance(point, b.points[i]), 0) / a.points.length;
  const movement = distance(a.move, b.move);
  return 0.75 * shape + 0.25 * movement;
}

function characterDistance(a, b, gap = 0.5) {
  const rows = a.length + 1;
  const cols = b.length + 1;
  const dp = Array.from({ length: rows }, () =>
    Array(cols).fill(0));

  for (let i = 1; i < rows; i++) dp[i][0] = i * gap;
  for (let j = 1; j < cols; j++) dp[0][j] = j * gap;

  for (let i = 1; i < rows; i++) {
    for (let j = 1; j < cols; j++) {
      dp[i][j] = Math.min(
        dp[i - 1][j - 1] + strokeCost(a[i - 1], b[j - 1]),
        dp[i - 1][j] + gap,
        dp[i][j - 1] + gap
      );
    }
  }
  return a.length || b.length
    ? dp[a.length][b.length] / Math.max(a.length, b.length)
    : 0;
}
```

Listing 3: A simple sequence-alignment distance based on stroke geometry and pen-up movements.

3.3 Computation

The distances reported below were calculated using Chromium 153, a web browser, and Playwright, a tool that automates browser operations. Chromium was run in *headless mode*, meaning without displaying a graphical window. This allowed the SVG files to be processed automatically using the browser's built-in functions, `getTotalLength()` and `getPointAtLength()`.

Each SVG file was loaded into an empty web page, where its strokes were extracted and processed using 12 sampled points per stroke. The distances were then calculated

using Listing 3, with weights of 0.75 for stroke geometry and 0.25 for pen-up movements, and a gap cost of 0.5.

4 Results

4.1 Comparison of Similar Kanji

To illustrate how the method works, three pairs of visually similar kanji were selected from KanjiVG. Each pair differs in a particular aspect of its written form. Table 1 reports the number of strokes and the calculated distance for each pair. Lower values indicate greater similarity according to the proposed method.

Chars	Strokes	Distance	Differences
未 / 末	5 / 5	0.032	Different lengths and positions of the horizontal strokes.
木 / 本	4 / 5	0.132	An additional short horizontal stroke in 本.
日 / 目	4 / 5	0.145	An additional horizontal stroke inside 目.

Table 1: Calculated distances between three pairs of visually similar kanji using KanjiVG release r20260714.

The first pair, 未 and 末, has the lowest distance (0.032). Both characters contain five strokes in the same order, but their first two horizontal strokes differ in length and position. The method detects these geometric differences while matching all five strokes. The second pair, 木 and 本, illustrates how characters with different numbers of strokes are compared. The four strokes of 木 can be matched with the first four strokes of 本, leaving only the additional horizontal stroke of 本 unmatched. This produces a distance of 0.132. A similar situation occurs with 日 and 目, which contain four and five strokes respectively. The comparison identifies an additional horizontal stroke inside 目, resulting in a distance of 0.145.

These examples illustrate how the procedure can compare characters with the same or different numbers of strokes, taking into account both their geometry and stroke sequence. The numerical values describe differences in the SVG representations and should not be interpreted as direct measurements of human visual perception.

4.2 All Pairs Among the Six Characters

Table 2 places these values on a common scale by comparing all six characters with each other. Each character's nearest neighbor among the six is its visually similar partner: 未/末, 木/本 and 日/目. An interesting result concerns 木, 本 and 未. Although 本 and

未 both contain five strokes, 木, which has only four, is considered more similar to 未. This happens because the four strokes of 木 appear in the same order in 未, with an additional horizontal stroke near the beginning. In 本, however, the additional horizontal stroke appears at the end. Since the procedure takes stroke order into account, this difference affects the calculated distance.

	未	末	木	本	日	目
未	0	0.032	0.151	0.262	0.361	0.374
末	0.032	0	0.155	0.265	0.358	0.372
木	0.151	0.155	0	0.132	0.327	0.359
本	0.262	0.265	0.132	0	0.312	0.304
日	0.361	0.358	0.327	0.312	0	0.145
目	0.374	0.372	0.359	0.304	0.145	0

Table 2: Distances between all six characters (default settings).

4.3 Contribution of Pen-Up Movements

To understand how pen-up movements affect the calculated distances, the same characters were compared in two different ways. The first calculation considered both stroke geometry and the movements from one stroke to the next, as in the previous comparisons. The second considered only stroke geometry, excluding pen-up movements. Comparing the two sets of results makes it possible to observe how the distances change when these movements are excluded.

Pair	With pen-up movement (0.75 / 0.25)	Geometry only (1.0 / 0)
未 / 末	0.032	0.032
木 / 本	0.132	0.133
日 / 目	0.145	0.138
未 / 日	0.361	0.318
未 / 目	0.374	0.298
本 / 目	0.304	0.251

Table 3: Distances with and without the pen-up movement component.

As shown in Table 3, excluding pen-up movements produces only small differences for the three visually similar pairs. Larger differences appear in some comparisons between structurally different characters, such as 未 and 目, whose distance decreases from 0.374 to 0.298.

These results suggest that pen-up movements can help distinguish characters with different stroke structures. However, within this small group, they do not change which characters are identified as the most similar.

4.4 Influence of the Number of Sampled Points

The calculations were also repeated using different numbers of sampled points per stroke, from 6 to 96. The results changed only slightly. For example, the distance between 未 and 末 was 0.032 with 12 points and 0.031 with 96 points. The relative ordering of the three character pairs remained unchanged, suggesting that 12 points per stroke provide a sufficiently stable approximation for these examples.

5 Conclusion

Unlike raster images, KanjiVG's vector-based SVG files preserve the geometry and order of individual strokes, allowing them to serve as a basis for computational comparisons of kanji writing structure. This article has described a simple procedure that samples stroke geometry, records movements between successive strokes, and applies sequence alignment to obtain a numerical distance between kanji.

Applied to six characters from KanjiVG release r20260714, the procedure assigns each character its visually similar partner as nearest neighbor, and its alignments locate the strokes that distinguish the members of each pair: the first two horizontals of 未 and 末, the final horizontal of 本, and an interior horizontal of 目. The results also show what the representation emphasises. The relative positions of strokes in their writing sequences affect the alignment, helping to explain why 木 is closer to 未 than 本 is. For the selected comparisons, including pen-up movements increases some cross-group distances while leaving the three similar-pair distances almost unchanged.

The procedure is intentionally modest. It uses fixed sampling and manually selected weights, and the pen-up movements are inferred from the canonical stroke paths rather than measured from an actual writer. Six characters are enough to illustrate the method, but not to evaluate it. A natural next step would be to compute distances across the full KanjiVG collection, inspect the nearest neighbors of a larger set of target characters, and compare the resulting rankings with human judgements of graphical similarity, for instance against the data sets used by Yencken and Baldwin (2008) and their stroke-type edit distance.

References

Apel, Ulrich. n.d. *KanjiVG: Kanji Vector Graphics*. Project website and source repository. https://kanjivg.tagaini.net/; https://github.com/KanjiVG/kanjivg (accessed 14 October 2026).

Apel, Ulrich, and Julien Quint. 2004. Building a Graphetic Dictionary for Japanese Kanji: Character Look Up Based on Brush Strokes or Stroke Groups, and the Display of Kanji as Path Data. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*. Geneva, Switzerland.

Wagner, Robert A., and Michael J. Fischer. 1974. The String-to-String Correction Problem. *Journal of the ACM* 21(1): 168–173. https://doi.org/10.1145/321796.321811

Yencken, Lars, and Timothy Baldwin. 2008. Measuring and Predicting Orthographic Associations: Modelling the Similarity of Japanese Kanji. In *Proceedings of the 22nd International Conference on Computational Linguistics (COLING 2008)*, 1041–1048. Manchester, UK. https://aclanthology.org/C08-1131/