ON-LINE CHINESE CHARACTER RECOGNITION

By

Jian-Zhuang Liu

A Thesis

submitted in partial fulfillment of the requirements

for the Degree of Doctor of Philosophy

Division of Electronic Engineering

The Chinese University of Hong Kong

July 1997

UMI Number: 9908127

To my wife and my son, and

to the memory of my parents

Acknowledgement

It is my pleasure to have the opportunity to thank all the people who have given me help during the course of this thesis. First, I would like to express my sincere gratitude to my supervisor, Associate Professor Wai Kuen Cham, for his informal guidance, useful suggestions, and enthusiastic discussions.

I am also very grateful to Professor Wei Xin Xie, the President of Shenzhen University, for introducing me to Professor Cham. He was my advisor and gave me much help in my research and teaching when I was working in Xidian University.

Thanks to Professor Hung Tat Tsui for his kindly and useful suggestions in my current and future research.

Thanks to Mr. Ding Wah Chan for his very good technical support in the laboratory of image processing and computer vision.

Thanks to Professor Chorkin Chan of Hong Kong University, Associate Professor Michael Ming Yuen Chang, and Associate Professor Kin Hong Wong, for serving as the examination committee members of my thesis.

Many thanks to my colleagues, Dr. Yingli Tian, Dr. Yibing Yang, Wai Tat Fu, Fu Wing Tse, Man Ching Auyeung, Hau Lai Ho, Guoliang Fan, Po Ming Wong, and Shu Yuk Yeung, who provided handwritten Chinese characters for testing my on-line Chinese character recognition methods.

I am deeply indebted to my parents who always encouraged me in my studies when they were alive.

Last but not least, I would like to give my special thanks to my wife, Qing Meng, for her patience, constant encouragement and support. She brought up our son in Xian, China while I was studying in Hong Kong.

Part of this research was supported by the Hong Kong RGC Earmarked Research Grant CUHK67/92E.

Abstract

The existing methods and commercial products for on-line Chinese character recognition (OLCCR) are not satisfactory when there are stroke order and stroke number variations. This thesis presents several methods for achieving better performance of OLCCR. We address three aspects: preprocessing of input handwriting, representations of Chinese characters and recognition methods.

First, we deal with the preprocessing problem. To facilitate the recognition of the types of strokes and segments, an input stroke is represented with a polyline. A method for recognizing the types of strokes with more than two segments is proposed by stroke chain code string matching. Some rules are presented to detect most of frequently-occurred connected strokes and then delete the extra segments in such strokes.

Next, we formally define complete relational graphs and distances for measuring the similarity between two graphs. With such graphs, we propose stroke-based and segment-based spatially-temporally relational representations for Chinese characters, using novel "don't care", "should" and "must" relational features.

Recognition methods are the key to OLCCR. We develop three methods in this thesis. The first one is a state space search method. We formulate the graph

matching as a state space search problem. To obtain good search efficiency, we use the A^* algorithm to perform heuristic search, and propose three schemes to speed up the A^*: utilize a heuristic function to make the A^* expand fewer nodes in a search tree; employ a tree pruning operation to let the A^* avoid searching the nodes that have very little chance to be located in the optimal path in a tree; introduce two new criteria, together with the original one, to stop the A^* by using the monotone of the evaluation function of the A^*.

In the two-layer assignment method, finding segment correspondences between two characters is formulated as an assignment problem (in layer 2), which can be solved by the Hungarian method. The cost matrix of this assignment problem is derived by the assignment problems in layer 1. To save computational time, a lower bound estimate and the geometric position features of model characters are used to reduce the complexity of the method from $O(n^5)$ to $O(n^3)$.

The third method is a fast string matching one, which incorporates the geometric position constraints of strokes (or segments) of Chinese characters into Wagner and Fischer's string matching algorithm. To allow more stroke order deviations for some characters, using two or more strings to represent one of these model characters is a feasible way. In this case, we present a scheme to save computational time, by combining two or more separate networks into one and employing a dynamic-programming procedure to solve the shortest path problem.

We also make comparisons of the first method with several other methods published recently, and find that our method is very promising. When segments of Chinese characters serve as primitives, the first two methods are stroke order and number free. The last one is stroke number free and runs much faster but is not stroke order free.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

1.1 Importance of on-Line Chinese Character Recognition

Recently, rapid development of computer techniques has made personal computers (PCs) cheap enough for family use. To enter text into a computer, using a keyboard is faster than handwriting for small-alphabet languages such as English, but it is cumbersome for large-alphabet Chinese. Hundreds of millions who use Chinese in their daily life are bothered all along by the input of Chinese characters into computers, except those who have taken a lot of time to learn by rote some input methods that encode Chinese characters. Therefore, a good on-line Chinese character recognition (OLCCR) system will provide a friendly interface for the use of Chinese and popularize PCs in China and some other areas. In addition to computers, some products, such as portable electronic diaries, electronic Chinese-English dictionaries, multi-functional telephones and simple Chinese typewriters, may also require to be able to recognize on-line inputted Chinese characters.

There are two research fields of handwritten Chinese character recognition: on-line recognition and off-line recognition. In this thesis we only discuss the former study. On-line recognition means that a machine recognizes the input characters while one writes on a digitizer with a stylus pen. Off-line recognition is performed after the handwriting is completed, generally with a scanner converting the image of handwriting on paper into a bit pattern. On-line devices can capture the temporal information of handwriting, such as the number, order and direction change of strokes of a Chinese character. Thus, for recognizing the Chinese characters written in a similar degree of distortion, on-line recognition is easier than off-line recognition. By the way, another advantage of on-line recognition is that a user may participate in the recognition process after the computer selects a small set of possible candidates for an input character.

Recognition of handwritten Chinese characters is considered as a hard problem because of large categories, complex structure, and widely variable and many similar shapes of Chinese characters. Although great progress has been made in OLCCR since the 1970's [36, 88, 97], a number of researchers are still involved in this topic for achieving better performance of OLCCR. From the review in the next section, we can see that the existing methods are not satisfactory. Researchers hope to develop better algorithms which are stroke order and stroke number free, and can run on general computers (e.g. PCs) within an acceptable computational time. Now commercial products for OLCCR are available but their performance still needs improving, because they require that input Chinese characters should be written both in the block (not cursive) style and basically according to their standard stroke orders (in other words, they allow only very few and common stroke order deviations).

1.2 Review of Recent Studies of the Subject

A lot of approaches to OLCCR have been proposed since 1970's, most of which may be classified into one of the techniques: transform, decision tree, string matching, syntactic-semantic analysis, radical (component) decomposition and graph matching [21, 36, 60, 88, 97]. There are so many approaches that we cannot mention them one by one. Thus, in the following we just give the review of recent studies, which in general, have better performance than the methods published earlier. For the reader who wants to know more about this topic, the survey papers [36, 88, 97] are recommended. To evaluate an OLCCR approach, we consider its three aspects: tolerance of stroke order variations, tolerance of stroke number variations, and running time.

In [56], Lin *et al.* proposed a deviation-expansion model to represent Chinese characters, and dynamic programming is used to carry out the character matching. Their approach is stroke-based and in essence a string matching one. It requires that an input character should not have more than one stroke number variation and more than two connected strokes. The running time of their algorithm is $O(2^n)$, where n is the stroke number of an input character. Chou *et al.* [21] extended the above model to a segment-based deviation tree, which is also a string matching one and is not stroke order free. It cannot tolerate more than two stroke order deviations. Their stroke-segment preprocessing scheme makes the approach allow more stroke number deviations. The computational complexity is also $O(2^n)$.

In [17], Chen *et al.* developed a stroke-sequence decision tree to represent Chinese characters and employed stroke positions to calculate the similarity between two characters. The approach is not stroke order and number free. It cannot handle the input characters with stroke number deviations more than one. Tsay and Tsai [92] used attributed string matching by split-and-merge for on-line Chinese character recognition. The proposed method can recognize cursive characters but imposes the constraint of correct stroke orders on them. The authors suggested that their approach could be used to design a writer-dependent system. In [22], Chou and Tsai proposed a discrete iteration scheme to solve the OLCCR problem. The features used to measure the similarity between two characters include lengths, orientations and locations of segments. Their method is not stroke order and number free. The provided test characters are in block style and almost have no connected strokes.

In [19], a hierarchical deformation model is proposed to describe the deformation of on-line cursive Chinese characters. An elastic matching algorithm and a constrained parabola transformation are used to find the correspondences of strokes between two characters. The method requires that input characters should be written in correct stroke orders. The algorithm is very computationally intensive. The time for recognizing a character is 4.2 seconds on a Sun 4 SPARC workstation when there are only 20 model characters.

In [40], Hsieh *et al.* employed a greedy algorithm for bipartite matching to carry out the recognition. The method is segment-based and stroke order free. The provided test characters are neatly written, some of which have one or two connected strokes. Their algorithm needs large amounts of computation. Its running time is $O(\max\{n^5, m^5\})$, where n and m are the segment numbers of two characters under matching. The average time for recognizing a character is 39 seconds on a Sun workstation when there are 452 model characters.

Chen and Lee [16] proposed a fuzzy attribute graph representation for Chinese characters. They used a set of segment intersection features to describe only the relations between segments within the same components. A maximum clique finding algorithm is employed to perform the graph matching. Two problems exist if the maximum clique finding algorithm is used: (1) it is NP-complete and so is time consuming; (2) the thresholds, which are utilized to build an association graph describing all possible compatible mappings between two graphs, may eliminate any possibility of including a given pair of nodes in the final clique, resulting in incorrect recognition [26]. The average time for recognizing a character is 2 seconds on a Sun SPARC-II workstation under the conditions: (1) there are 650 model characters each with a stroke number between 1 and 12, and (2) a preclassification is employed. Obviously, when their model base is enlarged, the algorithm is too slow to use. The method is stroke order free, but its tolerance of stroke number variations is unknown since no test data or characters are provided.

In summary, the above methods are not good enough. Only the last two are not stroke order free, but they require large amounts of computational time. In addition, the method in [40] can only recognize neatly written characters, some of which have one or two connected strokes. The tolerance of stroke number variations in [16] is unknown since no test characters are given. The last method [16] uses graphs to represent Chinese characters. Besides it and our work, there are several off-line recognition methods that also use graphs to represent Chinese characters [13, 18, 68]. Because they are related to part of our work in Chapters 3 and 4, we also give a brief review of them here.

In the application of graph representations and graph matching to Chinese character recognition, a computational problem arises due to the large categories of Chinese characters and the inherent combinatorial explosion of graph matching. In order to save computational time, Chen *et al.* [18] and Lu *et al.* [68] used a two-layer graph to represent a Chinese character. In the first layer, nodes describe components (or radicals) of a Chinese character and arcs describe the relations among these components. In the second layer, each component of the first layer is represented by a graph, in which nodes and arcs represent the strokes and the relations among these strokes of a component, respectively. This strategy results in several smaller graphs for each Chinese character, so matching time can be reduced. However, a new problem of how to correctly group the strokes of a Chinese character into its components arises. The wide handwriting variations and connected strokes make it very difficult to extract components of Chinese characters at a high rate of success. In [18], a relaxation matching algorithm is used to carry out the graph matching but it is still time consuming. Although there are only 24 models (10 numerals and 14 Chinese characters), the method needs 30 seconds to recognize an input character on a PC/AT compatible. In [68], exhaustive search with some search rules are employed to perform the graph matching, but the recognition time required is not reported.

Chan and Cheung [13] used character graphs to represent handwritten Chinese characters and radical graphs to represent model radicals. The recognition of a character is completed from the radicals found by matching radical graphs with the character graph. A NP-complete maximum clique finding algorithm is employed to perform the graph matching, which is expected to be computationally intensive (no recognition time is reported). As we have mentioned above, reliably extracting radicals from the characters with stroke type distortion or connected strokes is a difficult work. Therefore, these three methods are only suitable for recognizing neatly written characters. The test characters provided in [13, 18, 68] support this conclusion.

In Chapter 3, we propose complete relational graph representations of model and input Chinese characters. In a graph, nodes denote primitives (strokes or segments) and their types, and arcs describe the relations between any two primitives. Different from the methods in [13, 18, 68], we do not need to extract the radicals of Chinese characters. Also different from the method in [16], we use the relations between **any** two primitives in our representations, which provide much information that is very beneficial to the graph matching procedures (see Section 4.4.3). The features to describe the relations between primitives in our method are also different from those in [13, 16, 18, 68]. In addition, we use the state space search and the A^* algorithm to carry out the graph matching. With the proposed pruning strategy and stopping rules, our matching method is fast enough for practical application.

1.3 Outline of the Thesis

In the present thesis, we will address three aspects of OLCCR: preprocessing of input handwriting, representations of Chinese characters and recognition methods. The major contributions are shown in Fig. 1.1. The thesis is organized as follows.

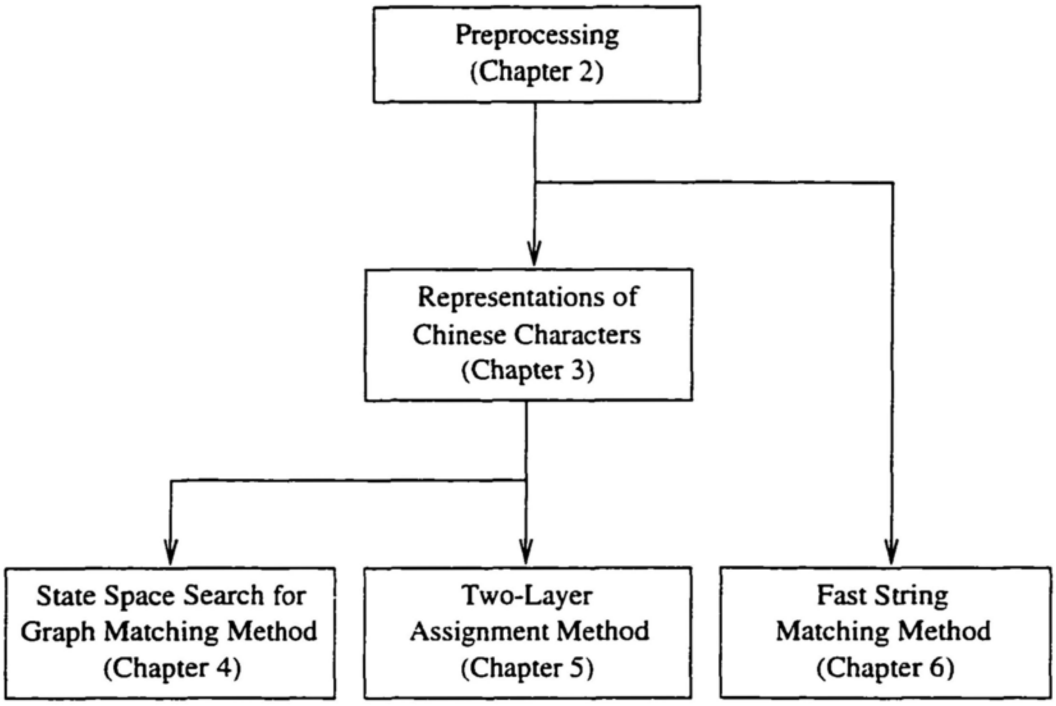


Figure 1.1: The major contributions of the thesis. A line connecting two boxes indicates that there are some relationships between them.

In Chapter 2, we develop several preprocessing approaches to OLCCR. First, we approximate input strokes with polylines by using an efficient splitting and merging algorithm, to facilitate the recognition of strokes and segments. Secondly, we propose a method for identify the types of strokes each with more than two lines, which consists of three procedures: normalization of strokes, extraction of stroke chain code strings, and matching between input code strings and model code strings. The method can be used not only in stroke-based OLCCR but also in segment-based OLCCR. Thirdly, we present some rules to detect frequently-occurred connected strokes and then delete the extra segments in such strokes.

In Chapter 3, we formally define the complete relational graphs and the distances for measuring the similarity between two graphs. With such graphs, we propose several relational representations for OLCCR. The representations incorporate the human knowledge of Chinese characters and can reflect their features well (except some very similar character pairs). The novel "don't care", "should" and "must" relational features allow us to represent unstable, stable and very stable primitive relations conveniently. We also deal with assigning costs to node and arc correspondences for calculating the graph matching distances.

In Chapter 4, we formulate the graph matching as a state space search problem. The optimal matching between two graphs is equivalent to finding the best goal node in a search tree. To obtain good search efficiency, we use the A^* algorithm to perform heuristic search and propose three schemes to speed up the A^*: (1) a heuristic function is defined to make the A^* expand fewer nodes in a search tree; (2) a tree pruning strategy, which employs the geometric position features of strokes (or segments) of Chinese characters to prune a search tree, is presented to let the A^* avoid searching the nodes that have very little chance to be located in the optimal path from the initial node to the best goal node in a tree; (3) two new criteria are proposed to stop the A^* by utilizing the monotone of the evaluation function of the A^*. To demonstrate the performance of the method, we also give the experimental results and make some comparisons between our method and other studies published recently.

In Chapter 5, we propose a two-layer assignment method for OLCCR. Finding segment correspondences between two characters is formulated as a weighted bipartite graph minimum cost complete matching problem, which corresponds to an assignment problem (in layer 2) and can be solved by the Hungarian method.

The cost matrix of this assignment problem is derived by the assignment problems in layer 1. To save the computational time, a lower bound estimate and the geometric position features of model characters are used to reduce the complexity of the method from $O(n^5)$ to $O(n^3)$, where $n = \max\{n_1, n_2\}$ and n_1 and n_2 are two segment numbers of two characters under matching. We also present some experimental results to show the performance of the method.

In Chapter 6, we propose a fast string matching method, which incorporates the geometric position constraints of primitives of Chinese characters into Wagner and Fischer's string matching algorithm. Some experimental results are given. In order to allow more stroke order deviations for some characters, we suggest to use two or more strings to represent one of these model characters, and present a scheme to save computational time. It combines two or more separate networks into one and employs a dynamic-programming procedure to solve the shortest path problem.

Finally, in Chapter 7, we summarize the contributions of the thesis and discuss the directions for future research.

Chapter 2

Preprocessing

2.1 Introduction

A pattern recognition method, in general, requires a set of features of objects to represent and recognize the objects. The procedures to obtain the features prior the classification phase are called preprocessing. For on-line Chinese character recognition, some of these features — stroke (segment) numbers, orders, coordinates, lengths and directions, radical relations, stroke (segment) relations, and so on — are employed to do the recognition job.

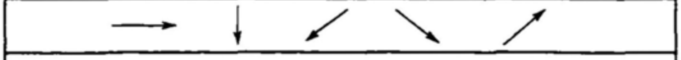
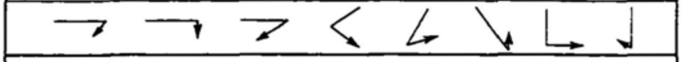
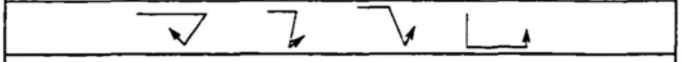
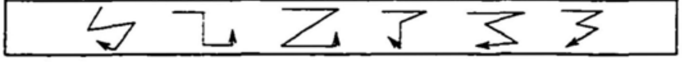
A tablet digitizer can capture on-line input data while a user writes on it with a stylus pen. These data may contain different types of noise, arising from the limited accuracy of the tablet, digitizing process, erratic hand motion, etc. Therefore, a few common techniques have been used to reduce the noise [88]. **Smoothing** averages a point with its neighbors [3, 4, 42, 43, 87, 102]. **Filtering** is utilized to eliminate duplicate points and to reduce the number of points [3, 4, 37, 42, 52]. **Wild point correction** can replace or eliminate occasional spurious points [37, 72, 87]. **Dehooking** eliminates hooks that occur at the beginning and the end of strokes [66, 87]. In addition to these, **normalization** that adjusts the character size to a standard is required in many methods [3, 4, 16, 40, 52, 72, 94, 95, 97].

Not all these preprocessing approaches are necessary for a recognition system, and some new preprocessing techniques may be more suitable for different methods. In this thesis, we propose two graph representations of Chinese characters. The former is stroke-based, i.e., the primitives are strokes, and the latter is segment-based, i.e., the primitives are segments. The nodes of a graph denote the strokes (or segments) of a character, and the arcs indicate the relations between any two strokes (or segments) of the character. Before recognition phase, we have to first extract the strokes (or segments) of an input Chinese character as efficiently and reliably as possible, and then construct its graph representation.

A stroke is defined as the writing from pen down to pen up when one writes on a digitizer with a stylus pen. A Chinese character consists of a set of standard strokes, and each standard stroke consists of from one to four segments, as shown in Table 2.1.[1] Segments are the smallest units that compose a Chinese character. On-line devices can capture the temporal information of the writing, such as the number, order and direction change of strokes. To conveniently identify strokes and segments of an input character, we represent each stroke with straight lines. A line splitting and merging method for reaching this goal is presented in Section 2.2. The stroke type recognition method is proposed in Section 2.3. Section 2.4 gives some schemes for obtaining the segments that

[1]The standard stroke " ㇁ " is not given in the table. It is difficult to define the segment number of this stroke. However, since it is very similar to the standard stroke " 亅 " in handwriting, we consider they belong to the same type in stroke identification.

Table 2.1: Standard strokes.

One segment strokes

Two segment strokes

Three segment strokes

Four segment strokes


are used to represent Chinese characters. This chapter is concluded with the summary in Section 2.5.

2.2 Stroke Approximation with Polylines

A polyline is a concatenation of straight lines and can be used to approximate an object's stroke boundary in computer vision [6]. Here we use it to represent an input stroke. A polyline can fit a stroke to any desired degree of accuracy. The problem is how to find corners or breakpoints that yield a polyline we desire. Fig. 2.1 shows three input strokes that have one, two and four segments, respectively. The arrow on each stroke indicates the direction of the stroke writing. We hope to obtain the results of one line representing stroke (a), two lines representing stroke (b) and four lines representing stroke (c).

A two-step approach is proposed to approximate a stroke with a polyline. Step 1, called a line splitting procedure, uses the iterated endpoint fit method [27] to recursively find a polyline fitting of a stroke. Step 2 is a line merging

(a) (b) (c)

Figure 2.1: Three input strokes.

procedure that merges some connected lines according to a rule.

To explain step 1 clearly, consider an input stroke shown in Fig. 2.2(a). The initial polyline is a line between the first and the end points of the stroke, marked by A and B (Fig. 2.2(b)). Suppose the point in the stroke that is farthest from the line is C. If the distance from C to the line is above a predefined threshold, then the line AB is split into two lines AC and CB (Fig. 2.2(c)). This procedure is recursively applied to lines and the points of the stroke. Note that these points are now partitioned into two groups corresponding to the two lines. A point D in the first group that is farthest from its corresponding line AC is found, and the line will be split again if the point is too far from the line (Figs. 2.2(c) and (d)). The procedure terminates when the distances, from all points of the stroke to their corresponding lines of the polyline, are all below the threshold. Fig. 2.2(f) shows the final polyline for the fitting of the stroke. In the following, we give an algorithm for the implementation of this recursive procedure.

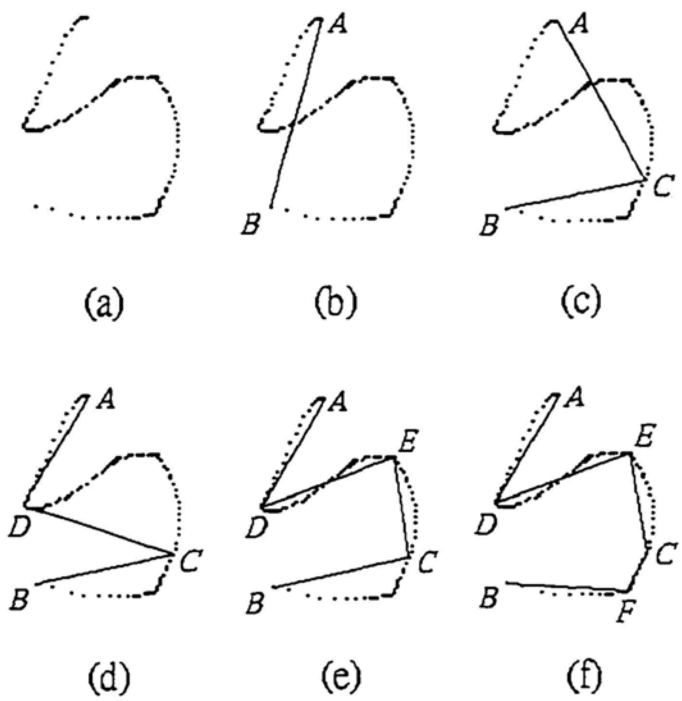


Figure 2.2: Recursive procedure of stroke fitting.

Polyline fitting algorithm

Input: An n-point stroke, represented by two arrays[2] $x[n]$ and $y[n]$, where $(x[l], y[l])$ is the 2D coordinate of the lth point of the stroke, $0 \le l \le n-1$.

Output: An m-line polyline, represented by an array $key\_p[m+1]$, where $key\_p[r]$ and $key\_p[r+1]$ denote the beginning and the end points of the $(r+1)$-th line of the polyline, $0 \le r \le m-1$, $key\_p[0] = 0 < key\_p[1] < \cdots < key\_p[m] = n-1$.

begin

[2]Throughout this thesis, if an array $x[n]$ is given, then it is meant that the size of the array is n and the index of the array is from 0 to $n-1$.

$line\_num := 1$;

$key\_p[0] := 0$;

$key\_p[1] := n - 1$; (comment: the initial line of the polyline)

for $i = 0, 1, ..., line\_num - 1$ **do**

begin

Loop: in the point set $\{(x[key\_p[i]], y[key\_p[i]]), (x[key\_p[i] + 1], y[key\_p[i] + 1]), \cdots, (x[key\_p[i + 1]], y[key\_p[i + 1]])$, find a point $(x[max\_p], y[max\_p])$ that is farthest from a line with two endpoints $\{(x[key\_p[i]], y[key\_p[i]])$ and $\{(x[key\_p[i + 1]], y[key\_p[i + 1]])$;

if the distance of the point from the line is larger than a predefined threshold T_d **then**

begin

$line\_num := line\_num + 1$;

for $j = line\_num, line\_num - 1, ..., i + 2$

do $key\_p[j] := key\_p[j - 1]$;

(comment: rearrange the lines of the polyline)

$key\_p[i + 1] := max\_p$;

go to Loop;

end

end

end

Algorithm Effectiveness. The main effort of the algorithm is the calculation of distances from the points of a stroke to their corresponding lines of the polyline in each iteration. Let n be the number of points of the stroke. If the last resulting polyline has l lines, then the algorithm will terminates after l iterations. In each iteration, the number of points to be visited is at most n. Thus the upper bound on the computational time of the algorithm is $O(ln)$. In general, l is much less than n (less than 10 in stroke fitting), so the algorithm is very efficient.

For an input stroke, the value of the threshold T_d in the algorithm determines the number of lines of the resulting polyline. The smaller the threshold is, the more lines the algorithm yields. It is desired that a stroke be fit by a polyline just as what we want. For example, we consider the stroke shown in Fig. 2.2(a) is a 4-segment stroke and wish the algorithm had yielded a 4-line polyline. If T_d is larger, we can obtain such a polyline. However, too large threshold may make the algorithm ignore some segments of a stroke when it is written on a small area. In order to obtain more desirable polylines, we employ a simple and fast line merging procedure after the stroke fitting.

Line merging algorithm

Input: A polyline.

Output: A modified polyline.

begin

calculate the angles of lines of the polyline;

while the change between angles of two connected lines of the polyline is less than a predefined threshold T_a **do**

begin

merge the two lines into one;

calculate the angle of the new line;

end

end

The thresholds T_d and T_a in the above two algorithms are determined by experiments. In our application, an input character is written on a 6cm$\times$6cm area of the digitizer and is normalized on a 100-pixel$\times$100-pixel image. T_d and T_a are chosen as 6 pixels and 50 degree, respectively.

Some stroke approximations by polylines are presented in Fig. 2.3, in which columns (a) and (d) are input strokes, columns (b) and (e) give their fitting results by the polyline fitting algorithm (step 1), and columns (c) and (f) are the polylines after the line merging processing (step 2) of the polylines in (b) and (e). From the examples, we can see that in most cases, step 2 does not change the results of step 1, while for the strokes in rows 4–8 of column (a), step 2 obtains improved polylines. In practical handwriting, erratic hand motion is easy to generate some wild points and hooks at the beginning or end of strokes such as those in rows 1–3 of Fig. 2.3(a). The polyline fitting algorithm can handle these kinds of noise.

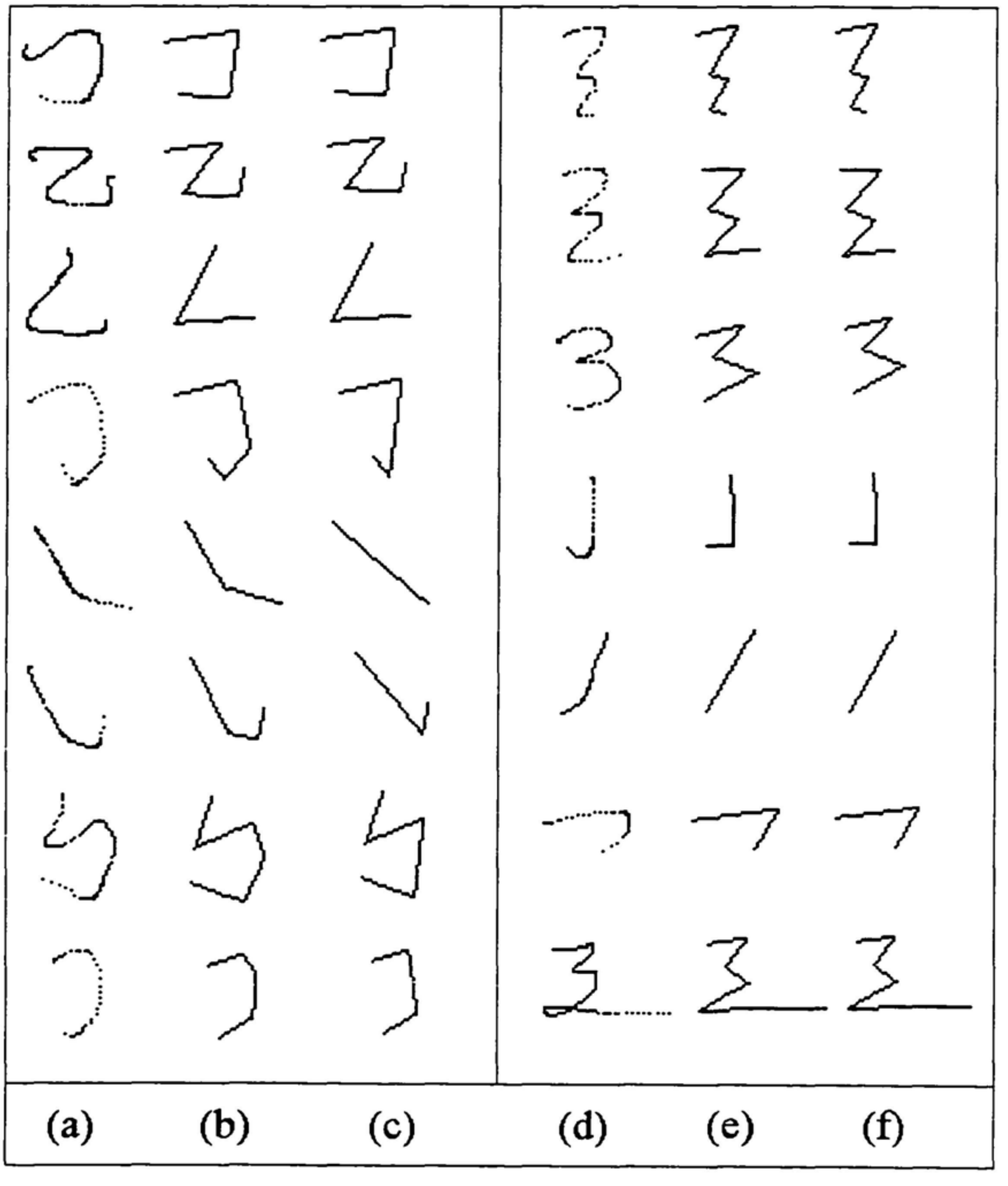
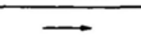
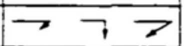
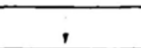
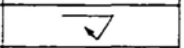
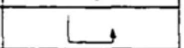
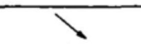
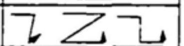
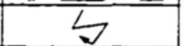
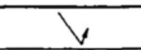
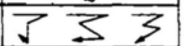
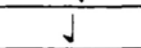
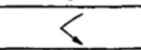
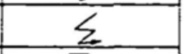
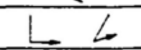
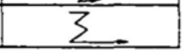


Figure 2.3: Stroke approximation. (a), (d) Input strokes. (b), (e) Polylines after fitting the input strokes in step 1. (c), (f) Polylines after the merging processing of the polylines in (b) and (e).

Table 2.2: 18 model strokes.

Type	Strokes	Type	Strokes
1		10	
2		11	
3		12	
4		13	
5		14	
6		15	
7		16	
8		17	
9		18	

2.3 Stroke Type Recognition

Strokes with different shapes provide very useful information for us to distinguish a character from the others. In the proposed stroke-based Chinese character recognition methods, the primitives are strokes. Therefore, recognition of types of input strokes is one of the important steps. The approximation of input strokes by polylines benefits the stroke recognition task.

The standard strokes 1–15 and three connected strokes occurring often are listed in Table 2.2. They are called model strokes. A connected stroke is a stroke that concatenates two or more standard strokes. We group some strokes together since they are similar to one another. Strokes of types 1–5 appear most frequently in Chinese characters. By analyzing these strokes in Chinese character handwriting, we define the writing angle intervals of $(-20°, 30°]$, $(250°, 290°]$, $(180°, 250°]$, $(290°, 340°]$, and $(30°, 75°]$, as shown in Fig. 2.4, for strokes 1–5, respectively.

If the polyline approximation of an input stroke has one or two lines, we can

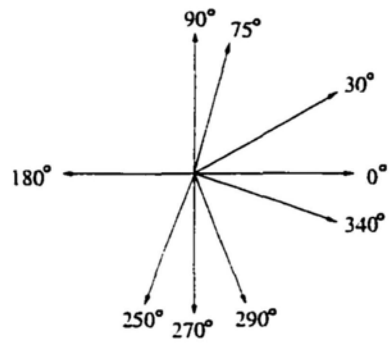


Figure 2.4: Angle intervals for strokes 1–5.

identify it easily by comparing the direction changes of the lines with those of the standard strokes having one or two segments. However, for a polyline with more than two lines, its stroke type recognition becomes complicated because (1) wide variations exist in handwriting, and (2) it is impossible that the polyline approximation of a stroke always produces a result we desire. Look at the two input strokes shown in Fig. 2.5(a). Comparing their fitting polylines in

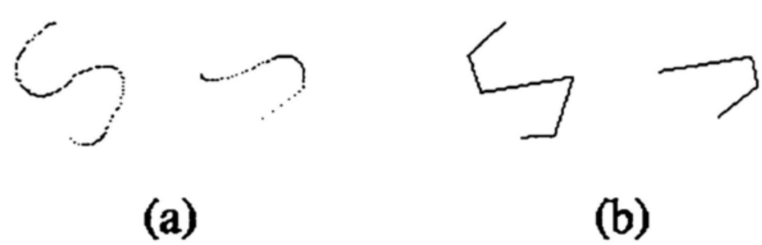


Figure 2.5: (a) Two input strokes. (b) Corresponding polylines.

Fig. 2.5(b) with their corresponding model strokes in Table 2.2, we find the differences between the line numbers of the polylines and the segment numbers of the model strokes. In order to identify these kinds of input strokes reliably, we propose a stroke recognition approach using chain code string matching in the following.

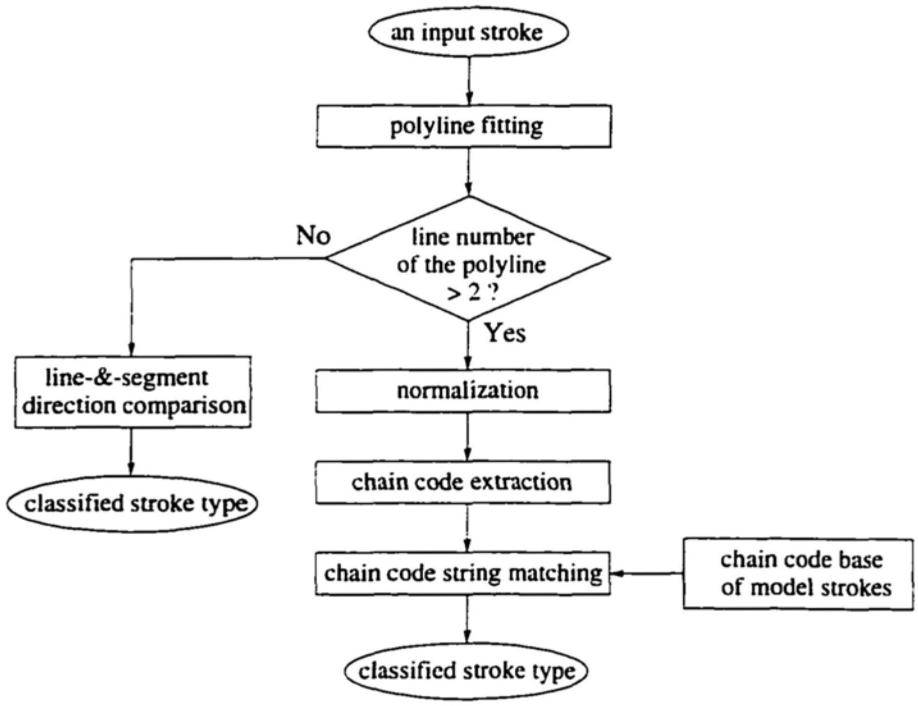


Figure 2.6: Stroke type recognition structure.

2.3.1 Normalization and Chain code extraction

The structure of stroke type recognition is shown in Fig. 2.6. For an input stroke approximated by a polyline with one or two lines, its type identification is an easy task as mentioned above, so we only consider the recognition of a stroke fit by a polyline with more than two lines. Strokes belonging to the same type may be written as having significantly different sizes. Thus normalization of polylines of strokes before recognizing them is reasonable.

Let $(x_{\max}, y_{\max})$ and $(x_{\min}, y_{\min})$ be the upper-right and the lower-left corner points of the smallest rectangle that surrounds a polyline. Let (x, y) be one of the vertexes that represent the polyline. The equations

$$x_l = \frac{100}{x_{\max} - x_{\min}}(x - x_{\min}) \tag{2.1}$$

$$y_l = \frac{100}{y_{\max} - y_{\min}}(y - y_{\min}) \tag{2.2}$$

will transform (x, y) to a new vertex (x_l, y_l) of the corresponding normalized polyline that is located in a square of size 100-by-100. Fig. 2.7 gives an example of normalization.

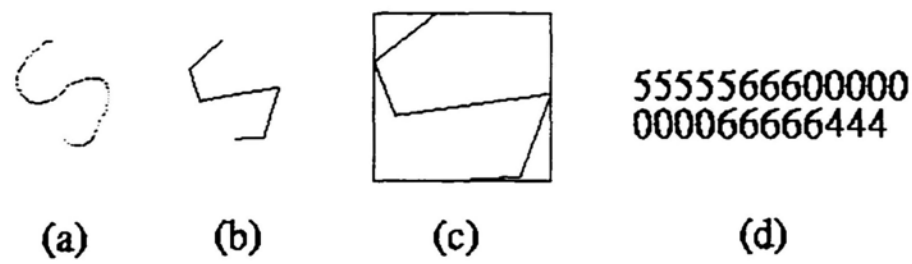


Figure 2.7: (a) Input stroke. (b) Polyline of (a). (c) Normalized polyline of (b). (d) Chain code string of (c).

A polyline consists of several lines, and the directions and lengths of the lines represent the feature of the polyline. If we divide each line into a set of shorter lines (called line units), each with the same length, then chain codes become a very suitable tool for the representation of a polyline.

Chain codes, in our application, are a notation for recording a string of line units along a polyline. A code specifies the direction of a line unit. There are eight quantized directions as shown in Fig. 2.8. Starting at the first line unit and ending at the last line unit of a (normalized) polyline, a string of chain codes is not difficult to obtain by investigating the direction and length of its every line. Fig. 2.7(d) presents such an example.

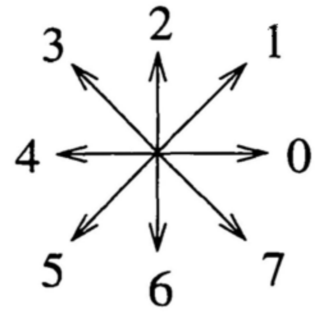


Figure 2.8: 8 directions of chain codes.

The chain code base contains a set of chain code strings of the model strokes having two or more segments. The procedure of constructing the base is just like that of finding the chain code string of an input stroke to be recognized, including (1) fitting strokes with polylines, (2) normalizing the polylines, and (3) obtaining the chain code strings from the polylines.

2.3.2 Chain Code String Matching

A critical step in stroke type recognition is to measure the similarity between two strokes. Now we represent normalized strokes with chain code strings. The string edit operations and the string edit distance proposed by Wagner and Fisher [96] can be used to reach this goal. Here we introduce only the basic concepts and the string matching algorithm following the work of Wagner and Fisher. More detailed description will be given in Section 6.2.

Let λ denote a **null** chain code. An edit operation is a pair (a, b) written as $a \rightarrow b$, where a or b may be a code of a string but if $a \neq \lambda$ and $b \neq \lambda$, both a and b must be two codes. The three edit operations on a string are code substitution, code insertion and code deletion, denoted by $a \rightarrow b$, $\lambda \rightarrow a$ and $a \rightarrow \lambda$, respectively. Obviously, there are infinite sequences of edit operations that can transform a string to another string. Let γ be a cost function that assigns to each edit operation $a \to b$ a nonnegative real number $\gamma(a \to b)$, and let the cost of a sequence of edit operations be the sum of all the edit operation costs. Then the edit distance $\delta(S_1, S_2)$ between two strings S_1 and S_2 is defined as the minimum cost of a sequence of edit operations that transforms S_1 to S_2.

Wager and Fisher provided the following efficient algorithm with the complexity $O(mn)$ for computing the distance between a string of length m and a string of length n.

String matching algorithm [96]

Input: String $S_1 = s_1 s_2 ... s_m$ and string $S_2 = s'_1 s'_2 ... s'_n$.

Output: Distance $D[m,n]$ between S_1 and S_2.

begin

$D[0,0] := 0;$

for $i = 1, 2, ..., m$ **do** $D[i,0] := D[i-1,0] + \gamma(s_i \to \lambda);$

for $j = 1, 2, ..., n$ **do** $D[0,j] := D[0,j-1] + \gamma(\lambda \to s'_j);$

for $i = 1, 2, ..., m$ **do**

for $j = 1, 2, ..., n$ **do**

begin

$d_1 := D[i-1, j-1] + \gamma(s_i \to s'_j);$

$d_2 := D[i-1, j] + \gamma(s_i \to \lambda);$

$d_3 := D[i, j-1] + \gamma(\lambda \to s'_j);$

$D[i,j] := \min\{d_1, d_2, d_3\};$

end

end

Wager and Fisher stated that after the algorithm terminates, $\delta(S_1, S_2) = D[m,n]$ if the cost function γ is a metric, i.e., γ fulfills

(a) $\gamma(a \to b) \geq 0$ (positivity);

(b) $\gamma(a \to b) = 0$ if and only if $a = b$ (definiteness);

(c) $\gamma(a \to b) = \gamma(b \to a)$ (symmetry);

(d) $\gamma(a \to b) + \gamma(b \to c) \geq \gamma(a \to c)$ (triangle inequality).

For our chain code string matching problem, we also have to choose reasonable cost values with respect to different edit operations. Let $s_i, s_j \in \{0, 1, ..., 7\}$ be two chain codes. We define

$$\gamma(s_i \to s_j) = \gamma(s_j \to s_i) = \min\{k_1|s_i - s_j|, k_1(8 - |s_i - s_j|)\} \tag{2.3}$$

as code substitution costs and

$$\gamma(s_i \to \lambda) = \gamma(\lambda \to s_i) = k_2 \tag{2.4}$$

as code deletion and code insertion costs, where k_1 and k_2 are two positive values.

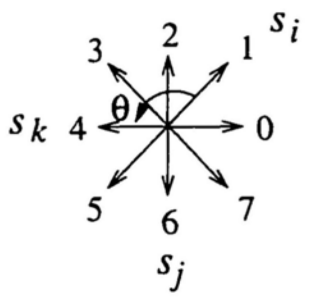
Theorem 2.1 *The cost function γ defined in (2.3) and (2.4) is a metric if $k_2 \geq 2k_1$.*

Proof. It is obvious that γ fulfills the conditions of positivity, definiteness and symmetry. To proof the triangle inequality

$$\gamma(s_i \to s_j) + \gamma(s_j \to s_k) \geq \gamma(s_i \to s_k),$$

let us consider two cases.

Case 1. Suppose $s_i, s_j, s_k \in \{0, 1, ..., 7\}$. From (2.3) and the figure below,



we see that $\gamma(s_i \to s_k)$ is directly proportional to the angle $\theta \leq 180°$ from s_i to s_k. If s_j is located in this interval, then the angle θ_1 from s_i to s_j and the angle θ_2 from s_j to s_k satisfy the relation $\theta_1 + \theta_2 = \theta$, and thus

$$\gamma(s_i \to s_j) + \gamma(s_j \to s_k) = \gamma(s_i \to s_k).$$

Otherwise, $\theta_1 + \theta_2 > \theta$, and

$$\gamma(s_i \to s_j) + \gamma(s_j \to s_k) > \gamma(s_i \to s_k).$$

Case 2. Recall that $a \to b \neq \lambda \to \lambda$. If $s_i = \lambda$, then $s_j, s_k \in \{0, 1, ..., 7\}$. We have

$$\gamma(\lambda \to s_j) + \gamma(s_j \to s_k) = k_2 + \gamma(s_j \to s_k) \geq k_2 = \gamma(\lambda \to s_k).$$

If $s_k = \lambda$, we also have

$$\gamma(s_i \to s_j) + \gamma(s_j \to \lambda) = \gamma(s_i \to s_j) + k_2 \geq k_2 = \gamma(s_i \to \lambda).$$

If $s_j = \lambda$, since $\gamma(s_i \to s_k) \leq 4k_1$ (see (2.3)), we further have

$$\gamma(s_i \to \lambda) + \gamma(\lambda \to s_k) = 2k_2 \geq 4k_1 \geq \gamma(s_i \to s_k).$$

Therefore the triangle inequality holds. □

Let $P_1, P_2, ..., P_q$ be q model strings and S be an input string to be classified. The solution to the problem of recognizing S is first to find a model string $P_k \in \{P_1, P_2, ..., P_q\}$ such that

$$\delta(P_k, S) = \min\{\delta(P_1, S), \delta(P_2, S), ..., \delta(P_q, S)\},$$

and then to classify S into the class which P_k belongs to if $\delta(P_k, S)$ is less than a predefined threshold; otherwise to classify it into an unknown-stroke-type class. Note that the number of model strings may be greater than the number of string classes. We use three to stand for one stroke class in our recognition scheme. This is beneficial to tolerating more handwriting variations.

Before applying the string matching algorithm to Chinese character recognition, we have to determine the two parameters k_1 in (2.3) and k_2 in (2.4). In our learning procedure, for each class, 20 strokes written by 4 people were collected as the training data. The prototypes of a class consist of three strokes. One was written in its standard style and the other two are its generally handwritten deformed versions. The aim of the learning procedure is to find the optimal parameters k_1 and k_2 that maximize the following recognition rate

$$R(k_1, k_2) = \frac{\text{The number of strokes classified correctly}}{\text{The total number of the training strokes}}. \tag{2.5}$$

Fig. 2.9(a) shows the relation between R and (k_1, k_2). From another viewpoint we can obtain the projected surface on the k_1-R plane (Fig. 2.9(b)). It is

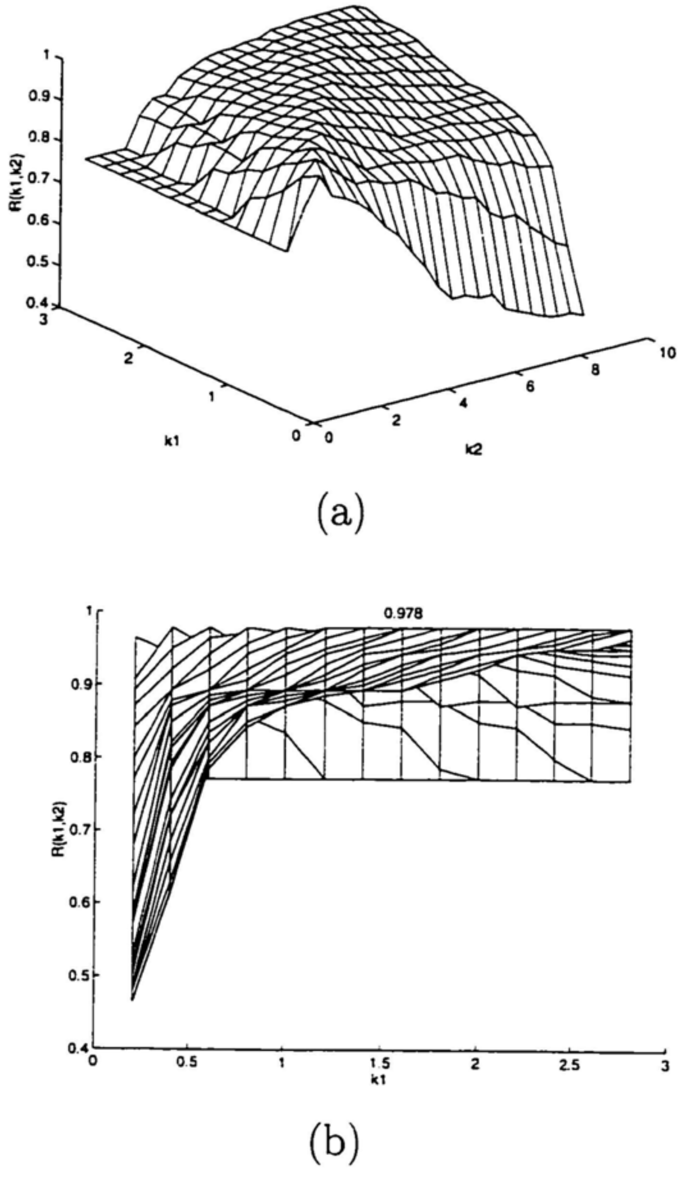


Figure 2.9: (a) Recognition rate R as a function of (k_1, k_2). (b) The surface obtaining from another viewpoint.

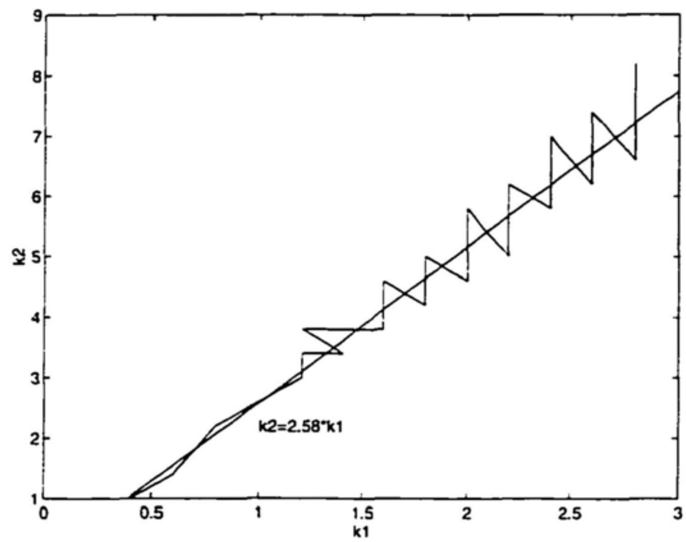


Figure 2.10: A curve and its fitting line. The curve presents the track where $R(k_1, k_2)$ takes the maximum value.

clear that there is an area where $R(k_1, k_2)$ takes the maximal value. The curve in Fig. 2.10 is generated by connecting the discrete points (k_1', k_2')'s satisfying $R(k_1', k_2') = \max\{R(k_1, k_2)\}$. Approximating the curve with a straight line we have

$$k_2 = 2.58k_1. \tag{2.6}$$

The figure indicates that a point (k_1, k_2) that fulfills $k_1 > 1.5$ and $k_2 \approx 2.58k_1$ has more neighbors (k_1'', k_2'')'s, where $R(k_1'', k_2'') = \max\{R(k_1, k_2)\}$. Therefore we choose $k_1 = 2$ and $k_2 = 5.16$ as the parameters of the string matching algorithm.

Some experiments have been carried out to test the performance of the proposed stroke recognition approach. The test data contain more than 1000 strokes written by 5 people. Fig. 2.11 shows some of the input strokes and the classification results, together with their corresponding stroke types. There are a variety

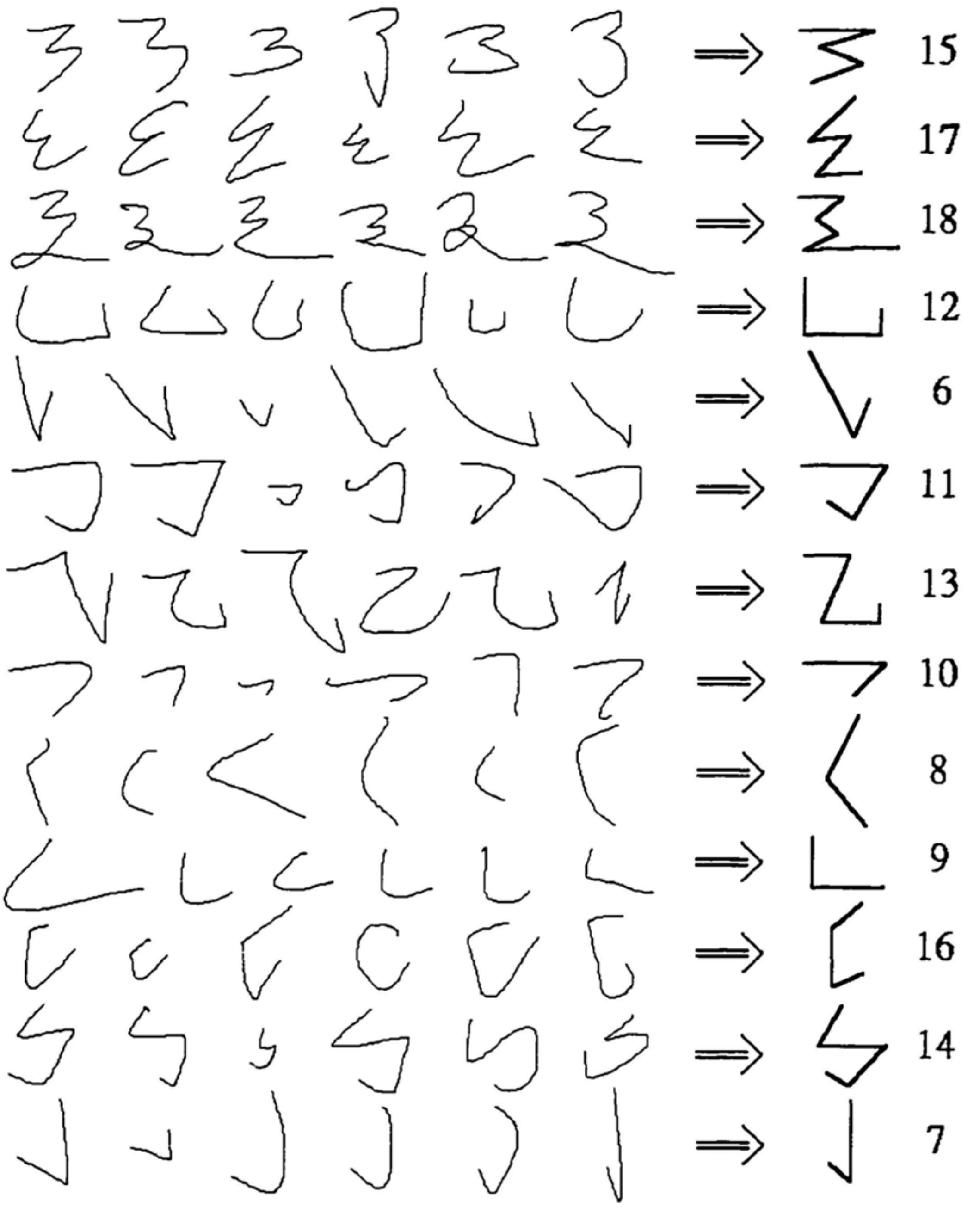


Figure 2.11: Stroke recognition experiments. On the left of the arrows are a set of input strokes. Their classification results: corresponding standard strokes and stroke types, are indicated on the right.

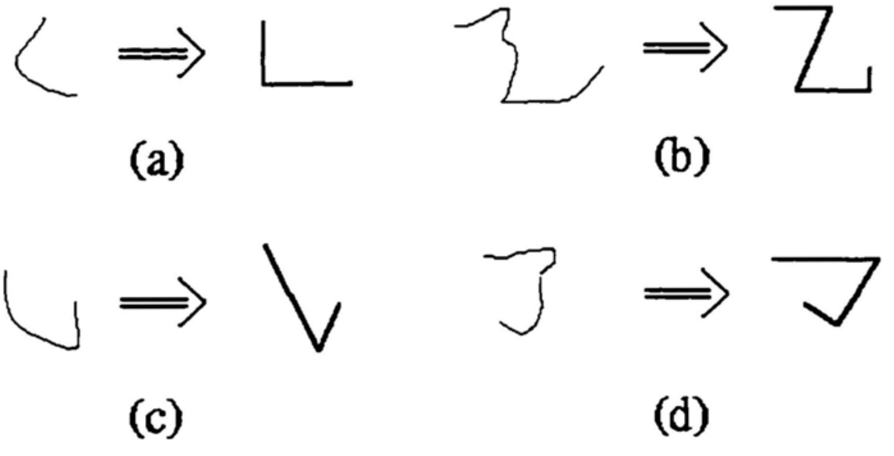


Figure 2.12: Four examples of misclassification.

of handwriting stroke sizes and styles in the data. Our approach achieves a correct recognition rate of 96.2%. Fig. 2.12 gives four strokes recognized incorrectly in the sense that the classified stroke type of an input stroke is different from the type that the subject expects the stroke should be. However, the input stroke in Fig. 2.12(a) is similar to both type 8 stroke and type 9 stroke, and the input stroke in Fig. 2.12(c) is similar to both type 6 stroke and type 12 stroke.

The stroke recognition experiments also show what pairs of strokes are easily confused. This information is very useful for the design of stroke-based Chinese character recognition methods in which assigning different costs for stroke type comparisons is necessary. By the way, it is not definite that misclassification of some strokes of a character leads to incorrect recognition of the character.

2.4 Segment Extraction and Processing

In out segment-based Chinese character recognition methods, the primitives are segments of strokes. Segments are the smallest units that construct Chinese characters. Each standard stroke consists of one to four segments. (see Table 2.1) A connected stroke may have more than four segments.

Six segment types are defined as the primitives: type 1 " $\rightarrow$ " (-20°,30°], type 2 " $\downarrow$ " (250°,290°], type 3 " $\swarrow$ " (180°,250°], type 4 " $\searrow$ " (290°,340°], type 5 " $\nearrow$ " (30°,75°] and type 0 denoting an unstable short 1-segment stroke that is easy to be written as one of the segment types 1–4 such as the top stroke of the character " 求 ". Actually, segment types 1–5 are just the same as stroke types 1–5, respectively. After approximating an input stroke by a polyline, if we consider a line of the polyline is equivalent to a segment, obtaining the polyline means that we have finished the segment extraction.

It is important to note that segment " $\nwarrow$ " with directions ranging from 75 ° to 180 ° is not included in the set of segment primitives while they exist in the standard strokes. By analyzing handwritten Chinese characters, we can find such two facts: (1) ignoring this kind of segments in Chinese characters does not make us be confused when we recognize them, and (2) they exist in many connected strokes of handwritten Chinese characters, as shown in Fig. 2.13. Therefore, we delete these segments before the processing described below. This scheme has also been used in several other methods [17, 21, 64].

From the next chapters, we can see that segment-based Chinese character recognition methods can deal with the problem of recognizing Chinese characters with more connected strokes better than stroke-based methods. Connected

Figure 2.13: Several examples where bold segments " ↖ " appear in connected strokes.

strokes often lead to extra segments. In order to facilitate our segment-based recognition, we use some rules to determine which segments of an input character should be employed for later recognition.

Besides the frequently occurring connected strokes led by the segment " ↖ ", there are many connected strokes in natural Chinese character handwriting. Some of them yield extra segments but some do not. Figs. 2.14(a) and (b) show two sets of characters or components of characters corresponding to the former and the latter cases, respectively. The segments of both standard strokes and the connected strokes that have no extra segments should be used to represent Chinese characters. However, some of these strokes, such as " Z " and " ʒ ", also appear in Fig. 2.14(a). Thus a trade-off must be made. We adopt such a scheme that all the segments of these strokes will remain.

To summarize, we give the following rules that are used in the segment preprocessing.

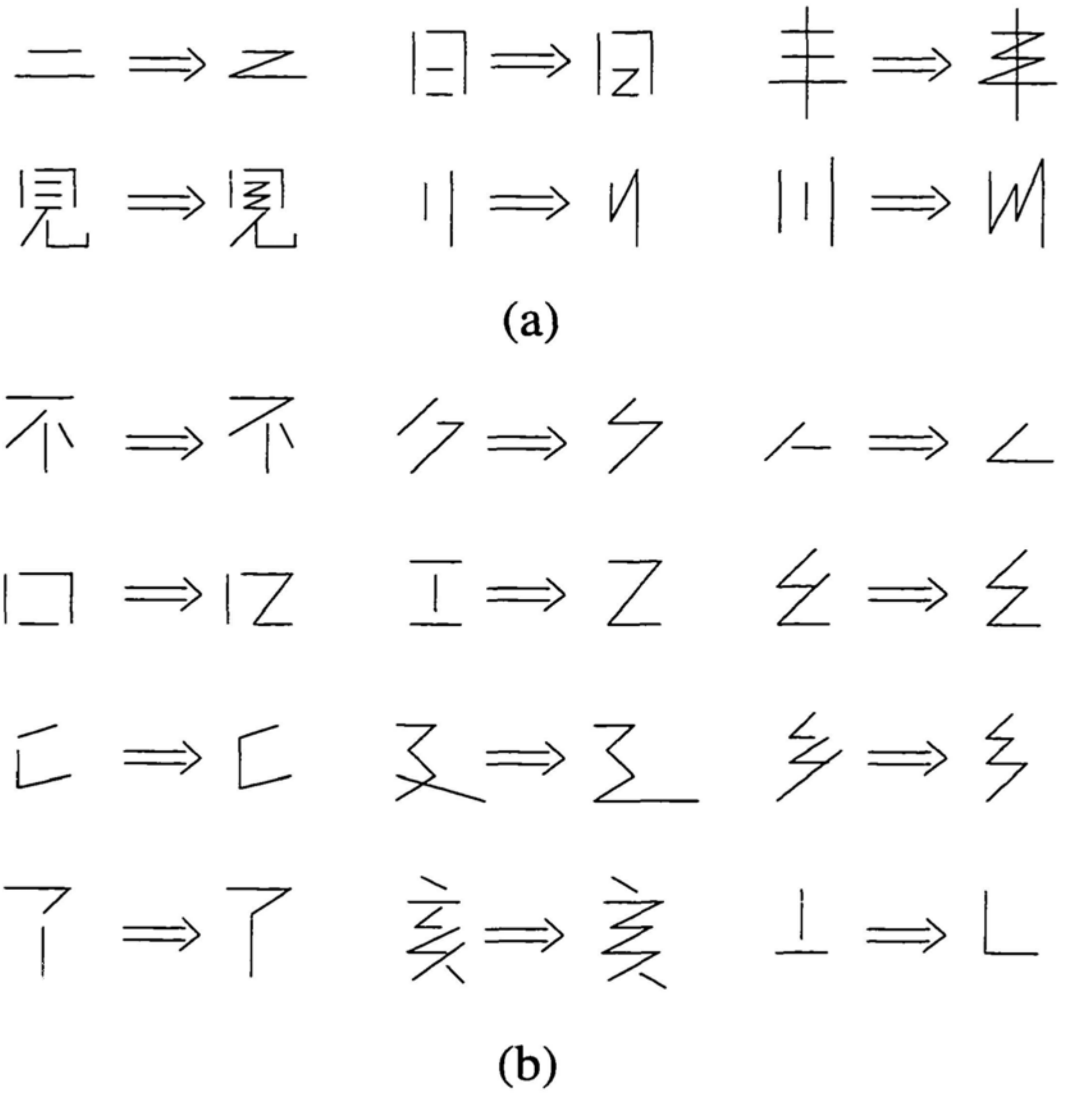
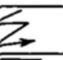
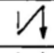
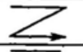
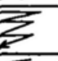
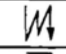
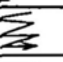
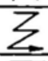
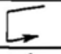
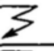
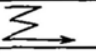


Figure 2.14: (a) Examples of connected strokes leading to extra segments. (b) Examples of connected strokes not leading to extra segments.

Table 2.3: 14 Multi-segment strokes used to determine whether an input stroke with more than two segments belongs to one of them.

	Stroke		Stroke		Stroke
1		6		11	
2		7		12	
3		8		13	
4		9		14	
5		10			

Rule 1. The segments " $\nwarrow$ " existing in all input strokes are deleted.

Rule 2. If the segment number of an input stroke is less than 3, all the segments of the stroke remain.

Rule 3. If the segment number of an input stroke S_i is greater than 2, the stroke recognition method presented in the last section is employed to find the minimum distance $\delta(S_i, S_j) = \min\{\delta(S_i, S_1), \delta(S_i, S_2), ..., \delta(S_i, S_{14})\}$, where S_j, $j = 1, 2, ..., 14$ is one of the strokes shown in Table 2.3.

- If $\delta(S_i, S_j) < T_r$ and $j < 11$, then all the segments of the stroke remain.
- If $\delta(S_i, S_j) < T_r$ and $j = 11$ or 12, then the segments " $\downarrow$ " remain and the others are deleted.
- If $\delta(S_i, S_j) < T_r$ and $j = 13$ or 14, then the segments " $\rightarrow$ " remain and the others are deleted.
- If $\delta(S_i, S_j) \geq T_r$, then Rule 4 is used.

Rule 4. Suppose a stroke (after the processing of deleting segment " $\nwarrow$ ") has m segments. If m is odd, then the 2th, 4th, ... , $(m-1)$-th segments are deleted and the others remain. If m is even, then the 2th, 4th, ... , $(m-2)$-th segments are deleted and the others remain.

Here T_r is a predefined threshold and Rule 4 is borrowed from [21]. Clearly, it is impossible to delete all extra segments or to obtain all the segments that should remain for later character recognition. However, these rules do provide us satisfactory processing results, as illustrated in Fig. 2.15. The model characters having 9 to 11 strokes were written as their input versions having 2 to 6 strokes. It can been seen that the input characters, after processing, are more recognizable. To break the connected strokes at proper positions, Rule 1 contributes the most because lots of connected strokes lead to the segments " $\nwarrow$ ".

Rule 1 is so effective that it, after modification, is also used in the stroke-based representation and recognition of Chinese characters. The modified rule is: delete the segments " $\nwarrow$ " that exist in input strokes and are not the last segments in these strokes.

Finally, we have to state that some segment processing errors of an input character (such as deleting a segment that should remain to represent the character) do not mean that a misclassification of the character must take place. To design robust recognition methods that can tolerate more handwriting variations and preprocessing errors is the aim of the next several chapters.

2.5 Summary

We have introduced several preprocessing approaches to on-line Chinese character recognition in this chapter. First, we have approximated input strokes with

Figure 2.15: Examples of segment processing. Columns (a) and (d) are input Chinese characters. Columns (b) and (e) are the segment processing results. Columns (c) and (f) show the corresponding model characters.

polylines by using the efficient polyline fitting and line merging algorithms, to facilitate the recognition of strokes and segments. Secondly, we have proposed a method for identify strokes each with more than two lines. It consists of three procedures: normalization of strokes, extraction of stroke chain code strings, and matching between input code strings and model code strings. The method works well and can be used not only in stroke-based but also in segment-based on-line recognition of Chinese characters. Thirdly, in the section of segment extraction and processing, some rules are presented to detect most of frequently-occurred connected strokes and then delete the extra segments in such strokes. These rules make our recognition methods have the ability to recognize more freely-written Chinese characters.

Parts of the results presented in this chapter have been published in [58, 59, 60, 61, 62, 63].

Chapter 3

Relational Graph Representations of Chinese Characters

3.1 Introduction

What kinds of features of Chinese characters to be chosen and how to represent Chinese characters using these features are two important issues on Chinese character recognition. Human beings have the best ability in recognizing complicated objects. Development of on-line Chinese character recognition systems that approach the human ability is the goal of researchers working on this field.

Chinese characters are two-dimensional (2D) pictographic characters. In general, the structure of a Chinese character having more than five strokes can be decomposed into four levels as shown in Fig. 3.1. The most basic elements constructing Chinese characters are strokes but the segment level is, in my opinion,

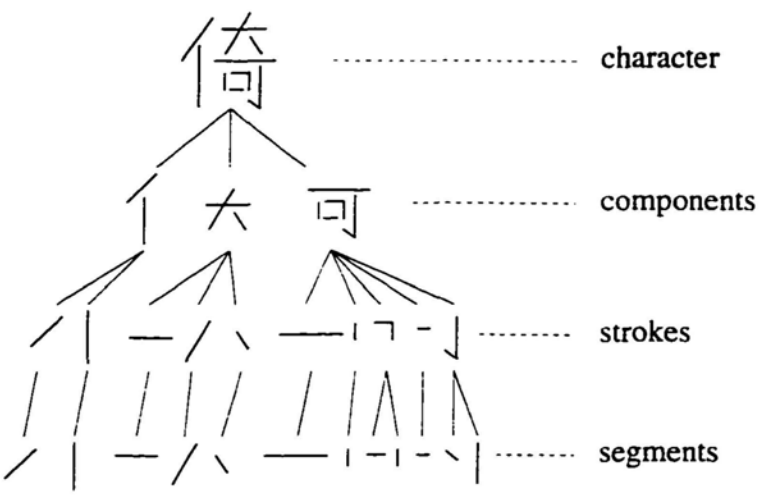


Figure 3.1: 4-layer structure of a Chinese character.

most useful for computer recognition of freely handwritten Chinese characters.

Let us consider how a human being distinguishes a Chinese character from the others. If he/she is familiar with Chinese characters, he/she recognizes a printed or neatly written character by identifying its each component and the 2D arrangement of the components. His/Her ability of quickly finding a component comes from his/her understanding of how the component is formed by the 2D arrangement of some strokes, the smaller elements. If a character is freely written and has several connected strokes, in order to recognize it, he/she also needs the knowledge of general freely handwritten styles of Chinese characters.

Obviously, human beings use 2D relational (structural) features of Chinese characters to conduct the recognition activity. The structural methods that can capture human knowledge of Chinese characters very well should have the best performance. Relational graphs are a powerful tool for the representation of relational structures of a pattern. They have been used for 2D or 3D scene analyses [11, 28, 33, 45, 54, 80, 82, 84, 85, 90, 100, 101] as well as on-line and off-line Chinese character recognition [13, 16, 18, 58, 59, 60, 61, 62, 63, 68].

In Section 3.2, we formally define the complete relational graphs and the distance measures for comparing the similarity between two graphs. Then, we propose several graph representations for on-line Chinese character recognition in Section 3.3, including stroke-based and segment-based spatially relational representations, as well as stroke-based and segment-based spatially-temporally relational representations. The assignments of costs to node and arc correspondences for calculating distances between two graphs are presented in Section 3.4. The chapter ends with the summary.

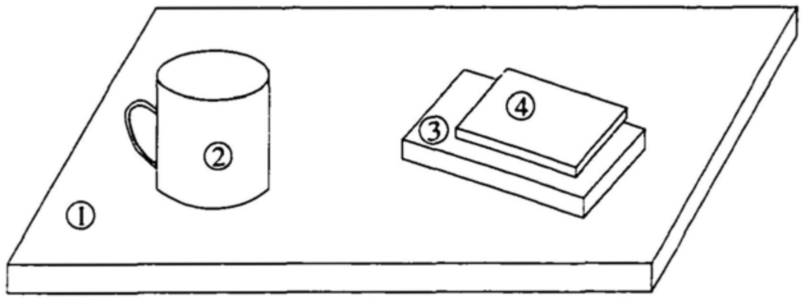
3.2 Relational Graphs and Distance Measures

3.2.1 Complete Relational Graphs

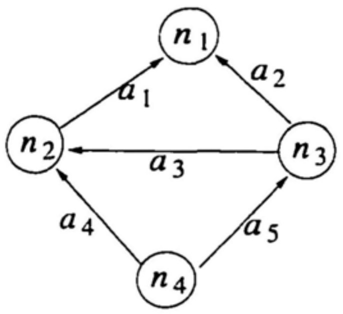
Definition 3.1 *Let V_N be a set of node labels and V_A a set of arc labels. A relational graph over $V = V_N \cup V_A$ is a 4-tuple $G = (N, A, \mu, \varepsilon)$, where*

- *N is a finite nonempty set of nodes;*
- *$A \subset N \times N$ is a set of distinct directed pairs of distinct elements in N called arcs;*
- *$\mu : N \rightarrow V_N$ is a function for labeling the node;*
- *$\varepsilon : A \rightarrow V_A$ is a function for labeling the arcs.*

Relational graphs can be used to describe the structural information of patterns. Fig. 3.2 shows a simple example of the representation of a scene, where nodes indicate object primitives and arcs describe spatial relations between objects. In this thesis, we use **complete** relational graphs to represent Chinese



(a)



V_N = {book, cup, board, clock, ...}
V_A = {above, right of, left of, ...}
Node set = {n_1, n_2, n_3, n_4}
Arc set = {a_1, a_2, a_3, a_4, a_5}
$\mu(n_1)$=board, $\mu(n_2)$=cup,
$\mu(n_3) = \mu(n_4)$=book,
$\varepsilon(a_1) = \varepsilon(a_2) = \varepsilon(a_5)$= above,
$\varepsilon(a_3) = \varepsilon(a_4)$= right of.

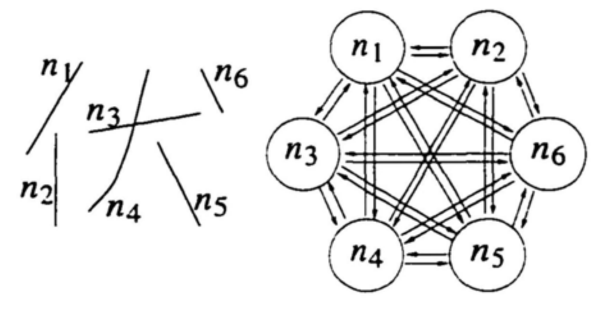
(b)

Figure 3.2: An example of relational graph representation of a scene. (a) A scene. (b) The corresponding relational graph.

characters.

Definition 3.2 *A complete relational graph is a relational graph such that for any two distinct nodes n_1 and n_2, there are two arcs: one from n_1 to n_2 and the other from n_2 to n_1, denoted by (n_1, n_2) and (n_2, n_1), respectively.*

A possible complete relational graph representation of the Chinese character in Fig. 3.3(a) is illustrated in Fig. 3.3(b). In this example, the primitives are strokes of the character and their types are represented by the nodes of the graph;



(a) (b)

	n_1	n_2	n_3	n_4	n_5	n_6
n_1		(1,2,0)	(2,1,0)	(2,1,0)	(1,1,0)	(2,1,0)
n_2	(0,2,0)		(2,1,0)	(2,1,0)	(2,1,0)	(0,1,0)
n_3	(2,0,0)	(2,0,0)		(2,2,1)	(1,2,0)	(2,1,0)
n_4	(2,0,0)	(2,0,0)	(2,2,1)		(2,1,0)	(0,1,0)
n_5	(0,0,0)	(2,0,0)	(0,2,0)	(2,0,0)		(0,2,0)
n_6	(2,0,0)	(1,0,0)	(2,0,0)	(1,0,0)	(1,2,0)	

(c)

Figure 3.3: (a) A Chinese character. (b) Corresponding complete relational graph. (c) Relation matrix of the graph.

the arcs describe the relations between any two nodes (strokes). Three relation types are used: (1) "below" (denoted by "0"), "above" (denoted by "1") or "don't care" (denoted by "2"); (2) "right of" (denoted by "0"), "left of" (denoted by "1"), or "don't care" (denoted by "2"); (3) "uncrossed" (denoted by "0"), "crossed" (denoted by "1"), or "don't care" (denoted by "2"). The relations of an arc (n_i, n_j) are described by a vector $(a_{ij}^1, a_{ij}^2, a_{ij}^3)$, where $a_{ij}^1, a_{ij}^2, a_{ij}^3 \in \{0, 1, 2\}$. The superscripts 1, 2 and 3 on respective a_{ij}^1, a_{ij}^2 and a_{ij}^3 represent the three types of relations. For example, the relations of arc (n_1, n_2) are $(1, 2, 0)$, which suggest that in general handwriting, the geometric center of stroke 1 is always above that of stroke 2; it is sometimes on the left of and sometimes on the right of that of stroke 2; these two strokes are uncrossed each other. All the relations among strokes of the character are shown in Fig. 3.3(c). The reason why we use complete relational graphs to represent Chinese characters and the more general representation will be discussed in Section 3.3.

Since we always deal with complete relational graphs in most parts of this thesis, we will just speak of graphs unless otherwise stated, and we use G or $G = (N, A)$ to denote $G = (N, A, \mu, \varepsilon)$ for short. In addition, the graph in Fig. 3.3(b) may be drawn in its simplified forms of Fig. 3.4(a) or (b).

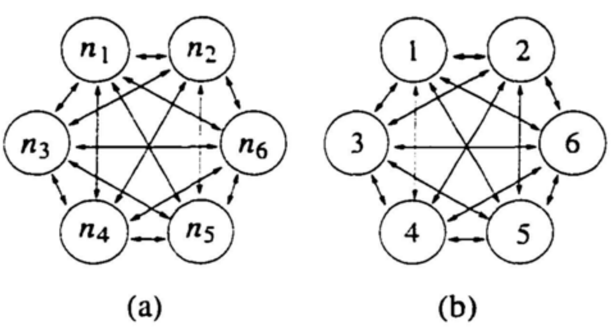


Figure 3.4: Two simplified forms of Fig. 3.3(b).

Definition 3.3 *An induced subgraph $G' = (N', A')$ of $G = (N, A)$ is a graph whose node set $N' \subseteq N$ and whose arc set comprises exactly the arcs of G which join nodes in N'.*

3.2.2 Edit Operations on Graphs

Edit operations are commonly used to transform a string, a tree or a graph to another [11, 28, 30, 69, 79, 81, 90, 91, 96, 98]. The concept of edit distances is easily understood in comparison of the similarity between two strings, two trees or two graphs. In this section, several edit operations on complete relational graphs are formally defined. The edit distance and the matching distance between two graphs are presented in the next section.

Let λ denote a **null** node or arc. An edit operation is written as $a \rightarrow b$, where $a \rightarrow b \neq \lambda \rightarrow \lambda$, and a or b may be a node or an arc of a graph but if $a \neq \lambda$ and $b \neq \lambda$, both a and b must be two nodes or two arcs. The following six kinds of edit operations are used:

- node insertion: $\lambda \rightarrow a$ (a is a node)
- node substitution: $a \rightarrow b$ (a and b are nodes)
- node deletion: $a \rightarrow \lambda$ (a is a node)
- arc insertion: $\lambda \rightarrow a$ (a is an arc)
- arc substitution: $a \rightarrow b$ (a and b are arcs)
- arc deletion: $a \rightarrow \lambda$ (a is an arc)

These edit operations are used to transform a graph to another. In our application, the graphs under study are required to be complete. So some constraints on these operations are necessary:

- If a node of a graph is inserted, arcs that join this node and all the existing nodes of the graph are also be inserted.
- If a node of a graph is deleted, arcs that join this node and all the other nodes of the graph are also be deleted.
- An arc is deleted only when one or two of its end nodes are deleted.
- An arc is inserted only when one or two of its end nodes are inserted.

The application of an edit operation $a \rightarrow b$ to graph G_A results in G_B, which is written as $G_A \Rightarrow G_B$ via $a \rightarrow b$. Let E be a sequence $e_1, e_2, ..., e_m$ of edit

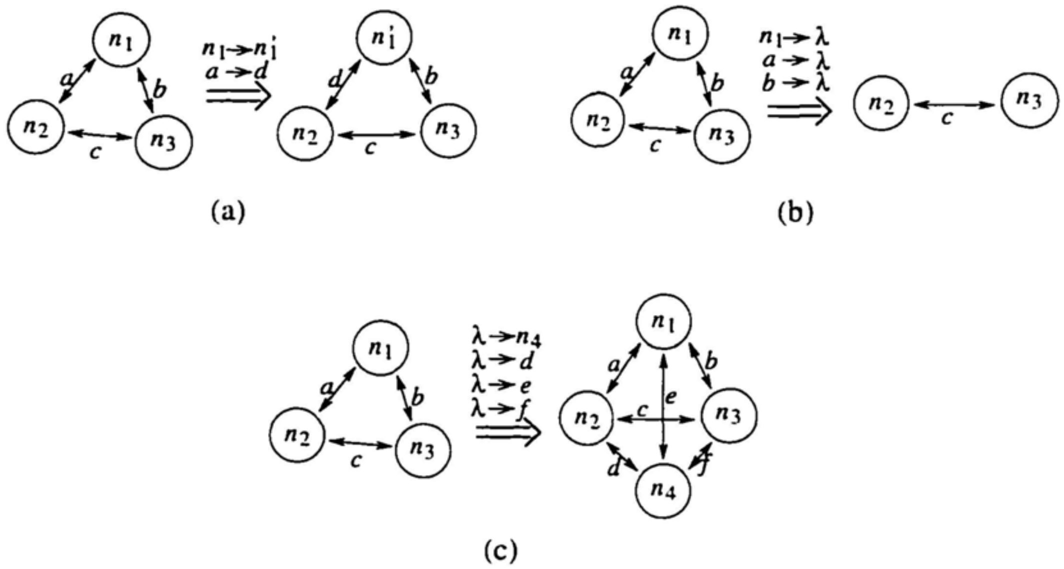


Figure 3.5: Examples of edit operations. (a) $G_1 \Rightarrow G_2$ via $n_1 \rightarrow n_1'$ and $a \rightarrow d$. (b) $G_1 \Rightarrow G_3$ via $n_1 \rightarrow \lambda, a \rightarrow \lambda$ and $b \rightarrow \lambda$. (c) $G_1 \Rightarrow G_4$ via $\lambda \rightarrow n_4, \lambda \rightarrow d, \lambda \rightarrow e$ and $\lambda \rightarrow f$.

operations. An edit transformation of graph G_A to graph G_B is a sequence of $G_0, G_1, ..., G_m$ such that $G_A = G_0, G_B = G_m$ and $G_{i-1} \Rightarrow G_i$ via e_i for $1 \leq i \leq m$. The transformation is also denoted by $G_A \Rightarrow G_B$. Several edit operations and transformations on graphs are shown in Fig. 3.5.

Note that for an edit transformation, in order to fulfill the constraints mentioned above, G_A and G_B need to be complete relational graphs, but $G_1, G_2, ...,$ G_{m-1} may not. For example, if e_1 is a node deletion operation, then G_1, which has a set of arcs each with only one node at its end, is not even any kind of graph defined in graph theory. However, sometimes we still call them graphs if there is no confusion.

3.2.3 Distances between Two Graphs

In practical recognition problems, objects belonging to the same class may have different degrees of distortion compared with their model object. As a consequence, the graphs representing them may also be different. To measure the similarity (or **distance**) between two graphs, costs associated with these edit operations are necessary. Let γ be a cost function that assigns to each edit operation $a \to b$ a nonnegative real number $\gamma(a \to b)$. γ can also be extended to a sequence of edit operations $E = e_1, e_2, \ldots, e_m$ by setting $\gamma(E) = \sum_{i=1}^{m} \gamma(e_i)$. If $m = 0$, i.e., no edit operation is applied, we define $\gamma(E) = 0$.

Definition 3.4 *Let G_i and G_j be two graphs. The* **edit distance** *from G_i to G_j is defined as*

$$\begin{aligned} \delta(G_i, G_j) &= \min\{\gamma(E) | E \text{ is a sequence of edit operations} \\ &\qquad \text{that transforms } G_i \text{ to } G_j\}. \end{aligned} \tag{3.1}$$

Theorem 3.1 *$\delta(G_i, G_j)$ is a metric on S_G if the following conditions are fulfilled, where S_G is the set of all (complete relational) graphs.*

(a) $\gamma(a \to a) = 0$;

(b) $\gamma(a \to b) > 0$ *if* $a \neq b$;

(c) $\gamma(a \to b) = \gamma(b \to a)$.

Proof.

(1) (Positivity) $\delta(G_i, G_j) \geq 0$ holds for all $G_i, G_j \in S_G$ since $\gamma(a \to b) \geq 0$.

(2) (Definiteness) On the one hand, if $G_i = G_j$, we have $\delta(G_i, G_j) = 0$ from the definition of $\delta(G_i, G_j)$. On the other hand, if $G_i \neq G_j$, in order to

transform G_i to G_j by a sequence of edit operations, at least one edit operation $\gamma(a \to b) > 0$ $(a \neq b)$ in the sequence must be applied, so $\delta(G_i, G_j) > 0$. This implies that if $\delta(G_i, G_j) = 0$, $G_i = G_j$.

(3) (Symmetry) Suppose $G_i \Rightarrow G_j$ by a sequence of edit operations $E_1 = e_1, e_2, ..., e_m$ and $\delta(G_i, G_j) = \gamma(E_1)$. A sequence $E_2 = e_m, e_{m-1}, ..., e_1$ will transform G_j to G_i and the transformation cost $\gamma(E_2) = \gamma(E_1)$ since $\gamma(a \to b) = \gamma(b \to a)$. Assume $\delta(G_j, G_i) \neq \gamma(E_2)$. By assumption, there must exist a sequence of edit operations $E_3 = e'_n, e'_{n-1}, ..., e'_1$ such that $G_j \Rightarrow G_i$ and $\gamma(E_3) < \gamma(E_2)$. With the application of the sequence $E_4 = e'_1, e'_2, ..., e'_n$ to G_i, $G_i \Rightarrow G_j$ will result and we have $\gamma(E_4) = \gamma(E_3) < \gamma(E_1)$, which is in contradiction to $\delta(G_i, G_j) = \gamma(E_1)$. Therefore, $\delta(G_j, G_i) = \delta(G_i, G_j)$.

(4) (Triangle inequality) We now show that $\delta(G_i, G_j) \leq \delta(G_i, G_k) + \delta(G_k, G_j)$ for all $G_i, G_j, G_k \in S_G$. Let $E_1 = e_1, e_2, ..., e_l$ transform G_i to G_k and $\delta(G_i, G_k) = \gamma(E_1)$, and let $E_2 = e_{l+1}, e_{l+2}, ..., e_m$ transform G_k to G_j and $\delta(G_k, G_j) = \gamma(E_2)$. Obviously, $E_3 = e_1, e_2, ..., e_l, e_{l+1}, ..., e_m$ transforms G_i to G_j. Hence $\gamma(E_3) = \gamma(E_1) + \gamma(E_2) = \delta(G_i, G_k) + \delta(G_k, G_j)$. By the definition of the edit distance $\delta(G_i, G_j)$, it is immediate that $\delta(G_i, G_j) \leq \gamma(E_3) = \delta(G_i, G_k) + \delta(G_k, G_j)$. □

Note that the edit distance fulfills the triangle inequality even if such a property does not hold for the cost function $\gamma(a \to b)$. In other words, $\gamma(a \to b) \leq \gamma(a \to c) + \gamma(c \to b)$ is not required.

There are infinite ways to transform a graph to another. For example, the substitution of a for b may be done not only by $a \to b$, but also by $a \to c$ and then $c \to b$. To simplify the problem of finding the distance between two graphs, a concept **mapping** is introduced in the following.

Definition 3.5 *Let $\Lambda_i = \{\lambda\}$ and $\Lambda_j = \{\lambda\}$ be two sets of null nodes. Let $G_i = (N_i, A_i)$ and $G_j = (N_j, A_j)$ be two graphs. A* **node mapping** *from G_i to G_j is a function*

$$f_N : N_i \cup \Lambda_i \to N_j \cup \Lambda_j$$

satisfying the following conditions:

(a) $f_N(\lambda) \neq \lambda$;

(b) If $n_i \neq m_i$ then $f_N(n_i) \neq f_N(m_i)$ for all $n_i, m_i \in N_i$ and $f_N(n_i), f_N(m_i) \in N_j$;

(c) For a node $n_i \in N_i$, there exists a node $n_j \in N_j \cup \Lambda_j$ such that $f_N(n_i) = n_j$;

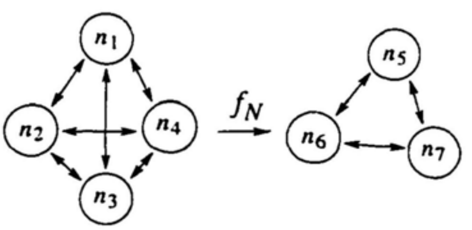
(d) For a node $n_j \in N_j$, there exists a node $n_i \in N_i \cup \Lambda_i$ such that $f_N^{-1}(n_j) = n_i$.

In this definition, for $n_i \in N_i$ and $n_j \in N_j$, $f_N(n_i) = \lambda$ and $f_N^{-1}(n_j) = \lambda$ indicate a node $n_i \in G_i$ and a node $n_j \in G_j$ are deleted. To guarantee that all the graphs under study are complete relational graphs, when a node is deleted, all arcs connecting it will be deleted too. Fig. 3.6 shows two examples of node mappings. From Definition 3.5 and Fig. 3.6 we can see that a node mapping f_N leads to an arc mapping.

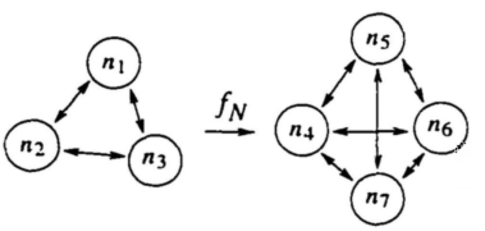
Definition 3.6 *Let $\Delta_i = \{\lambda\}$ and $\Delta_j = \{\lambda\}$ be two sets of null arcs. Let $G_i = (N_i, A_i)$ and $G_j = (N_j, A_j)$ be two graphs. An* **arc mapping** *led by a node mapping ($f_N : N_i \cup \Lambda_i \to N_j \cup \Lambda_j$) from G_i to G_j is a function*

$$f_A : A_i \cup \Delta_i \to A_j \cup \Delta_j$$

satisfying the following conditions:



(a)



(b)

Figure 3.6: Two node mappings. (a) $f_N(n_1) = \lambda$, $f_N(n_2) = n_5$, $f_N(n_3) = n_6$, $f_N(n_4) = n_7$. (b) $f_N(n_1) = \lambda$, $f_N(n_2) = n_4$, $f_N(n_3) = n_5$, $f_N(\lambda) = n_6$, $f_N(\lambda) = n_7$.

(a) For an arc $(n_i, m_i) \in A_i, n_i \neq m_i, n_i, m_i \in N_i$, *if* $f_N(n_i), f_N(m_i) \in N_j$, *then* $f_A((n_i, m_i)) = (f_N(n_i), f_N(m_i)) \in A_j$;

(b) For an arc $(n_i, m_i) \in A_i$, *if* $f_N(n_i) = \lambda$ *or* $f_N(m_i) = \lambda$ *or both* $f_N(n_i) = \lambda$ *and* $f_N(m_i) = \lambda$, *then* $f_A((n_i, m_i)) = \lambda \in \Delta_j$;

(c) For an arc $(n_j, m_j) \in A_j$, *if* $f_N^{-1}(n_j) = \lambda$ *or* $f_N^{-1}(m_j) = \lambda$ *or both* $f_N^{-1}(n_j) = \lambda$ *and* $f_N^{-1}(m_j) = \lambda$, *then* $f_A^{-1}((n_j, m_j)) = \lambda \in \Delta_i$.

As two examples, for Fig. 3.6(a), the arc mapping is:

$$f_A((n_1, n_2)) = \lambda, f_A((n_1, n_3)) = \lambda, f_A((n_1, n_4)) = \lambda, f_A((n_2, n_3)) = (n_5, n_6),$$
$$f_A((n_2, n_4)) = (n_5, n_7), f_A((n_3, n_4)) = (n_6, n_7),$$

and for Fig. 3.6(b), the arc mapping is:

$$f_A((n_1, n_2)) = \lambda, f_A((n_1, n_3)) = \lambda, f_A((n_2, n_3)) = (n_4, n_5), f_A^{-1}((n_6, n_5)) = \lambda,$$
$$f_A^{-1}((n_6, n_4)) = \lambda, f_A^{-1}((n_6, n_7)) = \lambda, f_A^{-1}((n_7, n_5)) = \lambda, f_A^{-1}((n_7, n_4)) = \lambda.$$

Here we call $f_N(n_i) = n_j$ and $f_A((n_i, m_i)) = (n_j, m_j)$ a node correspondence and an arc correspondence, respectively. The former associates node n_j to node n_i, and the latter associates arc (n_j, m_j) to arc (n_i, m_i).

For expression simplicity, we will also use "$\rightarrow$" to denote node and arc correspondences if there is no confusion. For example, $n_i \rightarrow n_j$ is equivalent to $f_N(n_i) = n_j$. Similarly, the arc correspondences $f_A((n_i, m_i)) = \lambda$ and $f_A^{-1}((n_j, m_j)) = \lambda$ will be written as $(n_i, m_i) \rightarrow \lambda$ and $\lambda \rightarrow (n_j, m_j)$, respectively, and if $n_i \neq \lambda, m_i \neq \lambda$, $n_i \rightarrow n_j \neq \lambda$, and $m_i \rightarrow m_j \neq \lambda$, then $f_A((n_i, m_i)) = (n_j, m_j)$ will be denoted by $(n_i, m_i) \rightarrow (n_j, m_j)$.

We can see that each of these node and arc correspondences corresponds to an identical edit operation, so we will also use the cost function γ to assign costs to these correspondences.

Definition 3.7 *Let $f_N : N_i \cup \Lambda_i \rightarrow N_j \cup \Lambda_j$ be a node mapping from $G_i = (N_i, A_i)$ to $G_j = (N_j, A_j)$, and let $f_A : A_i \cup \Delta_i \rightarrow A_j \cup \Delta_j$ be an arc mapping led by f_N. The pair (f_N, f_A) is termed a* **matching** *from G_i to G_j. The matching cost is calculated by*

$$\begin{aligned} \beta(f_N, f_A) &= \sum_{n_i \rightarrow n_j \in Q_1} \gamma(n_i \rightarrow n_j) + \sum_{n_i \rightarrow \lambda \in Q_2} \gamma(n_i \rightarrow \lambda) + \sum_{\lambda \rightarrow n_j \in Q_3} \gamma(\lambda \rightarrow n_j) \\ &+ \sum_{(n_i, m_i) \rightarrow (n_j, m_j) \in Q_4} \gamma((n_i, m_i) \rightarrow (n_j, m_j)) \\ &+ \sum_{(n_i, m_i) \rightarrow \lambda \in Q_5} \gamma((n_i, m_i) \rightarrow \lambda) + \sum_{\lambda \rightarrow (n_j, m_j) \in Q_6} \gamma(\lambda \rightarrow (n_j, m_j)) \end{aligned} \tag{3.2}$$

where (f_N, f_A) determines the sets Q_{1-6}, i.e., Q_1 is the set of $n_i \rightarrow n_j, n_i \in N_i, n_j \in N_j$; Q_2 the set of $n_i \rightarrow \lambda, n_i \in N_i$; Q_3 the set of $\lambda \rightarrow n_j, n_j \in N_j$; Q_4 the set of $(n_i, m_i) \rightarrow (n_j, m_j), n_i, m_i \in N_i, n_j, m_j \in N_j, n_i \neq m_i$; Q_5 the set of $(n_i, m_i) \rightarrow \lambda, n_i, m_i \in N_i$; Q_6 the set of $\lambda \rightarrow (n_j, m_j), n_j, m_j \in N_j$. The **matching distance** *from G_i to G_j is defined as*

$$\xi(G_i, G_j) = \min\{\beta(f_N, f_A) | (f_N, f_A) \text{ is a matching from } G_i \text{ to } G_j\}. \tag{3.3}$$

An **optimal matching** (f_N^*, f_A^*) *is a matching such that* $\beta(f_N^*, f_A^*) = \xi(G_i, G_j)$.

Comparing the definitions of the node mapping and the arc mapping with the definition of edit transformation, we can easily find the similarity between them, which implies that there is a relation between $\xi(G_i, G_j)$ and $\delta(G_i, G_j)$.

Lemma 3.1 *For a matching* (f_N, f_A) *from* G_i *to* G_j*, there exists a sequence* E *of edit operations, which transforms* G_i *to* G_j*, such that* $\gamma(E) = \beta(f_N, f_A)$.

Proof. The node mapping f_N and the arc mapping f_A consist of a set of node and arc correspondences, each of which is equivalent to an edit operation. These edit operations comprise a sequence E that transforms G_i to G_j. Thus $\gamma(E) = \beta(f_N, f_A)$. □

It is worth noting that for any sequence E of edit operations that transforms G_i to G_j, there may not exist a matching (f_N, f_A) such that $\beta(f_N, f_A) = \gamma(E)$. The reason is that clearly, $\beta(f_N, f_A)$ is finite,[1] so $\beta(f_N, f_A) < P$ where P is a positive value large enough, but a sequence E that transforms G_i to G_j with $\gamma(E) > P$ can easily be found because, say, a node correspondence $n_i \to n_j$ may be done by sufficiently many edit operations $n_i \to m, m \to p, \ldots, q \to n_j$ such that $\gamma(n_i \to m) + \gamma(m \to p) + \cdots + \gamma(q \to n_j) > P$.

Lemma 3.2 *For any edit sequence* E *transforming* G_i *to* G_j*, if the cost function fulfills the triangle inequality* $(\gamma(a \to c) \leq \gamma(a \to b) + \gamma(b \to c))$ *besides the conditions (a), (b) and (c) in Theorem 3.1, then there exists a matching* (f_N, f_A) *from* G_i *to* G_j *such that* $\beta(f_N, f_A) \leq \gamma(E)$.

[1] In this thesis, we consider only finite positive costs of edit operations (or node and arc correspondences), and deal with finite graphs, i.e., the cardinalities of the node sets of these graphs are finite.

Proof.

(1) Let N_1 be a set of nodes in G_i, to each node of which no any edit operations are applied in the transformation $G_i \Rightarrow G_j$. This set of nodes in G_j is denoted by N_1', where $|N_1| = |N_1'|$. For a node $n_1 \in N_1$, there is a node $n_1' \in N_1'$ such that the node correspondence cost

$$\gamma(n_1 \rightarrow n_1') = 0. \tag{3.4}$$

Similar conclusion can be drawn for a set of arcs in G_i, to each arc of which no any edit operations are applied.

(2) Let N_2 be a set of nodes in G_i which are deleted after $G_i \Rightarrow G_j$. Let $n_2 \in N_2$, and let the sequence of edit operations applied to n_2 be $n_2 \rightarrow m_1, m_1 \rightarrow m_2, ..., m_r \rightarrow \lambda$. From the triangle inequality for edit operations, we have the node correspondence cost

$$\gamma(n_2 \rightarrow \lambda) \leq \gamma(n_2 \rightarrow m_1) + \gamma(m_1 \rightarrow m_2) + \cdots + \gamma(m_r \rightarrow \lambda). \tag{3.5}$$

Similar conclusion can be drawn for a set of arcs in G_i which are deleted after $G_i \Rightarrow G_j$.

(3) Let N_3' be a set of nodes in G_j which are inserted after $G_i \Rightarrow G_j$. Let $n_3' \in N_3'$, and let the sequence of edit operations applied to a null node λ be $\lambda \rightarrow p_1, p_1 \rightarrow p_2, ..., p_s \rightarrow n_3'$. We also have the node correspondence cost

$$\gamma(\lambda \rightarrow n_3') \leq \gamma(\lambda \rightarrow p_1) + \gamma(p_1 \rightarrow p_2) + \cdots + \gamma(p_s \rightarrow n_3'). \tag{3.6}$$

Similar conclusion can be drawn for a set of arcs in G_j which are inserted after $G_i \Rightarrow G_j$.

(4) Let N_4 be a set of nodes in G_i which are substituted after $G_i \Rightarrow G_j$. Let $n_4 \in N_4$, and let the sequence of edit operations applied to n_4 be $n_4 \rightarrow q_1, q_1 \rightarrow$

$q_2, ..., q_t \to n'_4$, where $n'_4 \in N'_4$, $|N_4| = |N'_4|$ and $N'_4 \subseteq N_j$. We also have the node correspondence cost

$$\gamma(n_4 \to n'_4) \leq \gamma(n_4 \to q_1) + \gamma(q_1 \to q_2) + \cdots + \gamma(q_t \to n'_4). \quad (3.7)$$

Similar conclusion can be drawn for a set of arcs in G_i which are substituted after $G_i \Rightarrow G_j$.

The node correspondences, which correspond to the node correspondence costs on the left sides of (3.4)–(3.7), comprise a node mapping f_N from G_i to G_j. The arc mapping f_A led by f_N is not given explicitly for simplicity. The node edit operations, which correspond to the edit operation costs on the right sides of (3.4)–(3.7), and the arc edit operations not given explicitly comprise the edit sequence E. Adding all the node and arc correspondence costs and adding all the edit operation costs, we obtain

$$\beta(f_N, f_A) \leq \gamma(E). \quad \square$$

Theorem 3.2 *The edit distance $\delta(G_i, G_j)$ is equal to the matching distance $\xi(G_i, G_j)$, if the cost function γ satisfies the conditions in Theorem 3.1 and the triangle inequality.*

Proof. By Lemmas 3.1 and 3.2, Theorem 3.2 follows immediately. $\square$

Theorem 3.2 suggests that if γ is a metric, $\xi(G_i, G_j)$ is also a metric. In what follows, we will use the matching distance to measure the similarity between two graphs.

3.3 Representations of Chinese Characters

3.3.1 Stroke-Based Spatially Relational Representation

Since strokes are the most basic elements constructing Chinese characters, the idea comes first that using the standard strokes (see Table 2.2) as primitives to represent the structural information of Chinese characters. In the last chapter, We have describe the approach to recognizing input strokes. Obviously, strokes with different types are part of the features that are employed by human beings to identify Chinese characters.

As mentioned before, each Chinese character has its standard stroke writing order. If people almost always write a Chinese character according to its stroke order, the design of on-line Chinese character recognition systems will become much simpler. We can arrange the strokes of each model Chinese character in its standard stroke order to build a model stroke string base in advance. For an input character, the 2D recognition problem is now transformed into a 1D string matching problem by finding in the base the best matching string with the input stroke string. In general, a 1D recognition is easier to be solved and needs much less computational effort than a 2D one. String matching based approaches have been used in many on-line Chinese character recognition methods [21, 38, 55, 56, 64, 65, 86, 92].

However, there are lots of stroke order variations and stroke deformations in Chinese people's handwriting. These make it difficult to distinguish Chinese characters only by making use of the information of 1D stroke strings. Consider the characters shown in Fig. 3.7. Strokes of the characters are labeled with numbers indicating the stroke orders. Character 1 is a standard one and char-

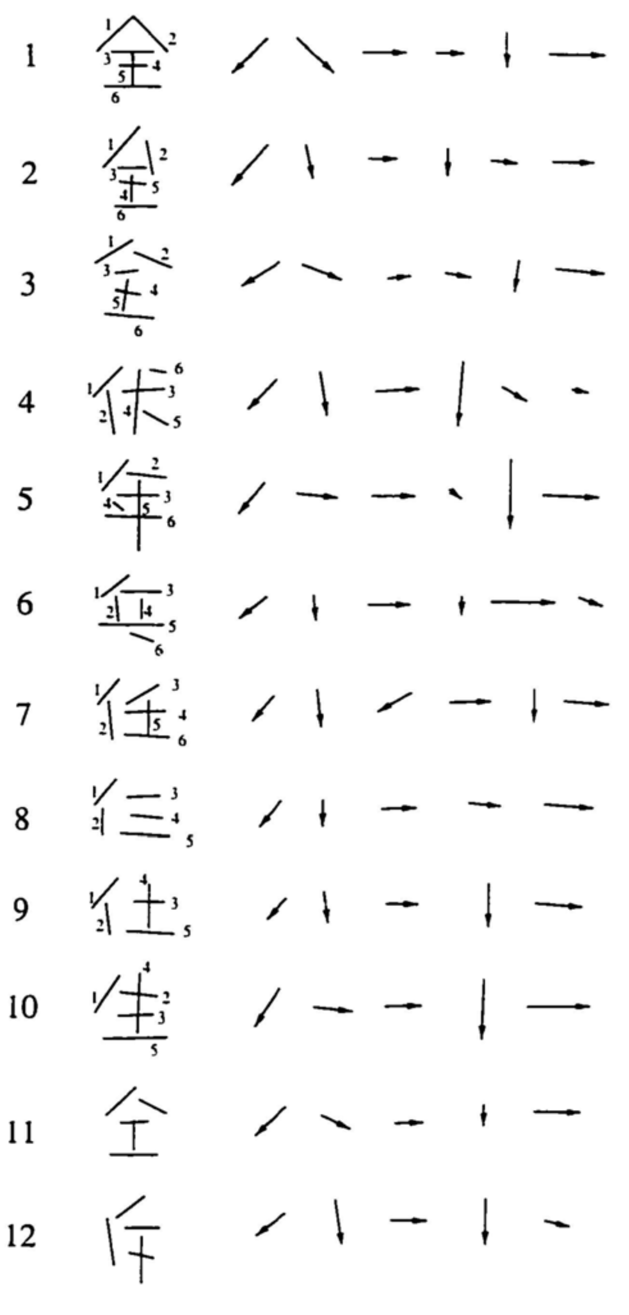


Figure 3.7: A set of characters with their strings of decomposed strokes. A number near a stroke indicates the order of the stroke when the character is written.

acters 2 and 3 are its common handwritten styles. The orders of two strokes in character 2 are exchanged compared with character 1. Characters 4–12 are other different characters. In the 2D plane, we can easily find that characters 2 and 3 are more similar to character 1 than characters 4–12. But this conclusion is difficult to draw if we only compare the stroke strings of these characters.

Let S_i be the ith string and $D(S_i, S_j)$ be the distance between S_i and S_j, $i, j \in \{1, 2, ..., 12\}$. By observing these strings, we have

$$D(S_1, S_2) \approx D(S_1, S_4) \approx D(S_1, S_6), \quad D(S_1, S_3) \approx D(S_1, S_5),$$

and

$$D(S_1, S_2) > D(S_4, S_2), \quad D(S_1, S_2) > D(S_6, S_2).$$

These lead to the result that when S_2 is inputted, it will be identified to be character 4 or 6 instead of character 1. Moreover, The strings of characters 10–12 are almost the same. In fact, from a matching point of view, all the strings in Fig. 3.7 are similar to each other, even though some of the characters have 1 more strokes than the others. Here we just give an example with the set of characters. Many similar examples exist in handwriting. Therefore, in order to develop a good on-line Chinese character recognition system, only the information of stroke strings of Chinese characters is not sufficient and the 2D structural features of Chinese characters must be utilized.

The relational graphs, as a tool of representation, are very suitable to represent the structural relations of Chinese characters, where nodes stand for primitives (strokes here), and arcs describe relations between these primitives. It is easy to come to mind that stroke types are used as the features of the primitives.

However, what relations between strokes are able to represent the deformation-tolerated features of a Chinese character as exactly as possible? Considering wide variations of handwriting, we choose three spatial relations to be the basic[2] relational features. They are relation 1 "below/above", relation 2 "right of/left of", and relation 3 "intersect/don't intersect". More detailed description of these relations is given in the following.

Let c_i and c_j be the geometric centers of stroke i and stroke j of a model Chinese character, respectively. A vector $r_{ij} = (a_{ij}^1, a_{ij}^2, a_{ij}^3)$, $i \neq j$, is used to represent the basic spatial relations from stroke i to stroke j, where $a_{ij}^1, a_{ij}^2 \in \{0, 1, 2, 3, 4\}$ and $a_{ij}^3 \in \{0, 1, 2\}$. $a_{ij}^1 = 0, 1, 2, 3$ and 4 indicate that c_i is below, is above, may be below or above, must be below, and must be above c_j, respectively. Also, $a_{ij}^2 = 0, 1, 2, 3$ and 4 indicate that c_i is on the right of, is on the left of, may be on the right of or the left of, must be on the right of, and must be on the left of c_j, respectively. $a_{ij}^3 = 0, 1$ and 2 indicate that stroke i and stroke j do not intersect, intersect, and may intersect or not, respectively. Here, $a_{ij}^k = 0$ or 1 ($k = 1, 2, 3$) is termed the "should" feature, $a_{ij}^k = 2$ ($k = 1, 2, 3$) the "don't care" feature, and $a_{ij}^k = 3$ or 4 ($k = 1, 2$) the "must" feature.

Note that the "must" feature is not used for a_{ij}^3. This is because many strokes in Chinese characters are easily written intersecting each other while they are not supposed to do so in standard writing, and on the other hand, two strokes that should intersect may easily written as two non-intersected ones.

The relational graph representation of a model Chinese character is obtained by assigning suitable values to each relational vector r_{ij}. Investigating handwritten Chinese characters, we learn that a relation between two strokes can be

[2]The term "basic" means that more other relations may be added when necessary.

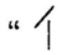
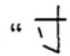
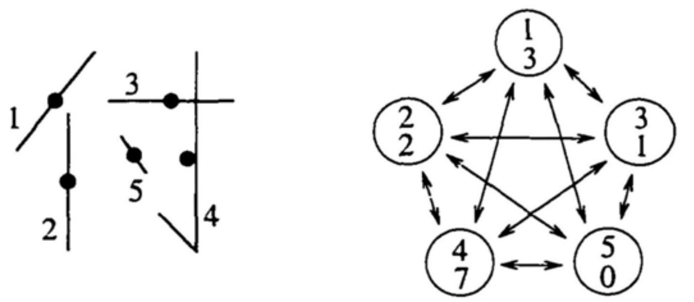
instable, stable or very stable, so we use the "don't care", "should" or "must" features to denote it. As an example, consider the character shown in Fig 3.8(a). Let c_i $(i = 1, 2, ..., 5)$ be the geometric center of stroke i. It is not difficult for a person familiar with Chinese character handwriting to find the fact that c_i is sometimes on the left and sometimes on the right of c_2, c_1 is above c_2 and they don't intersect in very high probability. Thus we set $a_{12}^1 = 1$, $a_{12}^2 = 2$, and $a_{12}^3 = 0$. Because the component "  " must be located on the left of the component "  ", the "must" feature is chosen so a_{13}^2, a_{14}^2, a_{15}^2, a_{23}^2, a_{24}^2, and a_{25}^2 are all set to be 4. Other relation value assignments can be seen in Fig 3.8(c).

Fig 3.8(b) shows the graph representing the character in Fig 3.8(a). A node of the graph represents a stroke by containing the stroke number (the upper number) and the stroke type (the lower number). All the standard stroke types are shown in Table 2.2. Note that now a new stroke type 0, which is not included in the table, is used to denote a short stroke that is easy to be written as a stroke belonging to one of the stroke types 1–4. In Fig. 3.8(a), stroke 5 is such a stroke so its type is set to 0.

There is a relation between vectors $r_{ij} = (a_{ij}^1, a_{ij}^2, a_{ij}^3)$ and $r_{ji} = (a_{ji}^1, a_{ji}^2, a_{ji}^3)$:

$$a_{ji}^k = \begin{cases} 1 & \text{if } a_{ij}^k = 0 \\ 0 & \text{if } a_{ij}^k = 1 \\ 2 & \text{if } a_{ij}^k = 2 \\ 4 & \text{if } a_{ij}^k = 3 \\ 3 & \text{if } a_{ij}^k = 4, \end{cases}$$



(a) (b)

	1	2	3	4	5
1		(1,2,0)	(2,4,0)	(2,4,0)	(2,4,0)
2	(0,2,0)		(2,4,0)	(2,4,0)	(2,4,0)
3	(2,3,0)	(2,3,0)		(2,2,1)	(4,2,0)
4	(2,3,0)	(2,3,0)	(2,2,1)		(2,0,2)
5	(2,3,0)	(2,3,0)	(3,2,0)	(2,1,2)	

(c)

Figure 3.8: Stroke-based spatially relational representation. (a) A Chinese character. (b) Complete relational graph representing the character. (c) Corresponding spatial relation matrix. A point on or near a stroke indicates the geometric center of the stroke. Different strokes are labeled with different numbers. A node of the graph represents a stroke by containing its number and type (the lower number).

where $k = 1, 2$, and

$$a_{ji}^{3} = \begin{cases} 1 & \text{if } a_{ij}^{3} = 0 \\ 0 & \text{if } a_{ij}^{3} = 1 \\ 2 & \text{if } a_{ij}^{3} = 2. \end{cases}$$

This property is useful for saving the memory space of a model graph base.

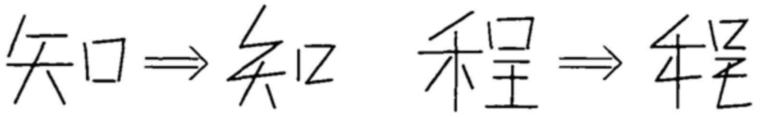
For an input character, the computation of its graph includes extracting every stroke, identifying the type of each stroke, and finding the relation vector $r'_{mn} = (a'^{1}_{mn}, a'^{2}_{mn}, a'^{3}_{mn})$ from stroke m to stroke n, where $(a'^{1}_{mn}, a'^{2}_{mn}, a'^{3}_{mn}) \in \{0, 1\}$. $a'^{k}_{mn} = 0$ or 1 has the same relational meaning as $a^{k}_{ij} = 0$ or 1, $k = 1, 2, 3$, except that a'^{k}_{mn} represents the relation from input stroke m to input stroke n while a^{k}_{ij} is the relation from model stroke i to model stroke j. In addition, stroke type 0 is not used for input strokes because in handwriting, short strokes are easily written as long as some long strokes, and vice versa.

Remark 1. From Table 2.2, we can see that the table contains 18 model stroke types: 15 standard ones and 3 frequently-used connected ones. In creating the graph base of model Chinese characters, we employ only the standard stroke types and the new stroke type 0. If we find a stroke of an input character is of stroke type 16, 17 or 18, we will use two standard strokes to represent it in the construction of the graph of the input character.

Remark 2. The creation of a model graph base seems to be a heavy task. We assign relation values between two strokes mainly based on the human knowledge of Chinese characters. However, the fact that Chinese characters may be constructed by much fewer components each with less than seven strokes can ease this task. We will give a detailed description of how to create the graphs of model Chinese characters in Section 7.2.

3.3.2 Segment-Based Spatially Relational Representation

There are lots of connected strokes in free fast Chinese character handwriting. Fig. 3.9 shows two examples. As mentioned in Section 2.4, the connected strokes



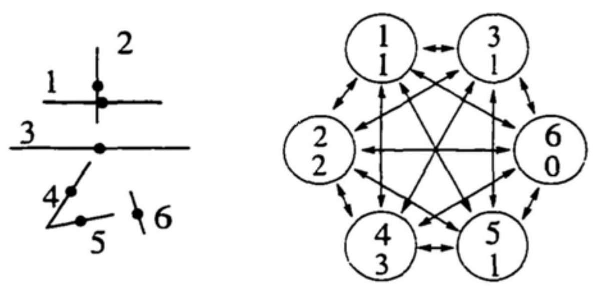
(a) (b)

Figure 3.9: Two model characters and their handwritten styles.

of the handwritten characters in the examples cannot be be detected. It is clear that stroke-based approaches are difficult to recognize such characters. In these cases, segment-based methods may play an important role.

Let us look at Fig. 3.9. The handwritten characters basically have the same segment types and relations that their corresponding model characters have except an extra segment in Fig. 3.9(a) and a segment and an extra segment in Fig. 3.9(b). In practice, there may be various connected strokes. With the help of the preprocessing, we can detect and then delete many extra segments in handwritten characters to facilitate the recognition.

Segment-based representation of Chinese characters is similar to the stroke-based representation but the primitives used are segments. An example is illustrated in Fig. 3.10, in which the two-segment stroke is represented with two nodes (segments). Recall that we have defined six segment types: type 1 " $\rightarrow$ " (-20°,30°], type 2 " $\downarrow$ " (250°,290°], type 3 " $\swarrow$ " (180°,250°], type 4 " $\searrow$ "



(a) (b)

	1	2	3	4	5	6
1		(2,2,1)	(4,2,0)	(4,2,0)	(4,2,0)	(4,2,0)
2	(2,2,1)		(4,2,2)	(4,2,0)	(4,2,0)	(4,2,0)
3	(3,2,0)	(3,2,2)		(4,2,2)	(4,2,0)	(4,2,0)
4	(3,2,0)	(3,2,0)	(3,2,2)		(1,2,0)	(2,1,0)
5	(3,2,0)	(3,2,0)	(3,2,0)	(0,2,0)		(2,1,2)
6	(3,2,0)	(3,2,0)	(3,2,0)	(2,0,0)	(2,0,2)	

(c)

Figure 3.10: Segment-based spatially relational representation. (a) A Chinese character. (b) Complete relational graph representing the character. (c) Corresponding spatial relation matrix.

(290°,340°], type 5 " $\nearrow$ " (30°,75°] and type 0 denoting an unstable short 1-segment stroke that is easy to be written as a segment belonging to one of the segment types 1–4. Segment 6 in Fig. 3.10(a) is a short one having segment type 0.

3.3.3 Spatially-Temporally Relational Representations

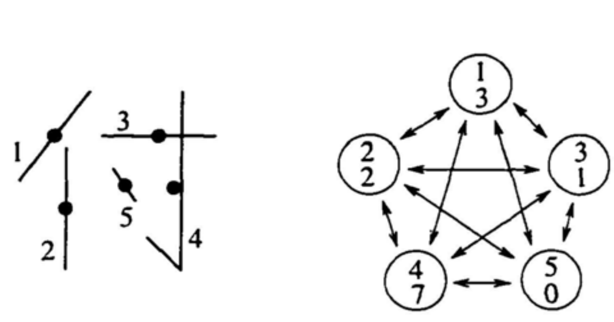
In on-line Chinese character recognition, an on-line device can capture the temporal information of the writing, which lets the order of strokes (segments) of an input character be known. Moreover, each Chinese character has a standard stroke writing order and Chinese people write a character basically (but not exactly) according to its stroke order. That is to say, most of the relative stroke (segment) order relations of a Chinese character are stable in daily handwriting. This fact makes both many methods [3, 21, 22, 55, 56, 57, 61, 92, 93] and the popular commercial products such as 蒙恬 and 揚友 in the Asian market utilize the temporal information to reach the recognition goal. In this section, we incorporate this stroke (segment) order information into the previous representations of Chinese characters.

Stroke-Based Spatially-Temporally Relational Representation. Recall that we use a relation vector $r_{ij} = (a_{ij}^1, a_{ij}^2, a_{ij}^3)$ to denote the relations from stroke i to stroke j. In order to represent the temporal information of strokes, r_{ij} is extended to a 4-dimensional vector[3] $r_{ij} = (a_{ij}^1, a_{ij}^2, a_{ij}^3, a_{ij}^4)$ where a_{ij}^1, a_{ij}^2, and a_{ij}^3 have the same definitions as before. $a_{ij}^4 = 0, 1$, and 2 indicate that stroke i is written before, after, and before or after stroke j, respectively. Similarly, $a_{ij}^4 = 0$ or 1 is termed the "should" feature and $a_{ij}^4 = 2$ the "don't care" feature.

In general, the rule of writing order of Chinese characters is that (1) write a character from its top to its bottom and from its left to its right, and (2) if a character consists of two or more components, finish writing a component before writing the next component.

Fig. 3.11 shows an example of the spatially-temporally relational representation. Fig. 3.11(a) and (b) are the same as Fig. 3.8(a) and (b), respectively, but in the relation matrix (Fig. 3.11(c)), the relative order relations between strokes are added. The numbers labeling the strokes also indicate the standard stroke order of writing of the character. The character has two components "亻" and "寸". Chinese people always write the former component first and then the

[3] We also use r_{ij} to denote a 4-dimensional vector when there is no confusion.



(a) (b)

	1	2	3	4	5
1		(1,2,0,0)	(2,4,0,0)	(2,4,0,0)	(2,4,0,0)
2	(0,2,0,1)		(2,4,0,0)	(2,4,0,0)	(2,4,0,0)
3	(2,3,0,1)	(2,3,0,1)		(2,2,1,2)	(4,2,0,0)
4	(2,3,0,1)	(2,3,0,1)	(2,2,1,2)		(2,0,2,2)
5	(2,3,0,1)	(2,3,0,1)	(3,2,0,1)	(2,1,2,2)	

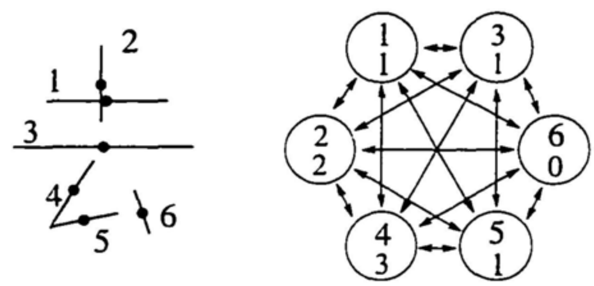
(c)

Figure 3.11: Stroke-based spatially-temporally relational representation. (a) A Chinese character. (b) Complete relational graph representing the character. (c) Corresponding spatial-temporal relation matrix.

latter. In addition, it is very common that stroke 1 is written before stroke 2 and stroke 3 before stroke 5. However, we found some people may write stroke 4 before strokes 3 and 5 or in the last. So we set $a^4_{34} = a^4_{45} = 2$. a^4_{ij} and a^4_{ji} have the following relation:

$$a^4_{ji} = \begin{cases} 1 & \text{if } a^4_{ij} = 0 \\ 0 & \text{if } a^4_{ij} = 1 \\ 2 & \text{if } a^4_{ij} = 2. \end{cases}$$

Segment-Based Spatially-Temporally Relational Representation. This representation is similar to the stroke-based spatially-temporally relational representation, but the primitives used are segments instead of strokes. An example is illustrated in Fig. 3.12.



(a) (b)

	1	2	3	4	5	6
1		(2,2,1,2)	(4,2,0,0)	(4,2,0,0)	(4,2,0,0)	(4,2,0,0)
2	(2,2,1,2)		(4,2,2,2)	(4,2,0,0)	(4,2,0,0)	(4,2,0,0)
3	(3,2,0,1)	(3,2,2,2)		(4,2,2,0)	(4,2,0,0)	(4,2,0,0)
4	(3,2,0,1)	(3,2,0,1)	(3,2,2,1)		(1,2,0,0)	(2,1,0,0)
5	(3,2,0,1)	(3,2,0,1)	(3,2,0,1)	(0,2,0,1)		(2,1,2,0)
6	(3,2,0,1)	(3,2,0,1)	(3,2,0,1)	(2,0,0,1)	(2,0,2,1)	

(c)

Figure 3.12: Segment-based spatially-temporally relational representation. (a) A Chinese character. (b) Complete relational graph representing the character. (c) Corresponding spatial-temporal relation matrix.

Remark 1. The assignment of values to a^4_{ij} is according to the human knowledge of Chinese character handwriting. First we may put more effort on getting the order relations between strokes/segments of components. Since the structures of components are simpler and the number of components is much fewer than those of Chinese characters, this task can be done, without much difficulty, by people who are familiar with Chinese characters, together with the help of some experiments. Then, we arrange several components to form a Chinese character in the order that the components are written in standard writing, and thus obtain all the stroke/segment order relations between any two strokes/segments of the character.

Remark 2. The use of stroke/segment order information is beneficial to reducing graph matching[4] time (see Section 4.4.3). However, the Chinese characters written with great stroke order variations may not be recognized correctly. For tolerating such stroke order deviations, we design our recognition approach having two phases. Phase 1 uses both spatial and temporal relations among strokes/segments to do the recognition task. In phase 2, no stroke/segment order relations are employed and thus writing a Chinese character in any stroke order is allowed. This flexible way gives a user another choice when he/she writes Chinese characters with too many stroke order deviations and at the same time obtains incorrect classification results.

3.4 Assigning Costs to Node and Arc Correspondences

In Section 3.2.3, we have defined the matching distance between two graphs. The computation of the distance needs to define the costs of node and arc correspondences in advance. This section introduces the assignments of these cost values for stroke-based relational graph matching and segment-based relational graph matching.

[4]As conventional, we use the term "graph matching" to denote the process of finding the distance between two graphs.

3.4.1 Assigning Costs for Stroke-Based Relational Graph Matching

Recall that the matching distance between graphs G_i and G_j is defined as

$$\xi(G_i, G_j) = \min\{\beta(f_N, f_A) | (f_N, f_A) \text{ is a matching from } G_i \text{ to } G_j\}, \tag{3.8}$$

where f_N and f_A are termed a node mapping and an arc mapping, respectively, and

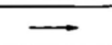
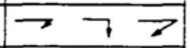
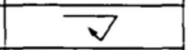
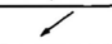
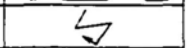
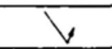
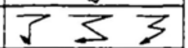
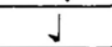
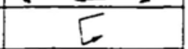
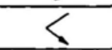
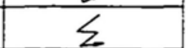
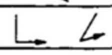
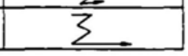
$$\begin{aligned}\beta(f_N, f_A) \;=\;& \sum_{n_i \to n_j \in Q_1} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q_2} \gamma(n_i \to \lambda) + \sum_{\lambda \to n_j \in Q_3} \gamma(\lambda \to n_j) \\ &+ \sum_{(n_i, m_i) \to (n_j, m_j) \in Q_4} \gamma((n_i, m_i) \to (n_j, m_j)) \\ &+ \sum_{(n_i, m_i) \to \lambda \in Q_5} \gamma((n_i, m_i) \to \lambda) + \sum_{\lambda \to (n_j, m_j) \in Q_6} \gamma(\lambda \to (n_j, m_j))\end{aligned} \tag{3.9}$$

is termed a matching cost. Here $\gamma(n_i \to n_j), \gamma(n_i \to \lambda)$ and $\gamma(\lambda \to n_j)$ are node correspondence costs, and $\gamma((n_i, m_i) \to (n_j, m_j)), \gamma((n_i, m_i) \to \lambda)$ and $\gamma(\lambda \to (n_j, m_j))$ are arc correspondence costs. For stroke-based graph matching, they correspond to stroke and stroke relation correspondence costs, respectively.

The assignments of costs to different stroke (type) correspondences are done mainly according to human knowledge of the stroke type variations. Moreover, the stroke type recognition method presented in Section 2.3 is also helpful for determining whether two strokes are easily confused. We show the model strokes in Table 3.1 again for convenient description.

In Chinese character handwriting, if stroke A may easily be written like stroke B but not like stroke C, then we should assign lower cost to the former case than to the latter. For example, Chinese people often use type 2 strokes to

Table 3.1: 18 model strokes.

Type	Strokes	Type	Strokes
1		10	
2		11	
3		12	
4		13	
5		14	
6		15	
7		16	
8		17	
9		18	

stand for type 7 strokes, and vice versa. Type 1 strokes and type 4 strokes are also easily confused. Moreover, from the experiments given in Section 2.3, we know that (type 8, type 9), (type 6, type 12) and (type 11, type 15) are similar pairs. Table 3.2 summarizes all the stroke type correspondence costs.

We mention again that stroke type 0 is used to denote short strokes of model characters, and the three frequently-used (not standard) input stroke types 16, 17 and 18, if detected in preprocessing, are split into their corresponding standard strokes.

In Table 3.2, there is a new stroke type 20. It is used to denote the unknown-type input strokes that are considered unlike any model stroke in the stroke type recognition. Because a stroke of type 20 is generally a multi-segment stroke, we assign smaller costs to its correspondence with multi-segment standard strokes.

Table 3.2 defines the cost function $\gamma(n_i \to n_j)$ where n_i $(\neq \lambda)$ and n_j $(\neq \lambda)$ are two nodes in G_i and G_j, respectively. For the node deletion cost $\gamma(n_i \to \lambda)$

Table 3.2: Costs associated with stroke type correspondences.

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	20
0	0	2	2	2	2	6	6	6	6	6	6	9	9	9	9	9	9
1	2	0	7	7	3	3	7	7	7	7	5	8	8	9	9	9	9
2	2	7	0	3	3	7	5	3	6	7	8	8	8	9	9	9	9
3	2	7	3	0	7	7	7	5	7	8	8	8	8	9	9	9	9
4	2	3	3	7	0	7	4	7	7	7	8	8	8	9	9	9	9
5	6	3	7	7	7	0	7	7	7	7	8	8	8	9	9	9	9
6	6	7	5	7	4	7	0	7	7	5	8	8	4	9	9	9	8
7	6	7	3	5	7	7	7	0	7	7	7	7	8	9	9	9	8
8	6	7	6	7	7	7	7	7	0	3	7	7	5	9	9	9	8
9	6	7	7	8	7	7	5	7	3	0	7	7	4	9	9	9	8
10	6	5	8	8	8	8	8	7	7	7	0	4	9	9	9	5	8
11	9	8	8	8	8	8	8	7	7	7	4	0	9	9	7	4	7
12	9	8	8	8	8	8	4	8	5	4	9	9	0	7	9	9	7
13	9	9	9	9	9	9	9	9	9	9	9	9	7	0	9	9	7
14	9	9	9	9	9	9	9	9	9	9	9	7	9	9	0	9	7
15	9	9	9	9	9	9	9	9	9	9	5	4	9	9	9	0	7
20	9	9	9	9	9	9	8	8	8	8	8	7	7	7	7	7	0

or $\gamma(\lambda \to n_j)$, after some experimental tests, we choose

$$\gamma(n_i \to \lambda) = \gamma(\lambda \to n_j) = 5. \tag{3.10}$$

Let n_i ($\neq \lambda$) and m_i ($\neq \lambda$) be two nodes in graph G_i, and n_j ($\neq \lambda$) and m_j ($\neq \lambda$) be two nodes in G_j. Then the relations from n_i to m_i and the relation from n_j to m_j are denoted by vectors $r_{n_i m_i} = (a^1_{n_i m_i}, a^2_{n_i m_i}, a^3_{n_i m_i}, a^4_{n_i m_i})$ and $r_{n_j m_j} = (a^1_{n_j m_j}, a^2_{n_j m_j}, a^3_{n_j m_j}, a^4_{n_j m_j})$, respectively. The arc correspondence cost $\gamma((n_i, m_i) \to (n_j, m_j))$ is defined as

$$\gamma((n_i, m_i) \to (n_j, m_j)) = \sum_{k=1}^{4} w_k \eta(a^k_{n_i m_i}, a^k_{n_j m_j}), \tag{3.11}$$

where w_{1-4} are weighting factors and η is defined by Table 3.3.

Table 3.3: Definition of the function η.

	0	1	2	3	4
0	0	1	0	0	M
1	1	0	0	M	0
2	0	0	0	0	0
3	0	M	0	0	M
4	M	0	0	M	0

In Table 3.3, $\eta(a^k_{n_i m_i}, a^k_{n_j m_j}) = 0$ suggests that the kth relation from n_i to m_i is compatible with the kth relation from n_j to m_j, and $\eta(a^k_{n_i m_i}, a^k_{n_j m_j}) = 1$ or M means the two relations are incompatible. For example, if $a^1_{n_i m_i} = 0$ (denoting that the geometric center of stroke n_i is below that of stroke m_i) and $a^1_{n_j m_j} = 0$, 1, 2, 3 and 4 (denoting that the geometric center of stroke n_j is below, is above, may be below or above, must be below, and must be above that of stroke m_j), then it is clear that the relation implied by $a^1_{n_i m_i} = 0$ is compatible with the relation implied by $a^1_{n_j m_j} = 0$, 2 or 3, and is incompatible with the relation implied by $a^1_{n_j m_j} = 1$ or 4. For the latter case, a cost (> 0) is assigned to this incompatible arc correspondence. Obviously, incompatible "must" relation correspondence should be punished by paying much higher cost, so we set $M = 10$.

From the definitions of the edit operations on complete relational graphs, we know that arc deletions are caused by node deletions. So we set 0 to all the arc deletion costs $\gamma((n_i, m_i) \to \lambda)$ and $\gamma(\lambda \to (n_j, m_j))$.

Let us consider the matching cost in (3.9). Large weights w_{1-4} mean that arc correspondence costs play a more important role in graph similarity comparisons; on the other hand, small weights make the importance of node correspondences increase. We found in our experiments that $w_1 = 6, w_2 = 6, w_3 = 4$ and $w_4 = 6$ can result in satisfactory recognition rate. As some strokes of Chinese characters are easily written intersecting each other while they are not supposed to do so in standard writing, we set w_3 a smaller value.

Remark. The node and arc correspondence cost function γ defined above is not a metric, which leads to the fact that the matching distance $\xi(G_i, G_j)$, in general, is not a metric either. (For example, $\eta(0,1) \not\leq \eta(0,2) + \eta(2,1)$ makes γ not satisfy the triangle inequality.) However, distances are not necessarily metrics in pattern recognition problems [30, 34, 35]. In fact, many distances (or similar measures) proposed in the pattern recognition literature are not metrics such as those in [30, 34, 35, 56, 69, 79, 83], but they may still be useful in comparing the similarity between objects. By the way, to our knowledge, no authors claimed that the distance measures used in their on-line Chinese character recognition methods are metrics.

3.4.2 Assigning Costs for Segment-Based Relational Graph Matching

There are only six segment types: type 1 " $\rightarrow$ ", type 2 " $\downarrow$ ", type 3 " $\swarrow$ ", type 4 " $\searrow$ ', type 5 " $\nearrow$ " and type 0 (denoting short segments) in the segment-based relational graph representation. Hence the assignment of costs $(\gamma(n_i \rightarrow n_j), n_i \neq \lambda, n_j \neq \lambda)$ to node (segment) correspondences is relatively direct, as shown in Table 3.4.

Table 3.4: Costs associated with segment type correspondences.

	0	1	2	3	4	5
0	0	1	1	1	1	7
1	1	0	7	7	2	2
2	1	7	0	2	2	7
3	1	7	2	0	7	7
4	1	2	2	7	0	7
5	7	2	7	7	7	0

Now we consider the assignment of costs to node deletions. The segment-based recognition methods proposed in this thesis are used to recognize more freely written Chinese characters that in general consist of more connected and missing strokes. As mentioned previously, the preprocessing algorithms presented in Chapter 2 cannot detect all extra segments. Therefore, it is very often that the segment number of an input character (after preprocessing) is different from that of its corresponding model character. We regard it reasonable that the segment deletion costs vary with the node numbers of the graphs under matching. For example, deleting any one of the segments of the 3-segment character " 大 " yields a different character or symbol, but for characters with more segments (e.g., 12), it may still remain recognizable after deleting one or two of its segments. Obviously, higher cost should be paid for the former case than the latter. Therefore, the node deletion costs are defined by

$$\gamma(n_i \to \lambda) = \gamma(\lambda \to n_j) = k(p), \tag{3.12}$$

where n_i and n_j are two nodes of graph G_i and G_j under matching, respectively; p is the node number of the model graph (G_i or G_j).

The cost $k(p)$ is not difficult to choose with the help of some experiments. For example, when $9 \le p \le 15$, we take $k(p)$ to be 4, and when $1 \le p \le 5$, we may take $k(p)$ to be 5.

The arc (segment relation) correspondence costs ($\gamma((n_i, m_i) \to (n_j, m_j))$, $(n_i, m_i) \ne \lambda$, $(n_j, m_j) \ne \lambda$) are defined the same as those in the last section (see (3.11) and Table 3.3), and the arc deletion costs are also set to 0.

3.5 Summary

In this chapter, we have formally defined the complete relational graphs and the distances for measuring the similarity between two graphs. With such graphs, we have proposed several relational representations for on-line Chinese character recognition. We have also dealt with the problem of assigning costs to node and arc correspondences in graph matching.

The stroke-based representations may be used to recognize relatively neat Chinese character handwriting while the segment-based representations will ease the recognition of more freely written characters. These representations have the following advantages:

- The representations incorporate the human knowledge of Chinese characters and can reflect their features well (except some very similar character pairs). In a complete relational graph, nodes describe primitive (stroke or segment) types and arcs represent the spatial and temporal relations between any two primitives. The proposed "don't care", "should" and "must" relational features allow us to represent unstable, stable and very stable primitive relations conveniently. Relations between **any** two primitives give much information and are very beneficial to the matching procedures, which will be discussed in the next chapter.
- The graph representations are directly based on strokes or segments. To obtain the representations, examining whether a stroke or segment belongs to some component is not required. However, the representations in [18, 68], as mentioned in Section 1.2, need to correctly extract components of Chinese characters first. The recognition method based on the representation in [13] also need to find components before performing recognition of a character. In fact, wide handwriting variations make it very difficult to extract components of Chinese characters at a high rate of success. In [16], the authors adopted only the relations between segments within the same components in their graph representation. This results in two shortcomings: (1) some relations represented in an input graph may not appear in its corresponding model graph, and vice versa; (2) most of the relations between segments are not utilized.
- The spatial and temporal relations between primitives are, at the first time, unified into the graph representations, which fully captures the online information of handwriting. The use of the primitive order relations

enhances the discrimination ability of the representations and helps to speed up the graph matching. Because of the "don't care" feature, the representations can tolerate common stroke order deviations.

- If the weight w_4 in (3.11) is set to 0 in graph match, then the stroke order relation will be ignored and our recognition methods presented in the next two chapters will be stroke order free.

Creation of a model character base for our recognition goal mainly depends on the human knowledge of Chinese characters, and thus is a relatively heavy task. It can be eased by constructing the graphs of components of Chinese characters first and then combining several component graphs to form the whole graph of a character.

Parts of the results presented in this chapter have been published in [58, 59, 60, 61, 62, 63].

Chapter 4

A State Space Search Method

4.1 Introduction

In the last chapter, we have introduced several graph representations and defined the matching distance for on-line Chinese character recognition. Now the problem of recognition of an input Chinese character can be transformed into a problem of graph matching. The term **graph matching**, as conventional, is used to denote the process of finding the distance between two graphs. Unfortunately, so far there are no efficient algorithms for our graph matching problem. Given two graphs G_i and G_j both with n nodes, a naive algorithmic approach to calculating the distance $\xi(G_i, G_j)$ is to generate all $n!$ permutations of the nodes and test them for being a solution (suppose $\Lambda_i = \Lambda = \emptyset$, and $\Delta_i = \Delta_j = \emptyset$, i.e., no node and arc deletions are carried out), then this algorithm has the computational complexity at least $O(n!)$.

Problems of such kind, which allow noise or structural deformations in practical object recognition, are extensions of the NP-complete subgraph isomorphism problem in graph theory [29, 46, 47] and are also NP-complete [33, 98]. All the problems in this class are believed to be intractable, i.e., no polynomial algorithms exist for one of the problems. If there is an efficient algorithm for some NP-complete problem, that is to say, the worst-case complexity of the algorithm is bounded by a polynomial function of the problem's parameters, then there is an efficient algorithm for every problem in the class [25, 75].

For our on-line recognition problem, because large categories of Chinese characters and the real-time recognition requirement, an algorithm is applicable only when it recognizes an input Chinese character with less than three seconds on some computer.[1] The less computational power the computer has, the fast the algorithm has to be. In addition, since we are dealing with a NP-complete problem, we cannot expect to obtain fast recognition speed by running an exponential-time algorithm without using any heuristic information.

Let us consider an example. As mentioned above, the exhaustive search for the simplified calculation of distance $\xi(G_i, G_j)$ needs at least $O(n!)$ time. Assume that after a preclassification stage, the graph of an input character has to be matched with 300 model graphs, and the node numbers of all these graphs are 12. Thus the computational time to recognize the input character is at least $A \times 12! \times 300$ ($\approx A \times 1.4 \times 10^{11}$), where A is the time required by a computer to perform one basic calculation. If A denotes one addition operation, then the recognition of an input character will take at least 15000 seconds ($>$ 4 hours) on a 166MHz PC/Pentium! It is clear that we have to seek efficient approaches

[1] In general, a Chinese needs about three seconds to write a ten-stroke character. An on-line recognition system can be designed to work in the way while one is writing a character, the system is recognizing the characters preceding it. In this case, the system should not take more than three seconds to finish the recognition of an input character.

to our graph matching problem.

In this chapter, we propose an efficient state space search method for the problem. The rest of this chapter is organized as follows. In Section 4.2, the graph matching is formulated as a search problem in a state space tree. The A^* algorithm that is employed to perform the search is presented in Section 4.3. Several schemes for increasing the search efficiency of the A^* are proposed in Section 4.4, including a lower bound estimate, a tree pruning strategy, and criteria for stopping the A^* algorithm. The experimental results are provided in Section 4.5. Comparisons of our segment-based recognition method with several other studies are presented in Section 4.6. The summary of this chapter is given in Section 4.7.

4.2 State Space Formulation of the Graph Matching

In an approach to problem solving by using state space search, a state space is a representation that consists of nodes and links, where each node denotes a state that is a description sufficient to determine the future, and each link connecting a tail node to a head node denotes a possible one-step transition from one state to another state. The goal state of a state space is where we want to be. The procedure to solve a problem is to find a sequence of transitions that leads from some initial state to the goal state. Now we transform our problem of calculating the distance between two graphs into a state space search problem.

Let $G_i = (N_i, A_i)$ and $G_j = (N_j, A_j)$ be two graphs. The distance from G_i

to G_j, which is defined in Definition 3.7, is

$$\xi(G_i, G_j) = \min\{\beta(f_N, f_A) | (f_N, f_A) \text{ is a matching from } G_i \text{ to } G_j\}.$$

If the arc deletion costs are all set to 0 (see Section 3.4), the matching cost $\beta(f_N, f_A)$ is then expressed by

$$\begin{aligned}\beta(f_N, f_A) &= \sum_{n_i \to n_j \in Q_1} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q_2} \gamma(n_i \to \lambda) + \sum_{\lambda \to n_j \in Q_3} \gamma(\lambda \to n_j) \\ &+ \sum_{(n_i, m_i) \to (n_j, m_j) \in Q_4} \gamma((n_i, m_i) \to (n_j, m_j)) \end{aligned} \tag{4.1}$$

where f_N, f_A and Q_{1-4}, are defined in Definitions 3.5, 3.6 and 3.7, respectively.

Definition 4.1 *Let $G_i = (N_i, A_i)$ and $G_j = (N_j, A_j)$ be two graphs, and $G'_i = (N'_i, A'_i)$ and $G'_j = (N'_j, A'_j)$ be two subgraphs*[2] *of G_i and G_j, respectively. A state is denoted by a set of node correspondences: $S = \{n_i \to n_j | f'_N(n_i) = n_j, n_i \in N'_i \cup \Lambda_i, n_j \in N'_j \cup \Lambda_j\}$, where f'_N is a node mapping from G'_i to G'_j, $N'_i \subseteq N_i$ and $N'_j \subseteq N_j$. When a state covers all the nodes in G_i and G_j, i.e., $N'_i = N_i$ and $N'_j = N_j$, it is called a* **goal state**; *otherwise, a* **middle state**. *The* **initial state** *is a state where there are no any node correspondences. If $S = \{n_i \to n_j | f'_N(n_i) = n_j, n_i \in N'_i \cup \Lambda_i, n_j \in N'_j \cup \Lambda_j\}$ is a middle state, a new state S_1 generated by expanding S is defined by $S_1 = S \cup \{n_{i1} \to n_{j1}\}$, where $n_{i1} \in N_i \cup \Lambda_i, n_{j1} \in N_j \cup \Lambda_j, n_{i1} \notin N'_i, n_{j1} \notin N'_j$ and $n_{i1} \to n_{j1} \neq \lambda \to \lambda$.*

An example of a state space for matching between two graphs is shown in Fig. 4.1. The initial state, middle states and goal states are indicated by "○", "●" and "□", respectively. The state space is actually a tree, so it is also called a state space tree or search tree, and a node[3] of the tree denotes a state.

[2]The definition of an induced subgraph (or subgraph for short) of a graph is given in Definition 3.3. Moreover, we consider that an empty graph is also one of the subgraphs of a graph.

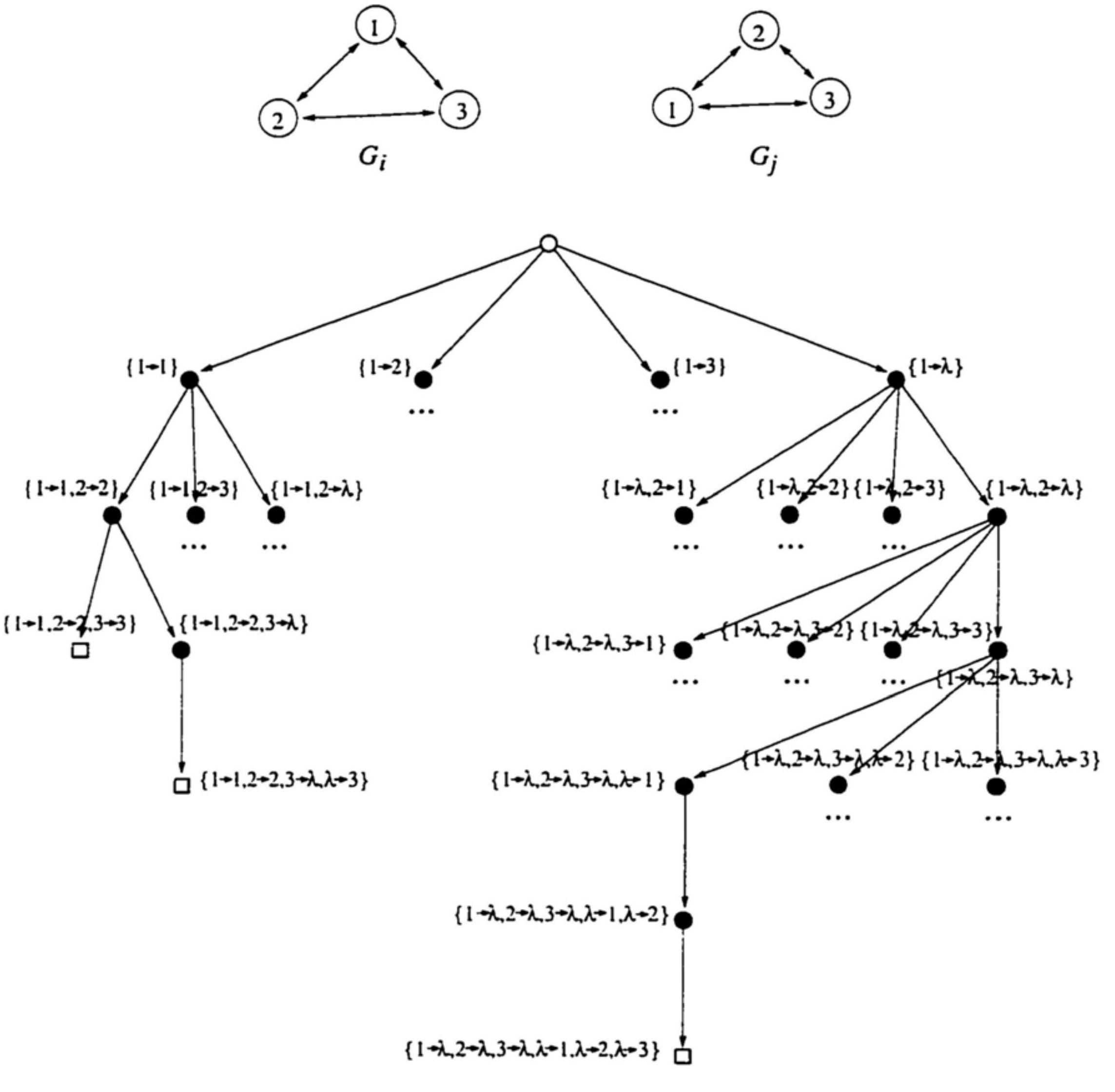


Figure 4.1: A state space tree for matching between G_i and G_j. Symbols "○", "●" and "□" denote the initial state, middle states and goal states, respectively.

From Definition 4.1 and Fig. 4.1, we can see that a goal state corresponds to a matching (f_N, f_A) from G_i to G_j, where f_A is an arc mapping led by f_N (see Definition 3.6); a middle state corresponds to a matching (f'_N, f'_A) from subgraph G'_i of G_i to subgraph G'_j of G_j, where f'_A is an arc mapping led by f'_N.

Definition 4.2 *The cost of a state* $S = \{n_i \to n_j | f'_N(n_i) = n_j, n_i \in N'_i \cup \Lambda_i, n_j \in N'_j \cup \Lambda_j\}$ *that corresponds to the matching* (f'_N, f'_A) *is calculated by*

$$\begin{aligned} \beta(f'_N, f'_A) &= \sum_{n_i \to n_j \in Q'_1} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q'_2} \gamma(n_i \to \lambda) + \sum_{\lambda \to n_j \in Q'_3} \gamma(\lambda \to n_j) \\ &+ \sum_{(n_i, m_i) \to (n_j, m_j) \in Q'_4} \gamma((n_i, m_i) \to (n_j, m_j)) \end{aligned} \tag{4.2}$$

where (f'_N, f'_A) *determines the sets* Q'_{1-4}, *i.e.,* Q'_1 *is the set of* $n_i \to n_j, n_i \in N'_i, n_j \in N'_j$; Q'_2 *the set of* $n_i \to \lambda, n_i \in N'_i$; Q'_3 *the set of* $\lambda \to n_j, n_j \in N'_j$; Q'_4 *the set of* $(n_i, m_i) \to (n_j, m_j), n_i, m_i \in N'_i, n_j, m_j \in N'_j, n_i \neq m_i$;

$\beta(f'_N, f'_A)$ is actually a subgraph matching cost on condition that all arc deletion costs are set to 0. When $(f'_N, f'_A) = (f_N, f_A)$, $\beta(f'_N, f'_A)$ is a graph matching cost. Therefore, we define a **best goal state** in a state space tree as a goal state with the minimum cost among all the goal states. There may be several best goal states in a tree. Now the problem of computing the distance from G_i to G_j is transformed to the problem of finding a best goal state in a state space tree. We will use the heuristic algorithm, A*, to perform the search.

[3] If we say such like "nodes of a graph", "node mapping" and "node correspondences", a node is referred to the node of a graph; otherwise, in this chapter, it stands for a node of a search tree.

4.3 The A* Algorithm

The A* algorithm is a popular heuristic search algorithm in problem solving, whose purpose is to find the cheapest path cost in a network [9, 31, 71]. For our application, it is used to find the best state in a search tree.

Let u be a node in a search tree, $g(u)$ be the cost of the path from the initial node to u, $h^*(u)$ be the minimal cost of a path from u to a goal node or a best goal node, and $h(u)$ be an estimate of $h^*(u)$. Note that there is only a path from the initial node to another node in the tree. Let the state of node u corresponds to a matching (f'_{Nu}, f'_{Au}). Thus we define

$$g(u) = \beta(f'_{Nu}, f'_{Au}). \tag{4.3}$$

Let $V = \{v\}$ be the set of the goal nodes to which there exist paths from u, and let

$$g(v') = \min_{v \in V}\{g(v)\}. \tag{4.4}$$

$h^*(u)$ is then given by

$$h^*(u) = g(v') - g(u). \tag{4.5}$$

The evaluation function of the A* is defined as

$$f(u) = g(u) + h(u). \tag{4.6}$$

It is a cost estimate of the minimal cost path constrained through node u. We call $f(u)$ the estimated value of node u, and h a heuristic function. The computation of $h(u)$ is according to some problem-dependent information. $h(u) \geq 0$ is always assumed. The A* algorithm for our tree search problem is presented below.

The A* algorithm

Step 1. Create a search tree, consisting of only the initial node. Put it on a list called $OPEN$.

Step 2. If the first node on $OPEN$ is a goal node, exit with the estimated value and the state of the goal node.

Step 3. Remove the first node from $OPEN$. Expand it, generating the set of its successors. Calculate the estimated values of the successors using (4.6). Add these nodes to $OPEN$, and arrange these new nodes and the old nodes on $OPEN$ in increasing order of their estimated values. (Thus the node having the smallest estimated value is at the first.)

Step 4. Go to Step 2.

The above algorithm is a special version of the A* for a search graph [31, 71]. The A* algorithm is always convergent for finite search graphs [71]. Two definitions and two theorems in [71] are given in the following.

Definition 4.3 *A search algorithm is called* **admissible** *if, for any search graph, it always terminates in an optimal path from an initial node to a goal node whenever a path from the initial node to a goal node exists.*

For a search tree in our application, the goal node in an optimal path is one of the best goal nodes we search for.

Theorem 4.1 *The A^* algorithm is admissible if, for every node u of a search graph being examined, the following inequality is satisfied:*

$$h(u) \leq h^*(u). \tag{4.7}$$

Definition 4.4 *A heuristic function h is* **monotonic** *if, for every node u and any of its successors w,*

$$h(u) - h(w) \leq c(u, w) \tag{4.8}$$

with

$$h(v) = 0, \tag{4.9}$$

where v is any goal node, and $c(u, w)$ is the path cost from u to w.

Theorem 4.2 *If the monotone restriction is satisfied, the estimated f values of the sequence of nodes expanded by the A^* is nondecreasing.*

In our tree search problem, $c(u, w) = g(w) - g(u)$. Theorem 4.1 guarantees that the A* algorithm can find the best node if (4.7) is satisfied. Theorem 4.2 is useful for reducing computational time when the A* is used in our on-line Chinese character recognition, which will be described in Section 4.4.3.

The search effectiveness of the A* algorithm relies heavily on how precise the estimate of h^* is. If $h(u) = h^*(u)$, the fewest nodes are expanded. Setting $h(u) \equiv 0$ assure admissibility but results in an inefficient breadth-first search. Although great efforts have been made to find good heuristic functions, in general, precise estimates of $h^*(u)$ are quite difficult for most applications [9, 44, 71, 76, 77, 78, 89, 90, 101], and thus the A* algorithm has exponential complexity [9, 104, 105]. From a probabilistic point of view, Pearl has made a thorough study about the relations between the precision of the heuristic estimates and the average complexity of the A* in [77].

4.4 Schemes for Speeding up the A* Algorithm

As mentioned above, the search efficiency of the A* algorithm is not satisfactory (or is even pessimistic). However, the schemes proposed in this section, which do not guarantee that the A* always finds optimal solutions, can greatly speed up the A* when it is used in our on-line Chinese character recognition.

4.4.1 A Lower Bound Estimate

The use of problem-dependent heuristic information, which is represented by the estimate function h, will make the A* expand fewer nodes than a search with $h \equiv 0$.

From Definition 4.2 and (4.3), we know that the cost of a node in a search tree is equal to a subgraph matching cost. Let a middle node be u, and two graphs under matching be $G_i = (N_i, A_i)$ and $G_j = (N_j, A_j)$. Let the state of u correspond to a matching (f'_{Nu}, f'_{Au}) from subgraph $G'_i = (N'_i, A'_i)$ of G_i to subgraph $G'_j = (N'_j, A'_j)$ of G_j. We use the costs of node correspondences from subgraph G''_i to subgraph G''_j as an estimate $h(u)$ of $h^*(u)$, where $G''_i = (N''_i, A''_i), N''_i = N_i \setminus N'_i$ ($G''_j = (N''_j, A''_j), N''_j = N_j \setminus N'_j$, respectively) is the subgraph of G_i (G_j, respectively) by deleting the nodes of G'_i from G_i (G'_j from G_j, respectively) and deleting all the arcs connecting these nodes. $h(u)$ is expressed by

$$\begin{aligned} h(u) &= \min_{f''_{Nu}} \left\{ \sum_{n_i \to n_j \in Q''_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(u)} \gamma(n_i \to \lambda) \right. \\ &\quad \left. + \sum_{\lambda \to n_j \in Q''_3(u)} \gamma(\lambda \to n_j) \right\}, \end{aligned} \tag{4.10}$$

where f''_{Nu} is a node mapping from G''_i to G''_j and it determines the sets $Q''_{1-3}(u)$: $Q''_1(u)$ is the set of $n_i \rightarrow n_j, n_i \in N''_i, n_j \in N''_j$; $Q''_2(u)$ the set of $n_i \rightarrow \lambda, n_i \in N''_i$; $Q''_3(u)$ the set of $\lambda \rightarrow n_j, n_j \in N''_j$.

Theorem 4.3 *Let u be a node in a search tree. $h(u)$ defined in (4.10) is a lower bound on $h^*(u)$ defined in (4.5), i.e., $h(u) \leq h^*(u)$.*

Proof. Let $V = \{v\}$ be the goal nodes to which there are paths from node u. By (4.4) and (4.5), we have

$$\begin{aligned} h^*(u) &= \min_{v \in V}\{g(v)\} - g(u) \\ &= \min_{v \in V}\left\{ \sum_{n_i \rightarrow n_j \in Q_1(v)} \gamma(n_i \rightarrow n_j) + \sum_{n_i \rightarrow \lambda \in Q_2(v)} \gamma(n_i \rightarrow \lambda) \right. \\ &\quad \left. + \sum_{\lambda \rightarrow n_j \in Q_3(v)} \gamma(\lambda \rightarrow n_j) + \sum_{(n_i, m_i) \rightarrow (n_j, m_j) \in Q_4(v)} \gamma((n_i, m_i) \rightarrow (n_j, m_j)) \right\} \\ &\quad - \left\{ \sum_{n_i \rightarrow n_j \in Q'_1(u)} \gamma(n_i \rightarrow n_j) + \sum_{n_i \rightarrow \lambda \in Q'_2(u)} \gamma(n_i \rightarrow \lambda) \right. \\ &\quad \left. + \sum_{\lambda \rightarrow n_j \in Q'_3(u)} \gamma(\lambda \rightarrow n_j) + \sum_{(n_i, m_i) \rightarrow (n_j, m_j) \in Q'_4(u)} \gamma((n_i, m_i) \rightarrow (n_j, m_j)) \right\}, \end{aligned}$$

where $Q_{1-4}(v)$ are defined in Definition 3.7, and $Q'_{1-4}(u)$ are defined in Definition 4.2. Let us look at Fig. 4.2. There is only one path from the initial node to node u and all the paths from the initial node to nodes in V go through u. Therefore

$$Q'_k(u) \subseteq Q_k(v), \quad k = 1, 2, 3, 4.$$

Therefore

$$h^*(u) = \min_{v \in V}\left\{ \sum_{n_i \rightarrow n_j \in Q_1(v) \backslash Q'_1(u)} \gamma(n_i \rightarrow n_j) + \sum_{n_i \rightarrow \lambda \in Q_2(v) \backslash Q'_2(u)} \gamma(n_i \rightarrow \lambda) \right.$$

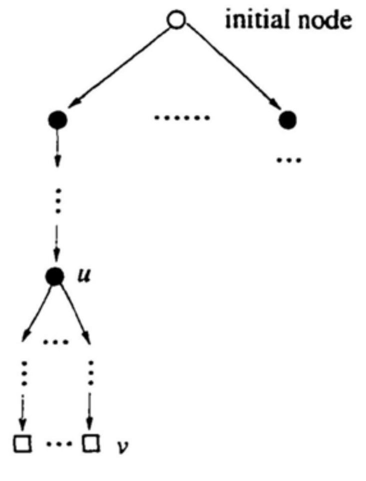


Figure 4.2: A search tree. u is a middle node and v is a goal node that can be reached from u.

$$+ \sum_{\lambda \to n_j \in Q_3(v) \setminus Q'_3(u)} \gamma(\lambda \to n_j)$$

$$\left. + \sum_{(n_i,m_i) \to (n_j,m_j) \in Q_4(v) \setminus Q'_4(u)} \gamma((n_i,m_i) \to (n_j,m_j)) \right\}. \tag{4.11}$$

Since $\gamma((n_i,m_i) \to (n_j,m_j)) \geq 0$, we have

$$\begin{aligned} h^*(u) \geq \min_{v \in V} \Bigg\{ & \sum_{n_i \to n_j \in Q_1(v) \setminus Q'_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q_2(v) \setminus Q'_2(u)} \gamma(n_i \to \lambda) \\ & + \sum_{\lambda \to n_j \in Q_3(v) \setminus Q'_3(u)} \gamma(\lambda \to n_j) \Bigg\} \end{aligned} \tag{4.12}$$

The right term in above inequality is an alternative expression of $h(u)$ in (4.10). Thus the theorem follows. □

Theorem 4.4 *The heuristic function h defined in (4.10) is monotonic.*

Proof.

Case 1. Let u be a middle node in a search tree and w be one of its successors generated by expanding u, as shown in Fig. 4.3. Let the states of u

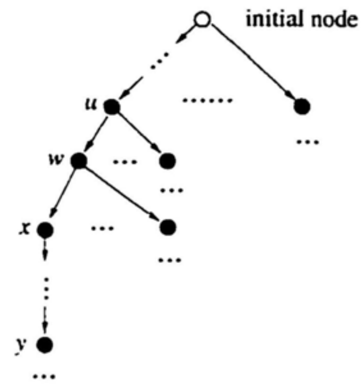


Figure 4.3: A search tree. u is a middle node. w, x and y are successors of u.

and w be $S(u)$ and $S(w)$, respectively. Then by definition 4.1, we have $S(w) = S(u) \cup \{n_{ia} \to n_{ja}\}$. Therefore

$$\begin{aligned}
c(u,w) &= g(w) - g(u) \\
&= \beta(f'_{Nw}, f'_{Aw}) - \beta(f'_{Nu}, f'_{Au}) \\
&= \left\{ \sum_{n_i \to n_j \in Q'_1(w)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q'_2(w)} \gamma(n_i \to \lambda) \right. \\
&\quad \left. + \sum_{\lambda \to n_j \in Q'_3(w)} \gamma(\lambda \to n_j) + \sum_{(n_i,m_i)\to(n_j,m_j) \in Q'_4(w)} \gamma((n_i,m_i) \to (n_j,m_j)) \right\} \\
&\quad - \left\{ \sum_{n_i \to n_j \in Q'_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q'_2(u)} \gamma(n_i \to \lambda) \right. \\
&\quad \left. + \sum_{\lambda \to n_j \in Q'_3(u)} \gamma(\lambda \to n_j) + \sum_{(n_i,m_i)\to(n_j,m_j) \in Q'_4(u)} \gamma((n_i,m_i) \to (n_j,m_j)) \right\}, \\
&= \sum_{n_i \to n_j \in Q'_1(w)\setminus Q'_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q'_2(w)\setminus Q'_2(u)} \gamma(n_i \to \lambda) \\
&\quad + \sum_{\lambda \to n_j \in Q'_3(w)\setminus Q'_3(u)} \gamma(\lambda \to n_j) \\
&\quad + \sum_{(n_i,m_i)\to(n_j,m_j) \in Q'_4(w)\setminus Q'_4(u)} \gamma((n_i,m_i) \to (n_j,m_j)) \qquad (4.13) \\
&\geq \sum_{n_i \to n_j \in Q'_1(w)\setminus Q'_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q'_2(w)\setminus Q'_2(u)} \gamma(n_i \to \lambda)
\end{aligned}$$

$$+ \sum_{\lambda \to n_j \in Q'_3(w) \setminus Q'_3(u)} \gamma(\lambda \to n_j) \tag{4.14}$$

$$= \gamma(n_{ia} \to n_{ja}). \tag{4.15}$$

By definition, we have

$$\begin{aligned} h(u) &= \min_{f''_{Nu}} \Bigg\{ \sum_{n_i \to n_j \in Q''_1(u)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(u)} \gamma(n_i \to \lambda) \\ &\quad + \sum_{\lambda \to n_j \in Q''_3(u)} \gamma(\lambda \to n_j) \Bigg\}, \\ h(w) &= \min_{f''_{Nw}} \Bigg\{ \sum_{n_i \to n_j \in Q''_1(w)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(w)} \gamma(n_i \to \lambda) \\ &\quad + \sum_{\lambda \to n_j \in Q''_3(w)} \gamma(\lambda \to n_j) \Bigg\}, \end{aligned}$$

where f''_{Nu} is a node mapping from $G''_{iu} = (N''_{iu}, A''_{iu})$ to $G''_{ju} = (N''_{ju}, A''_{ju})$ and f''_{Nw} is a node mapping from $G''_{iw} = (N''_{iw}, A''_{iw})$ to $G''_{jw} = (N''_{jw}, A''_{jw})$ (see the definition of $h(u)$ in (4.10) for more detailed description).

Suppose f''_{Nw*} is the optimal node mapping from G''_{iw} to G''_{jw} such that

$$\begin{aligned} h(w) &= \sum_{n_i \to n_j \in Q''_1(w*)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(w*)} \gamma(n_i \to \lambda) \\ &\quad + \sum_{\lambda \to n_j \in Q''_3(w*)} \gamma(\lambda \to n_j), \end{aligned}$$

where $Q''_{1-3}(w*)$ are determined by f''_{Nw*}. If $n_{ia} \in N''_{iu}$ and $n_{ja} \in N''_{ju}$, then $N''_{iu} = N''_{iw} \cup \{n_{ia}\}$ and $N''_{ju} = N''_{jw} \cup \{n_{ja}\}$. Therefore, we can find a node mapping f''_{Nu*} from G''_{iu} to G''_{ju} such that

$$\begin{aligned} &\sum_{n_i \to n_j \in Q''_1(u*)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(u*)} \gamma(n_i \to \lambda) + \sum_{\lambda \to n_j \in Q''_3(u*)} \gamma(\lambda \to n_j) \\ &\quad = \gamma(n_{ia} \to n_{ja}) + \sum_{n_i \to n_j \in Q''_1(w*)} \gamma(n_i \to n_j) + \\ &\qquad \sum_{n_i \to \lambda \in Q''_2(w*)} \gamma(n_i \to \lambda) + \sum_{\lambda \to n_j \in Q''_3(w*)} \gamma(\lambda \to n_j), \end{aligned}$$

where $Q''_{1-3}(u*)$ are determined by f''_{Nu*}, which suggests

$$\begin{aligned} h(u) &\leq \sum_{n_i \to n_j \in Q''_1(u*)} \gamma(n_i \to n_j) + \sum_{n_i \to \lambda \in Q''_2(u*)} \gamma(n_i \to \lambda) \\ &\quad + \sum_{\lambda \to n_j \in Q''_3(u*)} \gamma(\lambda \to n_j) \\ &= \gamma(n_{ia} \to n_{ja}) + h(w). \end{aligned}$$

Thus, it follows that

$$\begin{aligned} h(u) - h(w) &\leq \gamma(n_{ia} \to n_{ja}) && (4.16) \\ &\leq c(u, w). && (4.17) \end{aligned}$$

The same result can be obtained if $n_{ia} \in N''_{iu}$ and $n_{ja} = \lambda$ or if $n_{ia} = \lambda$ and $n_{ja} \in N''_{ju}$.

Case 2. Let w and x be successors of u, where w is generated by expanding u and x is generated by expanding w (see Fig. 4.3). Then we have

$$h(u) - h(w) \leq c(u, w),$$

$$h(w) - h(x) \leq c(w, x),$$

and further

$$\begin{aligned} h(u) - h(x) &\leq c(u, w) + c(w, x) \\ &= g(w) - g(u) + g(x) - g(w) \\ &= g(x) - g(u) \\ &= c(u, x). \end{aligned}$$

The same conclusion can be obtained when y is any successor of u. Finally, for any goal node v, by (4.10), it is clear that $h(v) = 0$. Therefore the theorem follows. $\square$

Nilsson showed that, if $h_1(u) \le h^*(u)$, $h_2(u) \le h^*(u)$ and $h_1(u) \le h_2(u)$, then the A_1^* using $h_1(u)$ expands at least as many nodes as does the A_2^* using $h_2(u)$ [71]. From (4.10) we know that only the costs of node correspondences from G_i'' to G_j'' are employed to estimate $h^*(u)$. The costs of arc correspondences from G_i'' to G_j'' are not considered. Such a scheme results in a relatively simple calculation of $h(u)$, but the heuristic power of $h(u)$ is not good enough. If the costs of arc correspondences are also used to estimate $h^*(u)$, the calculation of of the estimate might become another NP-complete graph matching problem, which is also time-consuming.

Even though we use only the costs of node correspondences to estimate $h^*(u)$, the calculation of $h(u)$ is not trivial. The optimization problem in (4.10) can be transformed to a minimum weight matching problem in a weighted bipartite graph [75].

A graph $B = (V_1 \cup V_2, E)$ with vertex set $V_1 \cup V_2$ and edge set E is called a bipartite graph if all its nodes can be partitioned into two subsets, V_1 and V_2, such that every edge in the graph connects some node in V_1 to some node in V_2. If a bipartite graph has weights associated with its edges, it is called a weighted bipartite graph. A **matching**[4] in B is a subset of edge $K \subset E$ such that no two edges of K are adjacent. Here we use the terms "vertex" and "edge" for bipartite graphs instead of "node" and "arc" to avoid possible confusion.

Now we consider an example. Let f''_{Nu} be a node mapping from $G_i'' = (N_i'', A_i'')$ to $G_j'' = (N_j'', A_j'')$ (see (4.10)). Suppose $N_i'' = \{n_{i1}, n_{i2}, n_{i3}\}$ and

[4]Note the difference between a matching defined here and the term "matching" we use previously. The former denoting a matching in a bipartite graph is a conventional term in graph theory while the latter defined in Definition 3.7 represents a node mapping and an arc mapping from a complete relational graph to another complete relational graph.

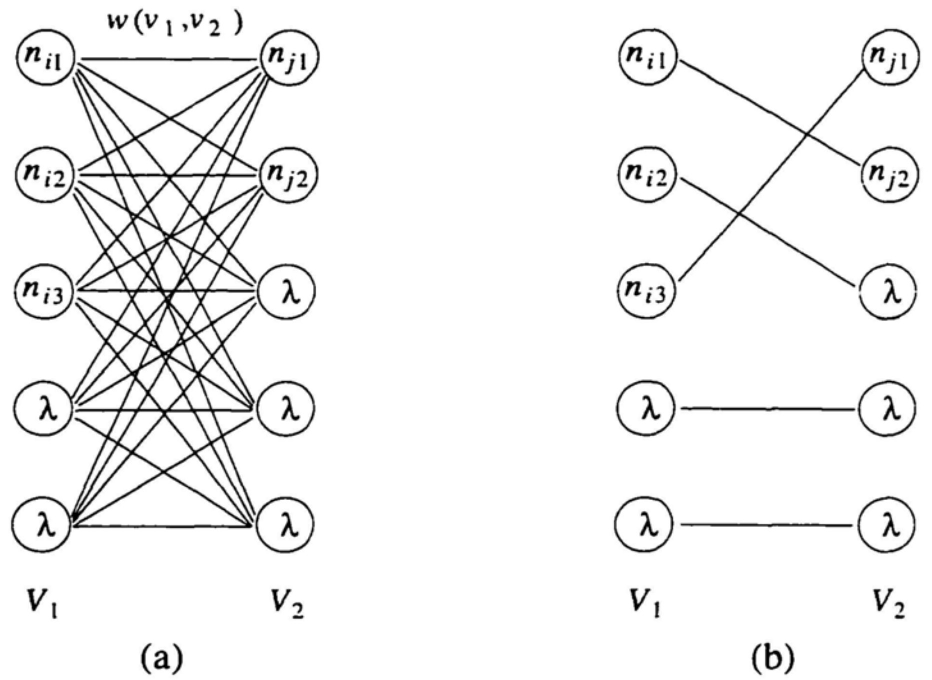


Figure 4.4: (a) A bipartite graph $B = (V_1 \cup V_2, E)$, where $V_1 = N_i'' \cup \Lambda_i$, $|\Lambda_i| = |N_j''|$, $V_2 = N_j'' \cup \Lambda_j$, and $|\Lambda_j| = |N_i''|$. (b) A matching in the bipartite graph.

$N_j'' = \{n_{j1}, n_{j2}\}$. All possible node mappings can be represented by the matchings of the weighted bipartite graph shown in Fig. 4.4(a), where $V_1 = N_i'' \cup \Lambda_i$, $|\Lambda_i| = |N_j''|$, $V_2 = N_j'' \cup \Lambda_j$, and $|\Lambda_j| = |N_i''|$. A vertex in V_1 is connected to all the vertexes in V_2. We denote by $e(v_1, v_2)$ an edge connecting $v_1 \in V_1$ and $v_2 \in V_2$. A weight associated with $e(v_1, v_2)$ is defined as the cost of node correspondence $\gamma(v_1 \rightarrow v_2)$ when $e(v_1, v_2) \neq e(\lambda, \lambda)$ and as 0 when $e(v_1, v_2) = e(\lambda, \lambda)$. Now the calculation of $h(u)$ in (4.10) is equivalent to the problem of finding a matching in B such that the sum of the weights associated with the edges of the matching is the smallest. The polynomial-time Hungarian method with complexity $O(|V_1|^3)$ $(= O((|N_1''| + |N_2''|)^3))$ is a solution to this problem [75]. A matching in B of

Fig. 4.4(a) is shown in Fig. 4.4(b), which corresponds to such a node mapping: $\{n_{i1} \to n_{j2}, n_{i2} \to \lambda, n_{i3} \to n_{j1}\}$.

The effort to computer $h(u)$ is one of the important factors that influence the search efficiency of the A* [71]. In our experiment, we found that if for every generated node in a search tree, the Hungarian method with complexity $O((|N_1''| + |N_2''|)^3)$ is used to calculate $h(u)$, the speed of the A* is too slow to accept. Therefore, we propose the following greedy algorithm to approximately calculate $h(u)$.

Greedy algorithm for calculating $h(u)$

Input: Node sets N_i'' and N_j''. (comment: suppose $|N_i''| \leq |N_j''|$)

Output: An approximate value of h.

begin

  $h := 0$;

  while $N_i'' \neq \emptyset$ **do**

    begin

      choose a node n_i in N_i'';

      remove n_i from N_i'';

      find a node n_j in N_j'' such that

$$\gamma(n_i \to n_j) = \min_{n_k \in N_j''} \{\gamma(n_i \to n_k)\};$$

      if $\gamma(n_i \to n_j) \leq \gamma(n_i \to \lambda)$ **then**

        begin

          $h := h + \gamma(n_i \to n_j)$;

remove n_j from N_j'';

end

else

$h := h + \gamma(n_i \to \lambda)$;

end

while $N_j'' \neq \emptyset$ **do**

begin

choose a node n_l in N_j'';

$h := h + \gamma(\lambda \to n_l)$;

remove n_l from N_j'' :

end

end

It is not difficult to analyze the running time of the greedy algorithm. Assuming $|N_i''| \leq |N_j''|$. The main computation is, for every remaining node n_i in N_i'', to find a remaining node n_j in N_j'' such that $\gamma(n_i \to n_j) = \min_{n_k \in N_j''}\{\gamma(n_i \to n_k)\}$. This requires $O(|N_i''| \cdot |N_j''|)$ time, which is also the complexity of the algorithm.

Remark. Let us denote the estimate of $h^*(u)$ by $h'(u)$ obtained with the greedy algorithm. Since $h'(u)$ is an approximate value of $h(u)$ defined in (4.10), we have $h'(u) \geq h(u)$. Theoretically, there exists a possibility that $h'(u)$ is not a lower bound on $h^*(u)$ when $h'(u) > h(u)$. However, by (4.11) and (4.12) in the proof of Theorem 4.3, we have

$$h^*(u) = h(u) + \sum_{(n_i,m_i)\to(n_j,m_j)\in Q_4(v)\setminus Q_4'(u)} \gamma((n_i, m_i) \to (n_j, m_j)),$$

where the last term is the sum of some arc correspondence costs. Because of this term, we observed in our experiments that $h'(u) \leq h^*(u)$ holds for almost all nodes u in a search tree. Even if $h'(u) > h^*(u)$ for some nodes u, it is not meant that the A* must not find the best goal node. Nilsson has pointed out that heuristic power can often be gained at the expense of admissibility by using some function for h that is not a lower bound on h^* [71].

4.4.2 A Tree Pruning Strategy

In Section 4.5, we will give some experimental results to show that the A* algorithm using only the evaluation function $f(u) = g(u) + h(u)$ to guide its search is too slow in on-line Chinese character recognition. To increase the search efficiency of the A*, we propose a tree pruning strategy that imposes geometric position constraints on strokes of Chinese characters and thus avoids expanding lots of nodes that are very impossible to be located in the optimal path from the initial node to the best goal node in a search tree.

Let us look at the two characters shown in Fig. 4.5. When one is asked to identify between the two characters the stroke pairs that he/she considers compatible, it is easy for he/she to get the correct answer: $\{1 \rightarrow 2, 2 \rightarrow 3, 3 \rightarrow$

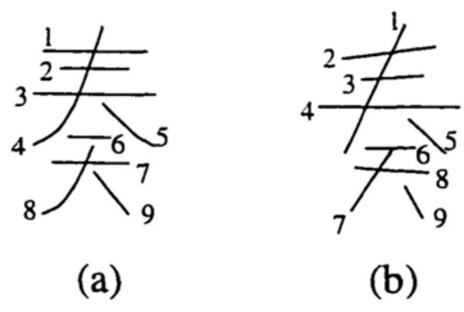


Figure 4.5: A model character (a) and its handwritten version (b).

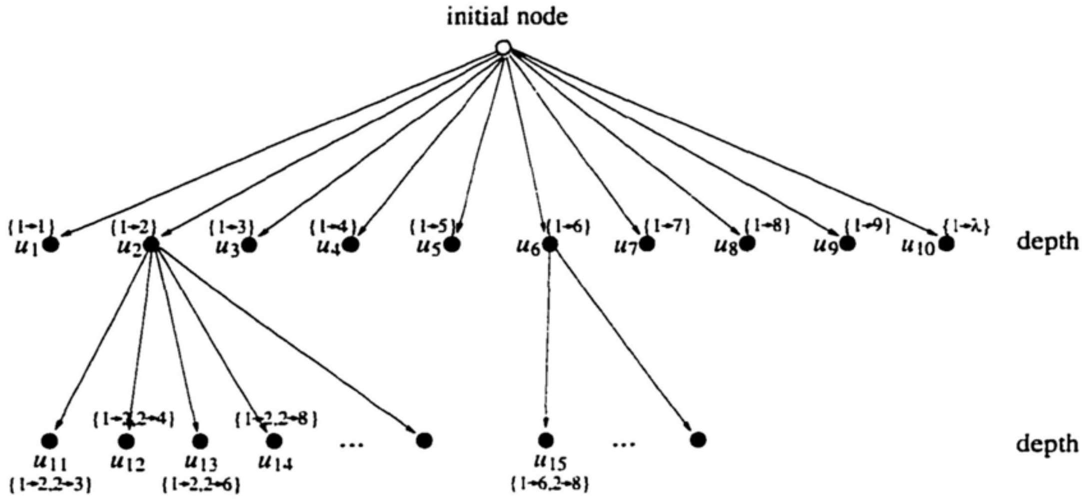


Figure 4.6: A partial tree for matching from the character in Fig. 4.5(a) to the character in Fig. 4.5(b).

$4, 4 \to 1, 5 \to 5, 6 \to 6, 7 \to 8, 8 \to 7, 9 \to 9\}$. The reason why we can find the answer easily and quickly is that we have a bird's eye view of the two characters. A tree search is hard just because a search algorithm does not have this function.

Consider the partial search tree (Fig. 4.6) for matching from the graph of the character in Fig. 4.5(a) to the graph of the character in Fig. 4.5(b). The states of the nodes are shown near the nodes, which represent corresponding stroke correspondences. At depth 1, nodes u_{1-10} are generated by expanding the initial node. Determining which node will be expanded next is according to the estimated f values of the nodes. For this example, the A* algorithm will find

$$f(u_2) = f(u_3) = f(u_4) = f(u_6) = f(u_8).$$

Thus every node in the set $\{u_2, u_3, u_4, u_6, u_8\}$ may be a candidate to be expanded. Similarly, at depth 2, the calculated estimated values $f(u_{11})$, $f(u_{12})$, $f(u_{13})$, $f(u_{14})$, and $f(u_{15})$ are all the same. Node u_{11} that is in the optimal path has

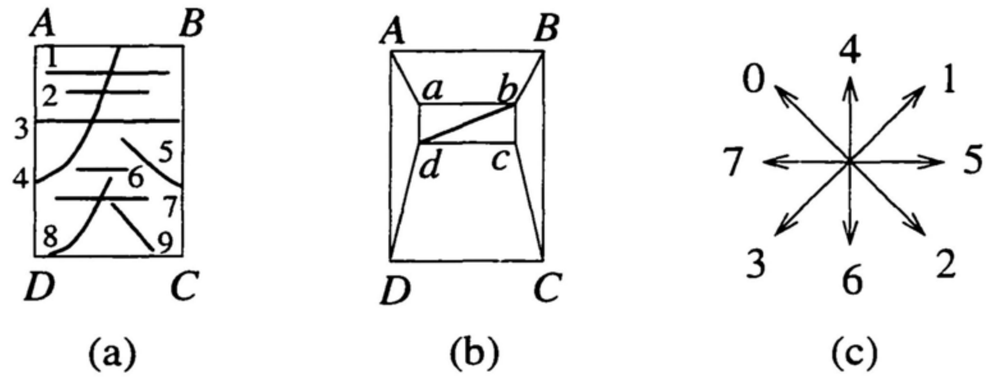


Figure 4.7: (a) A Chinese character and the smallest rectangle $ABCD$ surrounding the character. (b) Geometric illustration of $D_{0-3}(i)$. (c) 8 directions.

no priority over nodes u_{12-15} in being expanded first. We have observed that in the matching between a model character and its handwritten character having stroke type and relation deformations, the A* tends to spend much time to discriminate among the paths whose costs do not vary significantly, and thus it is very possible that all nodes at depth 1 are expanded in order to find the optimal path. However, glancing at Fig. 4.5, we are easy to know only u_2 and u_{11} in Fig. 4.6 are located in the optimal path. The objective of the pruning strategy presented below is to add more or less the function of this bird's eye view into the A* algorithm.

In on-line recognition, an input Chinese character as a whole can be regarded as no rotation variety. Hence a lot of information about the geometric positions of strokes of the character may be used to assist the A* algorithm when searching. Fig. 4.7(a) shows a Chinese character and the smallest rectangle $ABCD$ that surrounds it. What are the stable geometric position features of the strokes of the character in daily handwriting? Intuitively, strokes 1 and 2 are written near the upper side of the $ABCD$; strokes 3–7 in the middle; stroke 8 near the lower side or the lower-left corner; stroke 9 near the lower side or the lower-right corner. Now we formulate these character- and stroke-dependent features in the following.

Let *abcd* be the smallest rectangle that surrounds stroke j of an input character with t strokes (Fig. 4.7(b)). Eight directions shown in Fig. 4.7(c) are used to denote the directions of eight distances $D_{0-7}(j)$, where $D_{0-3}(j)$ are distances from a to A, b to B, c to C, and d to D, respectively, as shown in Fig. 4.7(b), and $D_{4-7}(j)$ are the distances from the geometric center of *abcd* to the respective four sides of the $ABCD$. A notation $od(D_q(j))$, $q \in \{0, 1, ..., 7\}$, is used to denote that $D_q(j)$ is the $od(D_q(j))$-th smallest distance among $\{D_q(1), D_q(2), ..., D_q(t)\}$. For example, $od(D_q(j)) = 1$ means that the distance $D_q(j)$ of stroke j is the smallest, and $od(D_q(j)) = m, m \leq t$, means that there are $m - 1$ distances among $\{D_q(1), D_q(2), ..., D_q(t)\}$ which are smaller than $D_q(j)$.

Definition 4.5 *The geometric position features (GPF) of the strokes of a model Chinese character with s strokes is defined as a set of s 3-tuples*

$$GPF = \{(d_i, x_i, y_i) | i = 1, 2, ..., s\}, \tag{4.18}$$

where $d_i \in \{0, 1, ..., 7\}$ *denotes one of the 8 directions, and* x_i *and* y_i *are two end points of the integer interval* $[x_i, y_i]$, $x_i, y_i \in \{1, 2, ..., s\}$. (d_i, x_i, y_i) *gives a geometric position constraint on input strokes in the sense that only the input strokes j's, satisfying* $x_i \leq od(D_{d_i}(j)) \leq y_i$, *can have the chance of being as the node (stroke) correspondences* $i \to j$ *or* $j \to i$ *in matching. A model stroke i and an input stroke j are called compatible if* $x_i \leq od(D_{d_i}(j)) \leq y_i$.

An example is given for better understanding of the GPF. For the character shown in Fig. 4.7(a), a possible assignment of its GPF is

$$\begin{aligned} GPF &= \{(4,1,2),(4,2,4),(4,3,6),(0,1,4),(4,4,8), \\ &\quad (6,4,7),(6,2,5),(6,1,3),(6,1,2)\}. \end{aligned} \tag{4.19}$$

Here $(d_1, x_1, y_1) = (4, 1, 2)$ denotes that the two input strokes j_1 and j_2 written at the top of an input character and satisfying $1 \leq od(D_4(j_k)) \leq 2$, $k = 1, 2$, are compatible with stroke 1 of the model character.

Now, for each model Chinese character, in addition to the features of its stroke number, stroke types and relation matrix, a new feature GPF is added. For an input character with t strokes, we not only recognize its t stroke types, find the relations between any two strokes, and create its relational graph, but also calculate its $od(D_q(j)), j = 1, 2, ..., t; q = 0, 1, ..., 7$.

The GPF can be used to prune the search tree efficiently. When searching the tree for the optimal matching from graph $G_i = (N_i, A_i)$ of a model character to graph $G_j = (N_j, A_j)$ of an input character, the A* algorithm runs with a pruning operation inserted. The generation of a successor node in the tree means that besides the node mapping from subgraph $G_i' = (N_i', A_i')$ of G_i to subgraph $G_j' = (N_j', A_j')$ of G_j in its father node in the tree, a new node (stroke) correspondence $i \to j$ is yielded, where $i \in (N_i \setminus N_i') \cup \Lambda_i$, $j \in (N_j \setminus N_j') \cup \Lambda_j$, and $i \to j \neq \lambda \to \lambda$. Now suppose $i \neq \lambda$ and $j \neq \lambda$. Let the ith element of the GPF of the model character be (d_i, x_i, y_i). If $x_i \leq od(D_{d_i}(j)) \leq y_i$, i.e., the stroke i is compatible with stroke j, then the newly-generated node will be put on the $OPEN$ of the A*; otherwise, the node is pruned away.

For the partial search tree in Fig. 4.6, with the GPF in (4.19), since $(d_1, x_1, y_1) = (4, 1, 2)$ and $(d_2, x_2, y_2) = (4, 2, 4)$, the tree after pruning will become a much

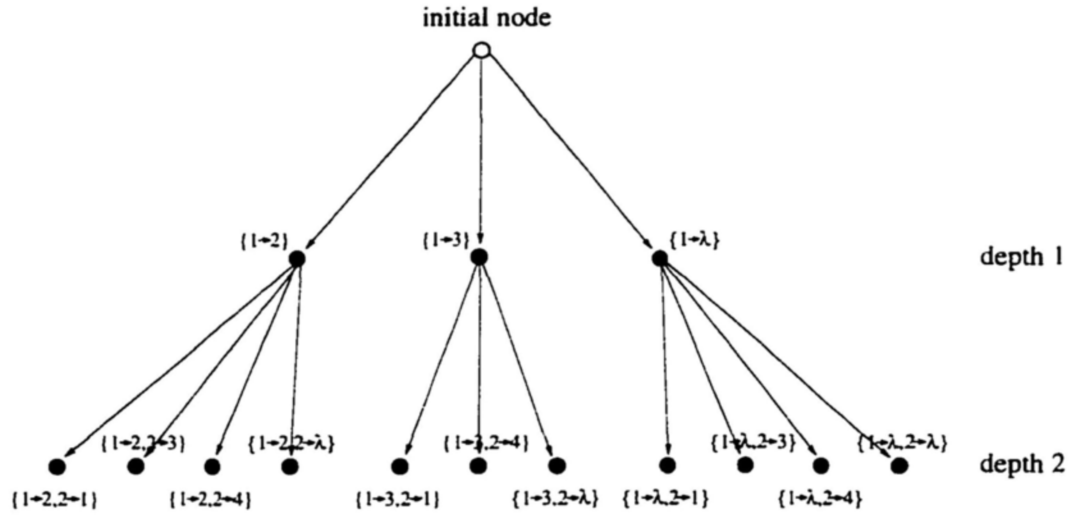


Figure 4.8: A partial tree obtained by pruning the tree in Fig. 4.6. All possible nodes at depths 1 and 2 are shown.

smaller tree as shown in Fig. 4.8, where all the nodes at depths 1 and 2 are given. As can be seen, only three nodes at depth 1 have the chance to be expanded but ten nodes do before pruning. Also expanding a node at depth 1 in Fig. 4.8 generates three or four successors, while expanding $u_i, i \in \{1, 2, ..., 9\}$ in Fig. 4.6 yields nine successors and expanding u_{10} yields ten. Comparing the two trees, we see that the nodes u_6 and u_{15} in Fig. 4.6, which are very possible to be expanded but are impossible to be located in the optimal path from a human view, now no longer appear on the $OPEN$ of the A*. In this sense, the A* has more or less the function of a bird's eye view.

The above content of this section discusses the tree pruning strategy for the stroke-based graph matching problem, where the GPF is the geometric position features of strokes of model characters and is used to impose constraints on input strokes. The same idea is also suitable for dealing with the segment-based graph matching problem. But the GPF is the geometric position features of segments of model characters and is used to impose constraints on segments of input characters. For the character shown in Fig. 4.7(a), its GPF in (4.19) can be used in both the stroke-based Chinese character recognition and the segment-based Chinese character recognition since the strokes of the character are also the segments of the characters. By the way, it is unnecessary for the Chinese characters with less than five strokes (or segments) to have the GPF. The search trees used for matching between these characters are small.

Remark. The construction of GPF for each model is based on the human knowledge of stroke (segment) structure of Chinese characters. Obviously, it is not unique and is character- and stroke-dependent. We hope that for the $GPF = \{(d_i, x_i, y_i) | i = 1, 2, ..., s\}$ of a model character with s strokes, intervals $[x_i, y_i], i = 1, 2, ..., s$, are designed as small as possible. In the case of $x_i = y_i, i = 1, 2, ..., s$, the fastest search is obtained. However, considering wide stroke variations in daily handwriting, the GPF of a character should be designed carefully. Too strict constraints on stroke positions may result in incorrect recognition. Therefore, in the construction of GPF, we prefer to give looser constraints to facilitate higher recognition rate.

4.4.3 Criteria for Stopping the A* Algorithm

The A* algorithm will stop if it finds a best goal node in a search tree. The best goal node corresponds to the optimal matching between two graphs. A fact in Chinese character recognition is that the number of model characters similar to an input character are much less than that of the other characters. It is unnecessary to obtain the final optimal matching between two characters (in other words, we may terminate the A* before it reaches the best goal node) if we know they are dissimilar while searching. Here, "two similar characters" means that the distance between the graphs of the two characters is relatively small. On the contrary, the distance between the graphs of two dissimilar characters are expected to be large. Theorems 4.2 and 4.4 can help reduce much search time.

From Theorem 4.2 we know that if a heuristic function h satisfies the monotone restriction, the estimated f values of the sequence of nodes expanded by the A* is nondecreasing. The estimated value of the best goal node equals the distance for which the A* searches. Theorem 4.4 states that the heuristic function h in (4.10) is monotonic. However, before giving the criteria for stopping the A*, we would like to clarify whether the approximate values $h'(u)$ of $h(u)$ obtained with the greedy algorithm presented in Section 4.4.1 is also monotonic.

Let u be a node in a search tree and w be a successor of u generated by expanding u. Let $h'(u)$ be the estimate of $h^*(u)$ and $h'(w)$ be the estimate of $h^*(w)$, obtained by the greedy algorithm. By (4.13)–(4.17), we have

$$c(u,w) = \gamma(n_{ia} \to n_{ja}) + \sum_{(n_i,m_i)\to(n_j,m_j)\in Q_4'(w)\setminus Q_4'(u)} \gamma((n_i,m_i) \to (n_j,m_j)), \tag{4.20}$$

and

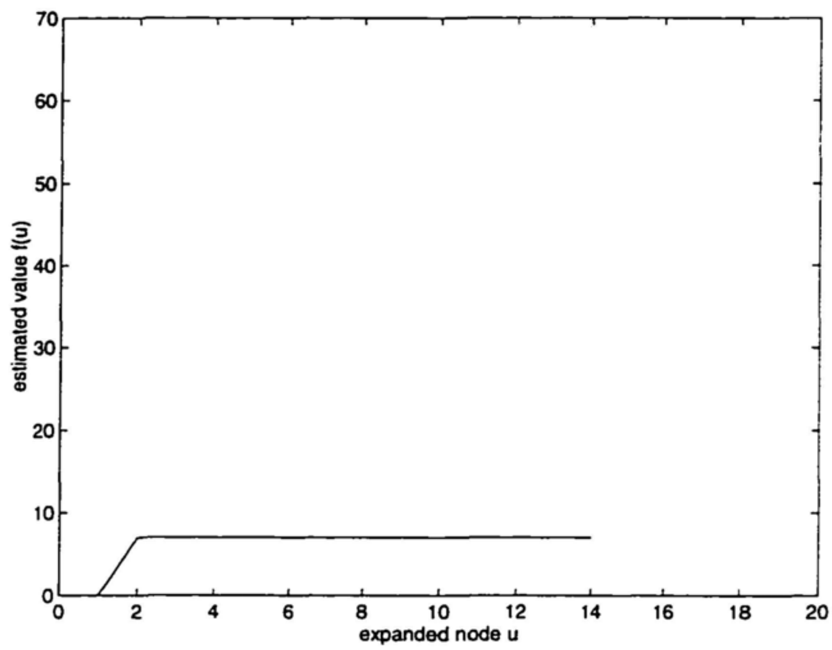
$$h(u) - h(w) \le \gamma(n_{ia} \to n_{ja}) \le c(u,w). \tag{4.21}$$

Since $h'(u)$ and $h'(w)$ are the approximate values of $h(u)$ and $h(w)$, respectively, we cannot derive that h' is also monotonic from the statement of h being monotonic. However, because of the right most item in (4.20), we have observed that $h'(u) - h'(w) \le c(u,w)$ holds in all the experiments we did for checking this monotone restriction. We have also observed that the estimated f values of the sequence of nodes expanded by the A* are indeed nondecreasing.

Three examples are given in Figs. 4.9–4.11. Figs. 4.9(a)–4.11(a) show three pairs of characters to be matched. The curves in Figs. 4.9(b)–4.11(b) illustrates the nondecreasing characteristic of the estimated f values with respect to the sequences of nodes expanded by the A*, when the A* searches for finding the optimal matching between two characters of each pair. The A* algorithm runs with the pruning operation introduced in the last section. In the experiments, the primitives used for representing characters are strokes and thus the matching procedure is a stroke-based recognition method. The two characters in Fig. 4.9(a) are the most similar. The two characters in Fig. 4.10(a) belong to the same class but the input character has a connected stroke. The two characters in Fig. 4.11(a) belong to different classes and are most dissimilar. From Figs. 4.9(b)–4.11(b), we can clearly see that in these experiments, the more dissimilar the two characters under matching, the greater their matching distance and the more the nodes expanded by the A* to reach a best goal node. By the way, the number of nodes generated by the A* is greater than the number of nodes expanded by the A* in a search. In general, expanding a node generates several successors of it in a tree. For the three examples, the node numbers generated by the A* are 27, 43 and 58, respectively.

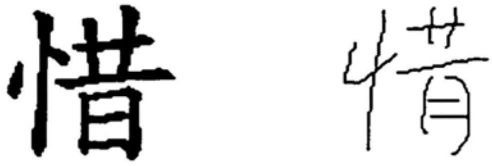
Now we continue discussing the criteria to stop the A*. Fig. 4.11(b) shows that the estimated f value reaches 40 quickly and then increases slowly in the matching between the two dissimilar characters. If we terminate the A* when the current estimated f value is greater than a threshold (say, 40), much search time will be saved. Therefore, we use the following criteria for stopping the A*.

(a)

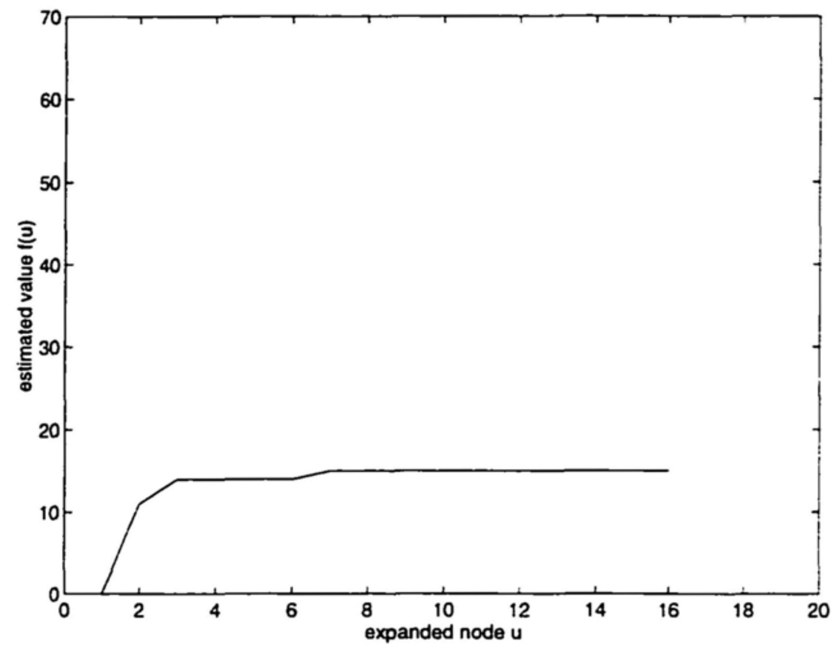


(b)

Figure 4.9: Example 1 for showing the nondecreasing estimated f values. (a) A model character and one of its handwritten characters. (b) Estimated f values. The node numbers denote the sequence of the nodes expanded by the A*, when the A* searches a tree for the optimal matching between the two characters. The A* terminates after expanding 14 nodes. The matching distance between the two characters is 7.

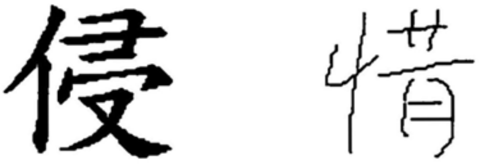


(a)

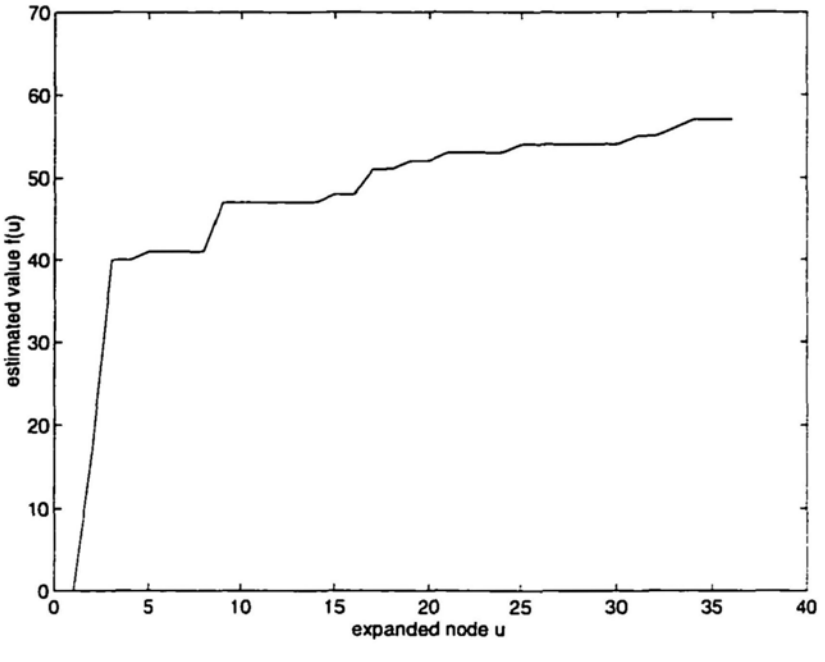


(b)

Figure 4.10: Example 2 for showing the nondecreasing estimated f values. (a) A model character and one of its handwritten characters. (b) Estimated f values. The node numbers denote the sequence of the nodes expanded by the A*, when the A* searches a tree for the optimal matching between the two characters. The A* terminates after expanding 16 nodes. The matching distance between the two characters is 15.



(a)



(b)

Figure 4.11: Example 3 for showing the nondecreasing estimated f values. (a) A model character and the input character in Fig. 4.10(a). (b) Estimated f values. The node numbers denote the sequence of the nodes expanded by the A^*, when the A^* searches a tree for the optimal matching between the two characters. The A^* terminates after expanding 36 nodes. The matching distance between the two characters is 57.

Criterion 1. The A* algorithm will be terminated if it reaches a best goal node.

Criterion 2. The A* algorithm will be terminated if the current estimated f value is greater than a global threshold T_g.

Criterion 3. The A* algorithm will be terminated if the current estimated f value is greater than a varied threshold T_v.

The global threshold T_g is a constant and is usually set not to be too small so that the model character corresponding to an input character may not be missed. Criterion 3 is used to further speed up the A* algorithm. In Chinese character recognition, even after a preclassification procedure, in general, there are still many model characters, ranging from tens to hundreds, which need to be matched with an input character. Suppose there are m such model characters $G_1, G_2, ..., G_m$. Let the matching distance between a model G_i and an input G be $\xi(G_i, G)$. If a model character G_i is more similar to the input than all the remaining characters $G_{i+1}, G_{i+2}, ..., G_m$ (that have not been matched with G yet), then we have

$$\xi(G_i, G) \leq \xi(G_j, G), \quad j = i+1, i+2, ..., m.$$

Now suppose the first i model characters have been matched with the input character. If we let

$$T_v = T_v(i) = \xi(G_l, G) = \min_{1 \leq k \leq i} \{\xi(G_k, G)\}, \tag{4.22}$$

then we can reduce the search time of the A* in successive matchings when $T_v(i) < T_g$. For example, the A* is now searching a tree for the optimal matching between G_{i+1} and G. If the current estimated f value is greater than $T_v(i)$, then that f is nondecreasing implies

$$\xi(G_{i+1}, G) > T_v(i) = \xi(G_l, G),$$

and thus no more search for the current matching is needed.

It is conventional that after finishing the recognition of an input character, an on-line recognition system gives several model characters that are considered most similar to the input, in order that the real model may not be missed. To reach this goal, we may set

$$T_v = T_v(i) = \epsilon + \min_{1 \le k \le i} \{\xi(G_k, G)\}, \tag{4.23}$$

where $\epsilon > 0$ is a predefined value.

Remark. Criteria 2 and 3 utilize the nondecreasing characteristic of the estimated f values of the sequence of expanded nodes to speed up the A*. In a search for the optimal matching between two dissimilar characters, if f increases very slowly, the two criteria may produce little effect. Recall that we use complete relational graphs to represent Chinese characters. The spatial and temporal relations between any two primitives (strokes or segments) give much information for the matching aim. In the matching between two dissimilar characters, with the help of the GPF constraints on input primitives, many relations between model primitives are not compatible with those between input primitives and thus large estimated f values yield quickly. Fig. 4.11 clearly shows such an example, in which if we set $T_g = 40$, then the A* will stop after expanding 5 nodes instead of expanding 36 nodes. Therefore, the Criteria 2 and 3 are very efficient for speeding up the A^*. It is also worth noting that in the other two common methods for graph matching, maximal clique-based approaches [5, 7, 10, 12, 13, 16] and relaxation labeling approaches [18, 23, 33, 41, 51, 54, 99, 106], the iteration procedures of the algorithms of these approaches are not directly relative to their corresponding distance measures. Whether or not a matching distance is large can be known only when an algorithm has terminated. This is one of the reasons why these approaches require large computational time when they are applied to on-line Chinese character recognition [13, 16, 18].

4.5 Experimental Results

In this section, we give some recognition results to demonstrate the performance of the graph matching based on-line Chinese character recognition method. We also provide some data to show the search efficiency of the A^* with the pruning operation. All algorithms, including the preprocessing algorithms, are implemented in C. The computer used is a PC/Pentium at 166MHz.

4.5.1 Stroke-Based Recognition

In the stroke-based recognition method, the primitives are strokes and the stroke-based representation of Chinese characters is employed. 300 frequently-used Chinese characters each with stroke number between 9 and 11 are selected for testing the performance of the proposed method. (A Chinese character has an average of 10 strokes [88].) Some of the model character are shown in Fig.4.12. The global threshold T_g and the parameter ϵ (see (4.23)) are set to be 40 and

15, respectively. More than 7000 Chinese characters written by 10 people were used as test data. The subjects were asked not to write the characters in their cursive styles, but there were no stroke order constraints on their writing. A set of handwritten characters having correct stroke numbers are shown in Fig.4.13, in which their corresponding model characters are also given. These characters are all recognized correctly. For such characters with no connected strokes, the recognition rate is about 98.7%.

Fig.4.14 shows another set of correctly-classified characters each having one or two connected/split strokes. For such characters, the recognition rate is about 91.2%. If we consider the first 5 model candidates that are regarded as the most similar to an input character, we obtain a recognition rate of 93.6%. Stroke-based recognition methods cannot tolerate too many connected strokes. The reason is that connected strokes change the stroke types, stroke spatial relations and stroke numbers of characters, all of which make the matching distance between an input character having several connected strokes and its model increase greatly.

The average time for classifying an input character is about 0.3 second. Such a satisfactory recognition speed is due to the heuristic estimate, the pruning strategy and the criteria for stopping the A^*. We have explained how Criteria 2 and 3 can save the computational time of the A^* in Section 4.4.3. Now we will present three matching examples each in three cases to demonstrate the usefulness of the pruning operation and the heuristic function h. The pairs of characters in Figs. 4.9–4.11 are employed in the experiments. Three cases are considered: (1) neither heuristic information nor the pruning operation is used ($h = 0$, $GPF = \emptyset$); (2) the heuristic estimate defined in (4.10) but not the

冒奏便侯冠侵信契峙度彥
很徊待律拱指按政故毒要
紅紀送計訂准凋兼倚借哲
造訓記訊哭宴屑展峨峰差
座徒悟恥捉捆捎氧鬼班高
偏奢寄屠崔崩崎崇彗彬彫
得徘惜情惟患悉挽捨毫堂
保俑俊俗俄俚前勇南咬哀
常您接掉斜族晚望梁械淡
都烹紡紋紗停挖息悍拳拿
挫旁時咳咽品哈咯型國夠
密巢蚊造訓記訊教假做偶
姜娃姚狩牲室客封屎奕急
恫拼拭拷是昨柿染某架柯

Figure 4.12: Some model Chinese characters for testing the stroke-based recognition method.

Figure 4.13: Some test characters having correct stroke numbers, together with their corresponding models.

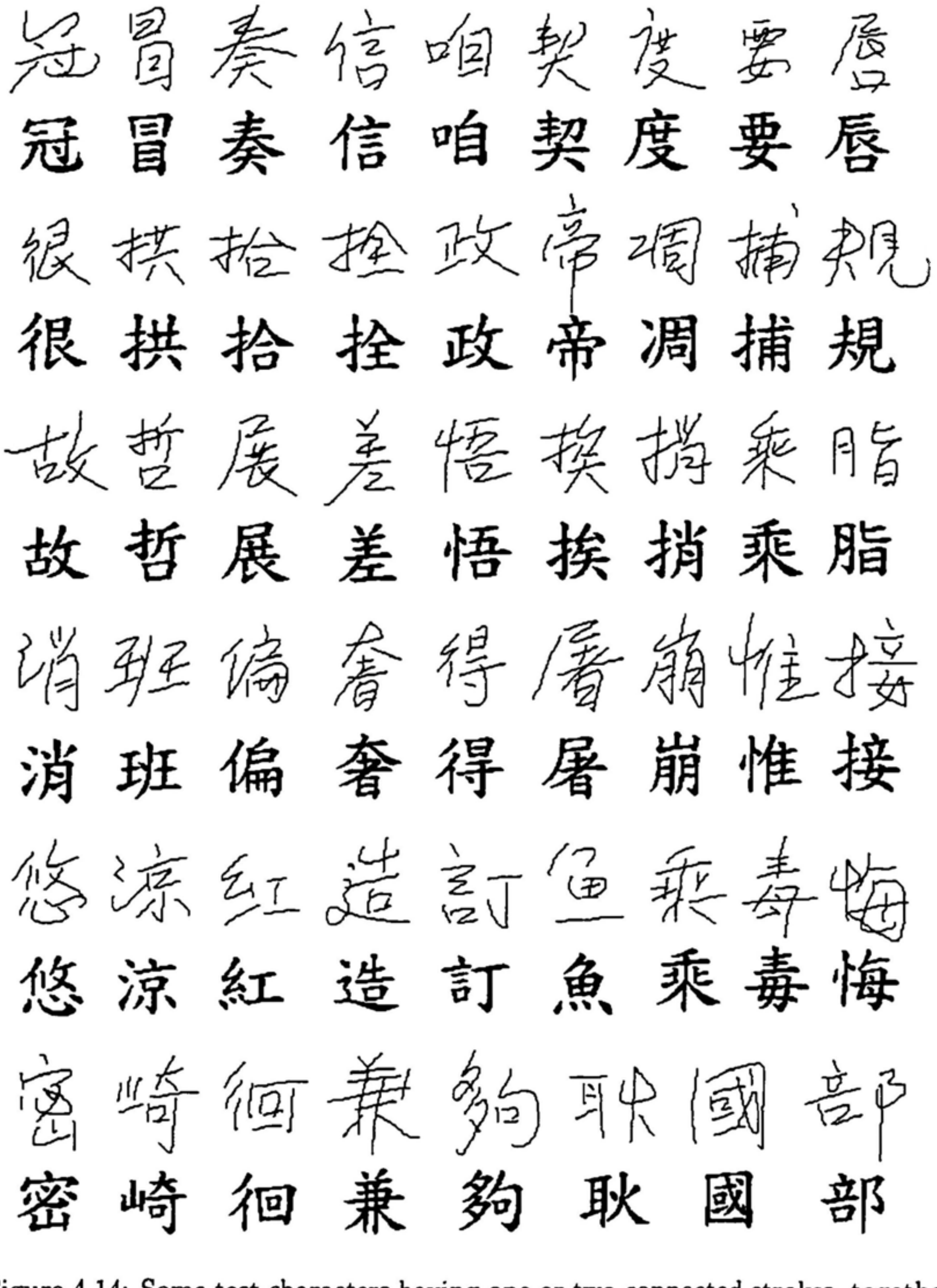


Figure 4.14: Some test characters having one or two connected strokes, together with their corresponding models.

Table 4.1: Numbers of nodes generated by the A* for three matching examples each in three cases.

	惜—惜	惜—惜	侵—惜
$h = 0, GPF = \emptyset$	780	1202	5540
$h \neq 0, GPF = \emptyset$	114	380	2458
$h \neq 0, GPF \neq \emptyset$	27	43	58

pruning operation is used ($h \neq 0$, $GPF = \emptyset$); (3) both the heuristic estimate and the pruning operation are used ($h \neq 0$, $GPF \neq \emptyset$). Table 4.1 gives the numbers of nodes generated by the A* algorithm in search. As can be seen, the A* needs to generate a lot of nodes to obtain the optimal matching between two characters in case 1. In case 2, the A* can be speeded up by using the heuristic information, but the results are still not satisfactory, especially when two characters are not similar. In case 3, the search efficiency of the A* is improved significantly by adding the tree pruning operation.

4.5.2 Segment-Based Recognition

In the segment-based recognition method, the primitives are segments and the segment-based representation of Chinese characters is employed. 54 Chinese characters (a subset of the models in the stroke-based recognition experiments) each with stroke number between 9 and 11 are used for testing. The values of the global threshold T_g and the parameter ϵ are also chosen to be 40 and 15, respectively.

More than 6000 Chinese characters written by 9 people were used as test data. No stroke number and order constraints were imposed on the writing.

The recognition rate mainly varies with the numbers of connected strokes appearing in the handwritten characters. For the characters each having less than 3 connected strokes such as those in Figs.4.13 and 4.14, the recognition rate achieves 98.2%. For the characters written each with 4 to 7 strokes, as shown in Fig.4.15, the recognition rate is 94.2%. For the characters written each with only 1 to 3 strokes, as shown in Fig.4.16, the recognition rate is 88.6%. The average recognition rate is about 95%. These results are very promising.

Compared with the stroke-based recognition method, the segment-based recognition method can allow more connected strokes in freely-written characters. This is because (1) most spatial relations among the segments (not including extra segments in connected strokes) of an input character remain unchanged, and (2) the rules in the segment preprocessing are very useful for breaking connected strokes and deleting some of the extra segments (see Section 2.4).

In the experiments, we found that the assignment of temporal relations between strokes/segments of the model characters almost tolerated all the handwriting order deviations in the test data, thanks to the "don't care" temporal relations. Of course, incorrect recognition will occur if too many stroke order deviations exist in input characters. For example, Fig. 4.17 shows a Chinese character, the strokes of which are labeled with the numbers indicating their standard order of writing, and one of its handwritten characters having many stroke order deviations.[5] In this case, a user may choose the re-classification phase, which do not utilize the stroke/segment order information of Chinese

[5]One who is not familiar with Chinese characters such as a foreigner may write the character in that stroke order.

Figure 4.15: Some test characters written each having 4 to 7 strokes, together with their corresponding models.

Figure 4.16: Some test characters written each having 1 to 3 strokes, together with their corresponding models

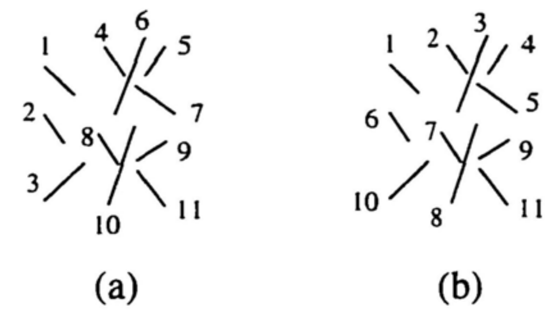


Figure 4.17: (a) A Chinese character whose strokes are labeled with the numbers that indicate their standard order of writing. (b) A handwritten version of (a) which has many stroke order deviations.

characters for recognition,[6] to obtain a correct recognition result, without the need to write the character again. Therefore, our segment-based recognition method is stroke number and stroke order free.

The average time for classifying an input character is about 0.09 second when there are 54 model characters. The stroke-based recognition method requires 0.06 second to classify an input if the 54 models are also used. The reason why the segment-based method takes more time is that (1) the number of segments of a character is greater than or equal to the number of strokes of the same character, and allowing more freely written characters often leads to extra segments (the more the node numbers of two graphs under matching, the larger the state space tree for the graph matching); (2) the heuristic function h in the stroke-based method is more precise than that in the segment-based method because the stroke-based method uses more stroke types (15 standard stroke types), which provide more information of Chinese characters than the 5 segment types. Comparing the segment-based and the stroke-based methods,

[6]The recognition method of the re-classification is the same as the original one except that the weight w_4 in (3.11) is set to be 0.

we prefer the former if the computer running it is not too slow to accept.

4.6 Comparisons of the Segment-Based Recognition Method with Several Other Studies

In this section, we will make some comparisons between our segment-based recognition method and several other methods published recently in international journals. Generally speaking, it is difficult, if not impossible, to compare the recognition results of various methods for on-line Chinese character recognition. This is because of different subjects in different experiments, different constraints imposed on handwriting, no standard on-line captured Chinese character databases for testing a method, and so on. A Chinese may write the Chinese characters that is very difficult to be recognized by others if there are no constraints on his/her writing. For example, our method fails to recognize the characters shown in Fig. 4.18, which are written either too cursively or having great distortions in shape. Thus we cannot say a method is absolutely better than another just according to the recognition rates reported. Besides the recognition rate, we also have to consider more factors such as recognition time, stroke

Figure 4.18: Some handwritten characters that the segment-based method cannot recognize. Their corresponding model characters are also shown.

number and stroke order constraints, and degrees of deformation of handwritten characters.

Lin *et al.* [56] proposed a deviation-expansion model to represent Chinese characters. The dynamic programming is used to perform the character matching. Their approach is stroke-based and in essence a string matching one. The approach requires that an input character should not have more than one stroke number variation and more than two connected strokes. There were 5400 models in their experiments. A preclassification step was employed. A recognition rate of 87.4% and an average recognition time of 2.5 seconds per character on a PC/386 at 25MHz were reported.

Chou *et al.* [21] extended the above model to a segment-based deviation tree. The approach is also a string matching one, so cannot tolerate more than two stroke order deviations. There were 5104 models in the experiments and a preclassification step was employed. They reported a recognition rate of 94.88% for untrained characters and a recognition time of 0.7 second per character on a PC/486 at 25MHz.

In [17], Chen *et al.* developed a stroke-sequence decision tree and position matching method, which can only recognize the handwritten characters with less than two stroke number variations and is not stroke order free. No recognition rate and recognition time were reported.

Tsay and Tsai [92] used attributed string matching by split-and-merge for on-line Chinese character recognition. The proposed method can recognize cursive characters but imposes the constraint of correct stroke orders on them. There were 3100 model characters with stroke numbers ranging from 1 to 24. A recognition rate of 96.2% and a recognition time of 2.5 seconds per character on a PC/AT were reported.

Chou and Tsai [22] proposed a discrete iteration scheme to solve the problem. Their method is not stroke order free. The provided test characters are in block style and almost have no connected strokes. There were 5401 models in their experiments. A preclassification stage was used. A recognition rate of 91.8% and a recognition time of less than 2 seconds were reported. But the authors did not mention what kind of computer was used.

In [40], Hsieh et al. employed a greedy algorithm for bipartite matching to complete the recognition. The method is stroke order free. The provided test characters each with less than 10 strokes are neatly written, some of which have one or two connected strokes. There were 452 models in the experiments. No preclassification stage was used. A recognition rate of 89.7% was reported. If the first three candidates were considered, they obtained a recognition rate of 96.29%. The average recognition time was 39 seconds per character on a Sun workstation.

Chen and Lee [16] proposed a fuzzy attribute representation for Chinese characters and used a NP-complete maximum clique finding algorithm to perform the graph matching. There were 650 models each with a stroke number between 1 and 12 in their experiments. A preclassification stage was utilized to reduce the number of models required to be matched with an input. A recognition rate of 95.64% and a recognition time of 2 seconds per character on a Sun SPARC-II workstation were reported. The method is stroke order free, but no test data was provided.

In summary, the above methods except the last two are not stroke order free and impose basically correct stroke order constraint on handwriting. The method in [40] is stroke order free, but it can only recognize the neatly written characters, some of which have one or two connected strokes. The tolerance of stroke number variations in the method of [16] is difficult to judge since no test data was given. In general, when there are several thousand model characters in a recognition system, a preclassification stage is required to save the overall recognition time.

Our segment-based method is stroke order and stroke number free. From the test data provided in our and the other experiments, it is seen that our method can recognize more cursively written characters, and at the same time imposes no stroke order constraint on the handwriting. There is not much difference between the recognition rate obtained in our experiments and the others. As mentioned above, the recognition rate is just one of the factors to judge how good a recognition method is.

When there are several thousands of model characters added in our recognition system, a preclassification stage is also necessary. We will discuss the preclassification problem in Section 7.2. If 500 models are required to be matched with an input character after preclassification, our segment-based method can complete a recognition within one second on average. Now we compare the recognition time required by our method and those in [40] and [16], all of which are stroke order free. Since the computational power of the PC/Pentium is similar to that of the Sun workstations used in [40] and [16], our method requires much less recognition time than the method in [40]. It is also faster than that in [16]. The recognition time of 2 seconds per character reported in [16] was obtained under the conditions: there were 650 models each with stroke number from 1 to 12 and a preclassification stage was employed,[7] while our method can have a recognition time of about 1 second per character if there are 500 models each with stroke number from 9 to 11 and no further preclassification is performed. From the above comparisons, we see that our segment-based recognition method is very promising.

4.7 Summary

In this chapter, we have formulated the graph matching as a state space search problem. The optimal matching between two graphs is equivalent to finding the best goal node in a search tree. State space search itself do not change the NP-complete property inherent in the graph matching problem. To obtain good search efficiency, we have used the A^* algorithm to perform the heuristic search, and proposed the following schemes to speed up the A^*.

- A heuristic function h, which has been proved to be a lower bound on h^* and monotonic, is defined to make the A^* expand fewer nodes in a search tree.
- A tree pruning strategy, which employs the geometric position features of strokes (or segments) of Chinese characters to prune a search tree, is proposed to let the A^* have more or less the function of a bird's eye view, in other words, to let the A^* avoid searching the nodes that have very little chance to be located in the optimal path from the initial node to the best goal node in a tree.

[7] Obviously, the recognition speed of this method is too slow when there are several thousands of models.

- Criteria 2 and 3 are presented to stop the A*, together with Criterion 1, by utilizing the monotone of the estimated f value. The two criteria are based on the fact that in Chinese character recognition, finding the final optimal matching between two dissimilar characters is not necessary if we have known their distance is great enough.

The experimental results show that the recognition speeds of our stroke-based and segment-based recognition methods are sufficiently fast for practical applications, even if the frequently-used 5000 or more Chinese characters are added. In common recognition of input Chinese characters (the first phase), the methods can tolerate most of the stroke order variations due to the "don't care" temporal relations between strokes (segments). To deal with a character with great stroke order deviations, the re-classification stage (the second phase) can be effected (without the need to write the character again), which ignores the stroke (segment) order information to perform recognition. Therefore, the methods are stroke order free. The results also show that the segment-based method can recognize the handwritten characters having many connected strokes, so it is stroke number free too.

We have made some comparisons between our segment-based method and several other studies published recently in international journals. Considering their recognition rates, recognition time and tolerances of stroke order and stroke number variations, we see that out method is very promising.

Parts of the results presented in this chapter have been published in [58, 59, 60, 61, 62, 63].

Chapter 5

A Two-Layer Assignment Method

5.1 Introduction

The assignment problem is a well known one in operations research and can be solved by the Hungarian method [39, 70, 75]. In this chapter, we propose a two-layer assignment method for the on-line Chinese character recognition problem. The objective of the first layer assignment is to estimate the costs of primitive (stroke or segment) correspondences between two Chinese characters according to their primitive types and spatial-temporal relations, and the objective of the second layer assignment is to find the primitive correspondences between the two characters such that the total correspondence cost is minimized. We also present two schemes to save computational time, which reduce the complexity of the method from $O(n^5)$ to $O(n^3)$, where n is the greater number between the two primitive numbers of two characters.

In Section 5.2, we briefly review the assignment problem and the methods to solve it. The two-layer assignment formulation of on-line Chinese character recognition is given in Section 5.3. The two complexity reduction schemes are discussed in Section 5.4. Some experimental results are presented in Section 5.5. In the last of this chapter is the summary.

5.2 The Assignment Problem

The assignment problem is a special type of the linear programming problem. An assignment is useful for modeling a situation, in which there are two distinct sets of objects of equal numbers (say, n), and we need to form them into pairs on a one-to-one base. There is a cost c_{ij} associated with mapping object i to object j, $i, j = 1, 2, ..., n$. We call $[c_{ij}]_{n\times n}$ a cost matrix. Given $[c_{ij}]_{n\times n}$, an assignment problem of order n is to find a permutation matrix[1] $\mathbf{P} = [x_{ij}]_{n\times n}$ to

$$\text{Minimize } cost(\mathbf{P}) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{5.1}$$

$$\text{Subject to } \sum_{j=1}^{n} x_{ij} = 1 \text{ for } i = 1 \text{ to } n \tag{5.2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \text{ for } j = 1 \text{ to } n \tag{5.3}$$

$$x_{ij} \in \{0, 1\}. \tag{5.4}$$

[1] A permutation matrix is a square matrix $[x_{ij}]_{n\times n}$ whose elements satisfy: $\sum_{i=1}^{n} x_{ij} = 1, \sum_{j=1}^{n} x_{ij} = 1$, and $x_{ij} = 0$ or 1.

We call $\min\{cost(\mathbf{P})\}$ the minimum total assignment cost. Here is a feasible solution for an assignment of order 4:

$$\begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \tag{5.5}$$

The cost matrix $[c_{ij}]_{n \times n}$ can be represented by a complete bipartite graph (see Fig. 5.1(a)). The vertexes in the left column denote a set of objects and those in the right column denote the other set of objects. Each left vertex is

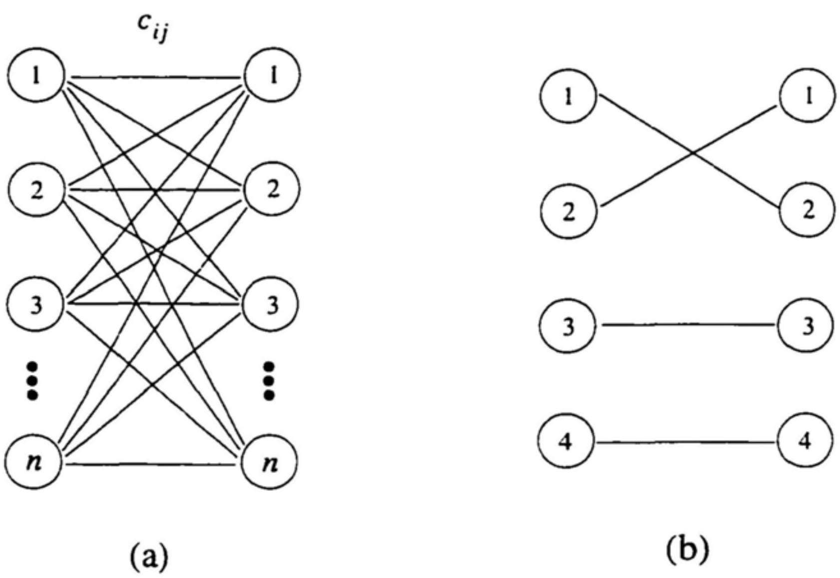


Figure 5.1: (a) A complete bipartite graph. (b) A complete matching of a bipartite graph corresponding to the assignment in (5.5).

connected to each right vertex by an edge. The cost of an edge joining left vertex i with right vertex j is defined as c_{ij}. A complete matching[2] of the

[2] In graph theory, a **matching** in a graph is a set of edges, no two of which share a vertex. When the cardinality of a matching is the largest possible, the matching is termed **complete**.

bipartite graph corresponds to an assignment, and vice versa. For example, the matching in Fig. 5.1(b) corresponds to the assignment in (5.5). Thus the assignment problem in (5.1)–(5.4) is equivalent to that of finding a minimum cost complete matching in the bipartite graph of Fig. 5.1(a). That is why it is also known as the weighted bipartite graph minimum cost complete matching problem. We also call $[c_{ij}]_{n\times n}$ an edge cost matrix.

The Hungarian method is a popular one with the complexity $O(n^3)$ for solving the assignment problem [75]. In addition, there are several other methods for it, such as the cost scaling algorithm [32], the auction algorithm [8] and the auction algorithm incorporating scaling [74].

5.3 A Two-Layer Assignment Formulation of on-Line Chinese Character Recognition

The similarity comparison between two characters can be made by the two steps: (1) find the segment[3] correspondences between the two characters; (2) use a measure to calculate their similarity based on the segment correspondences. We will discuss these two steps respectively in the following.

5.3.1 Finding Segment Correspondences between Two Characters

Fig.5.2 shows a model character and its handwritten character after the preprocessing. We want to obtain such segment correspondences:

[3]In the rest of this chapter, the segments of characters are used as primitives.

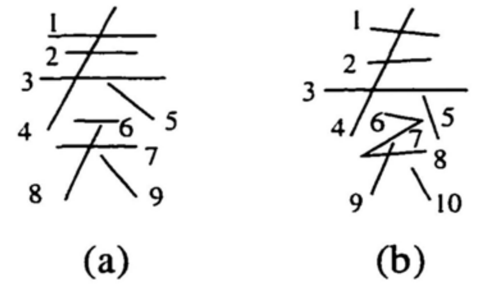


Figure 5.2: A model character (a) and its handwritten character (b).

$$1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 5 \to 5, 6 \to 6, 7 \to 8, 8 \to 9, 9 \to 10, \lambda \to 7, \quad (5.6)$$

where $\lambda \to 7$ denotes a correspondence between a **dummy** segment in character (a) and the extra segment in character (b). Now we represent various segment correspondences using a bipartite graph, as shown in Fig.5.3. Segments 1–9 in

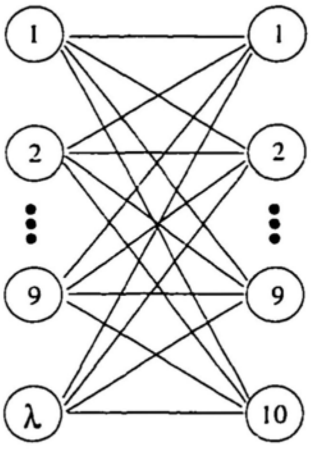


Figure 5.3: A bipartite graph formulation of segment correspondences between the two characters in Fig. 5.2.

character (a) are denoted by vertexes 1–9 in the left column, respectively, and segments 1–10 in character (b) are denoted by vertexes 1–10 in the right column, respectively. Vertex λ (a dummy segment) is added in the left column to make the segment numbers of the two characters equal.

To sum up, for obtaining the segment correspondences between character 1 with segment number n_1 and character 2 with segment number n_2 (assume $n_1 \leq n_2$ without loss of generality), a complete bipartite graph is constructed by two vertex columns each with n_2 vertexes. There is an edge joining a vertex in the left column with a vertex in the right column. n_1 vertexes in the left column denote n_1 real segments of character 1 and the other $n_2 - n_1$ vertexes denote $n_2 - n_1$ dummy segments. In the right column, n_2 vertexes denote n_2 real segments of character 2.

Now a critical issue is how to derive the edge costs of a bipartite graph such that after solving the corresponding assignment problem, we can have the desirable segment correspondences between two characters. The idea in graph matching discussed in the last two chapters may be borrowed, the goal of which, roughly speaking, is to find an optimal matching between two characters such that the sum of all the costs of segment type correspondences and the costs of relation correspondences between segments is minimized, in other words, to make the segment type correspondences and the relation correspondences between two characters as compatible as possible. Keeping this in mind, we will propose an approach for deriving the costs of edges of a bipartite graph, i.e., the cost matrix $[c_{ij}]_{n\times n}$, where c_{ij} is the cost of edge joining vertex i in the left vertex column with vertex j in the right column in the bipartite graph.

First, we define $c_{ij} = dr$ if i denotes a dummy segment and j a real segment, or vice versa, where dr is a positive value and is determined by experiment. Let B_n be the bipartite graph created with character 1 and character 2, and its edge cost matrix be $[c_{ij}]_{n\times n}$. Let i and j be two real segments in characters 1 and 2, respectively. The information used to derive $[c_{ij}]_{n\times n}$ is the segment types and the spatial-temporal relation matrixes of the two characters. We define

$$c_{ij} = \gamma(i \to j) + \rho_{ij}, \tag{5.7}$$

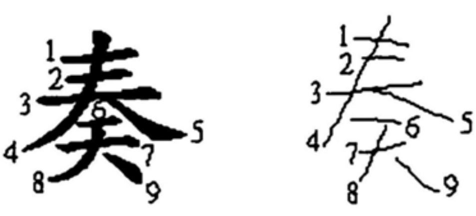
where $\gamma(i \to j)$ is the correspondence cost between the type of segment i and the type of segment j as defined in Table 3.4,[4] and ρ_{ij} is the minimum cost of a complete matching in a new bipartite graph B^{ij}_{n-1} (or the minimum total assignment cost of the new assignment problem corresponding to B^{ij}_{n-1}). The left column of B^{ij}_{n-1} consists of all the vertexes except vertex i in the left column of B_n, and the right column of B^{ij}_{n-1} consists of all the vertexes except vertex j in the right column of B_n. The edge cost matrix $[c^{ij}_{kl}]_{(n-1)\times(n-1)}$ of B^{ij}_{n-1} is directly derived by

$$c^{ij}_{kl} = \gamma(k \to l) + \gamma((i,k) \to (j,l)), \tag{5.8}$$

where k and l are two real segments in characters 1 and 2, respectively, $\gamma(k \to l)$ is the segment type correspondence cost, (i,k) and (j,l) are the spatial-temporal relation from i to k and from j to l respectively, and $\gamma((i,k) \to (j,l))$, called relation correspondence cost, is defined the same as the arc correspondence cost in (3.11). If there is one dummy segment between segment k and segment l, we define $c^{ij}_{kl} = dr$.

Now we explain the meaning of the definition of c_{ij}. c_{ij} is equal to a sum of two terms. The first is a segment type correspondence cost. When the type of segment i is the same as that of segment j, this term costs the least ($\gamma(i \to j) = 0$). To better understand the second term ρ_{ij}, more description is needed. In the following, by using an example, we will state that if the two

[4] Note that in this chapter, c_{ij} is called a segment correspondence cost, and $\gamma(i \to j)$ is called a segment type correspondence cost.



(a) (b)

Figure 5.4: A Chinese character (a) and its very similar handwritten style (b).

characters under comparison is very similar to each other and if segment i is just the segment that should correspond to segment j in character 2, then $\rho_{ij} = 0$.

Consider two very similar characters, character 1 and character 2, as shown in Figs.5.4(a) and (b). The segment correspondences:

$$1 \to 1, 2 \to 2, 3 \to 3, 4 \to 4, 5 \to 5, 6 \to 6, 7 \to 7, 8 \to 8, 9 \to 9 \qquad (5.9)$$

are what we want to obtain. We say they are very similar in the sense that (1) the costs of segment type correspondences in (5.9) are all equal to 0; (2) the relation correspondence cost $\gamma((i_1, i_2) \to (j_1, j_2)) = 0$, where i_1, j_1, i_2, j_2 satisfy the condition that $i_1 \to j_1$ and $i_2 \to j_2$ are any two different segment correspondences in (5.9). Now suppose $i = 9$ and $j = 9$. Then $c_{99} = \gamma(9 \to 9) + \rho_{99}$. ρ_{99} corresponds to a new assignment problem of order 8. Let $[c_{kl}^{99}]_{8\times8}$ be the cost matrix of the assignment problem. Without loss of generality, suppose the subscripts k and l on c_{kl}^{99} denote just segment k and segment l in character 1 and character 2, respectively. By (5.8), we then have

$$c_{mm}^{99} = \gamma(m \to m) + \gamma((9, m) \to (9, m)) = 0, \quad m = 1, 2, ..., 8. \qquad (5.10)$$

Thus all diagonal elements in $[c_{kl}^{99}]_{8\times8}$ are 0. A permutation matrix $\mathbf{P}' = [x_{kl}]_{8\times8}$ with $x_{mm} = 1$, $m = 1, 2, ..., 8$ and $x_{kl} = 0, k \neq l$ is a solution to the assignment problem and the minimum total assignment cost $\rho_{99} = 0$.

Obviously, if the two characters are not so similar, the costs of the segment type correspondences or segment relation correspondences discussed above will not be 0 and we have $\rho_{99} \neq 0$. Besides, if $i = 9$ and $j = 1$, by (5.8) we can see that all the elements of the cost matrix $[c_{kl}^{91}]_{8\times 8}$ (of another assignment problem) will be greater than 0 because of the incompatible segment type correspondences and relation correspondences. This results in a minimum total assignment cost $\rho_{91} > 0$.

Table 5.1 gives the whole matrix $[\rho_{ij}]_{9\times 9}$ by solving the 81 assignment problems of order 8, and Table 5.2 shows the corresponding cost matrix $[c_{ij}]_{9\times 9}$, for finding the segment correspondences between the two characters in Fig. 5.4. By applying the Hungarian method to the assignment problem with this cost matrix $[c_{ij}]_{9\times 9}$, we can obtain the segment correspondences in (5.9). Therefore, c_{ij} defined in (5.7) better reflects the degree of incompatibility between segment i and segment j.

Fig.5.5 shows the structure for obtaining segment correspondences between two characters. There are $n \times n$ assignment problems of order $n - 1$ in layer 1 and there is one assignment problem of order n in layer 2. Obtained by solving the assignment problem with the cost matrix $[c_{kl}^{ij}]_{(n-1)\times(n-1)}$ in layer 1, ρ_{ij} is used together with the segment type correspondence cost $\gamma(i \rightarrow j)$ to estimate the cost c_{ij} in the assignment problem in layer 2. The cost c_{kl}^{ij} is calculated by utilizing the information of segment types and relations of the two characters. Note that if one of the segments i and j is a dummy segment, c_{ij} is simply set to be dr. We do not specify these cases in the figure for simplicity. By Fig. 5.5, we call the proposed method a **two-layer assignment method**.

Table 5.1: Matrix $[\rho_{ij}]_{9\times9}$ for estimating the cost matrix $[c_{ij}]_{9\times9}$.

	1	2	3	4	5	6	7	8	9
1	0	12	30	39	48	58	72	85	96
2	12	0	12	21	30	40	56	67	78
3	24	12	0	15	18	28	44	55	72
4	21	15	15	0	37	37	59	50	91
5	54	36	42	57	0	26	30	67	48
6	64	52	34	55	20	0	16	35	50
7	72	60	48	63	36	16	0	7	30
8	79	67	55	56	41	23	7	0	29
9	96	84	78	103	42	38	24	35	0

Table 5.2: Cost matrix $[c_{ij}]_{9\times9}$ for finding the segment correspondences between two characters in Fig. 5.4.

	1	2	3	4	5	6	7	8	9
1	0	12	30	46	50	58	72	92	98
2	12	0	12	28	32	40	56	74	80
3	24	12	0	22	20	28	44	62	74
4	28	22	22	0	44	44	66	50	98
5	56	38	44	64	0	28	32	74	48
6	64	52	34	62	22	0	16	42	52
7	72	60	48	70	38	16	0	14	32
8	86	74	62	56	48	30	14	0	36
9	98	86	80	110	42	40	26	42	0

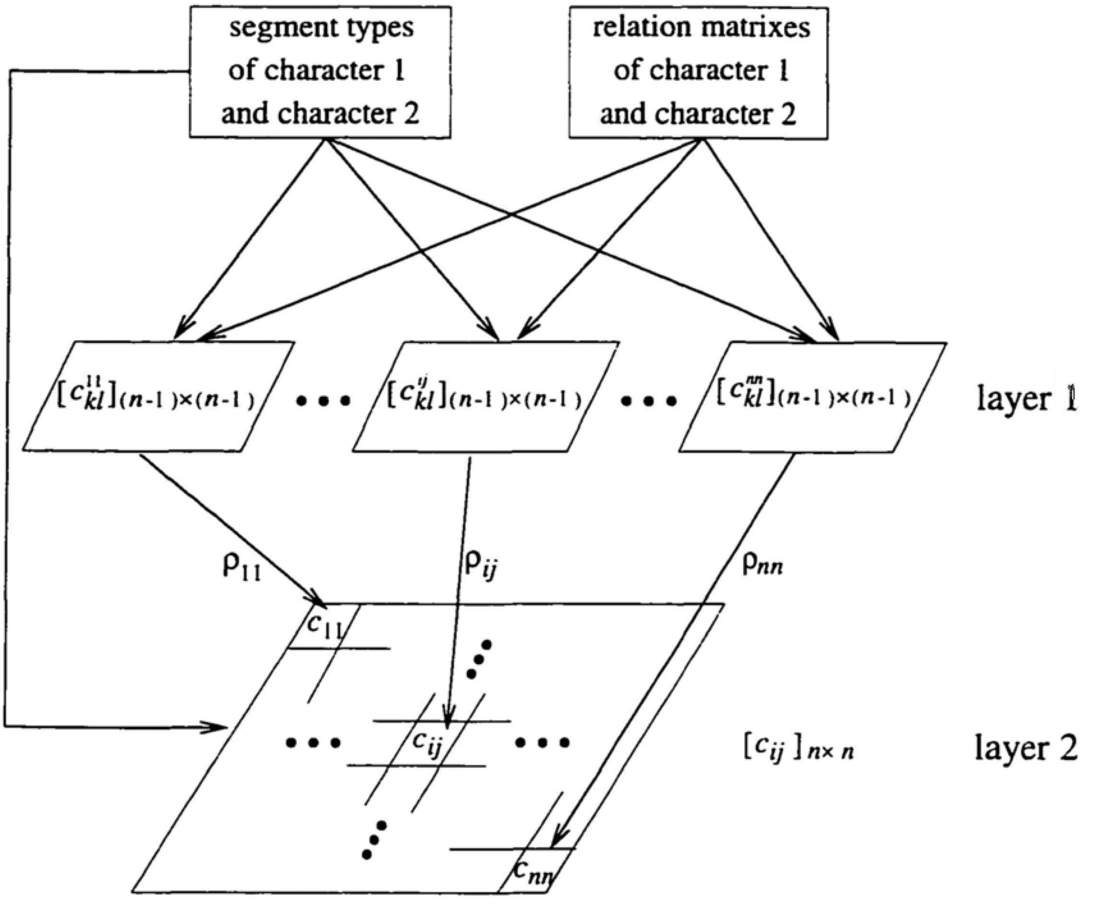


Figure 5.5: The structure for obtaining segment correspondences between character 1 and character 2.

It is not difficult to estimate the computational complexity of the two-layer assignment method. Let the numbers of segments of two characters under comparison be n_1 and n_2, respectively, and $n = \max\{n_1, n_2\}$. Then the $n \times n$ assignment problems of order $n - 1$ in layer 1 can be solved by applying the Hungarian method $n \times n$ times, each requiring $O((n-1)^3)$ time. The effort to create the cost matrix of each assignment problem in layer 1 is $O((n-1)^2)$. Thus the total effort made in layer 1 is $O(n^5)$. There is only one assignment problem of order n in layer 2, which can be solved in $O(n^3)$ time. Therefore, the complexity of the entire two-layer assignment method is $O(n^5)$.

5.3.2 Calculating the Similarity of Two Characters

After solving an assignment problem in layer 2, we obtain a permutation matrix $\mathbf{P}^*$ that denotes a set of segment correspondences between two characters. With the $\mathbf{P}^* = [x^*_{ij}]_{n\times n}$ and the cost matrix $[c_{ij}]_{n\times n}$ of the assignment problem, we have the minimum total assignment cost ($MTAC$):

$$MTAC(\mathbf{P}^*) = \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x^*_{ij}. \tag{5.11}$$

It can be used as a distance to compare two characters. However, in our experiments we found that it is not good enough.

Suppose A and B are two model characters and C is an input handwritten character belonging to the class of A. Let $\mathbf{P}^*{}_{AC}$ and $\mathbf{P}^*{}_{BC}$ be the permutation matrixes for segment correspondences between A and C and for segment correspondences between B and C, respectively. In general, $MTAC(\mathbf{P}^*{}_{AC}) < MTAC(\mathbf{P}^*{}_{BC})$, but sometimes, when C is not very similar to A, we may have $MTAC(\mathbf{P}^*{}_{AC}) > MTAC(\mathbf{P}^*{}_{BC})$. However, if the **matching cost** β defined in the following (5.12) is used, we will still have $\beta_{AC} < \beta_{BC}$, where β_{AC} and β_{BC} are the matching cost between A and C and the matching cost between B and C respectively.

By Definitions 3.5 and 3.6, it is not difficult to see that $\mathbf{P}^{*}$ corresponds to a node mapping f_N^{*} (and an arc mapping f_A^{*} led by f_N^{*}). Therefore, we use the following matching cost, which is similar to (4.1) and is a simplified version of (3.2) in Definition 3.7, to calculate the distance between two characters:

$$\begin{aligned}\beta(f_N^{*}, f_A^{*}) &= \sum_{i\to j\in Q_1}\gamma(i\to j) + \sum_{i\to\lambda\in Q_2}\gamma(i\to\lambda) + \sum_{\lambda\to j\in Q_3}\gamma(\lambda\to j) \\ &+ \sum_{(i,k)\to(j,l)\in Q_4}\gamma((i,k)\to(j,l)) \qquad (5.12)\end{aligned}$$

where Q_1 is the set of correspondences between real segments; Q_2 and Q_3 are the sets of correspondences between real segments and dummy segments; Q_4 is the set of relation correspondences from (i,k) to (j,l), $i \neq \lambda$, $j \neq \lambda$, $k \neq \lambda$, $l \neq \lambda$. Compared with $MTAC$, β reflects more directly and fully the relation compatibility of segment correspondences between two characters.

5.4 Two Complexity Reduction Schemes

Although the two-layer assignment method can be implemented in polynomial time $O(n^5)$, we find that an algorithm with such running time is not suitable for on-line recognition of Chinese characters. Hence, we propose two complexity reduction schemes for the recognition problem.

5.4.1 A Lower Bound Estimate

In the two-layer assignment method, the main computational effort is to solve the $n \times n$ assignment problems of order $n - 1$ with the Hungarian method. This results in an $O(n^5)$ algorithm in layer 1, the aim of which is to obtain ρ_{ij} $(i, j = 1, 2, ..., n)$ and then derive the cost matrix $[c_{ij}]_{n\times n}$. The following theorem is useful for obtaining an estimate of ρ_{ij}.

Theorem 5.1 *Given an assignment problem:*

$$\text{Minimize } cost(\mathbf{Q}) = \sum_{i=1}^{n}\sum_{j=1}^{n} d_{ij}x_{ij},$$

where $\mathbf{Q} = [x_{ij}]_{n\times n}$ *and* $[d_{ij}]_{n\times n}$ *are a permutation matrix and a cost matrix respectively, a lower bound estimate* e *of the minimum value of* $cost(\mathbf{Q})$*, i.e.,*

$$\min_{\mathbf{Q}}\{cost(\mathbf{Q})\} \geq e \tag{5.13}$$

can be found by

$$e = \sum_{i=1}^{n} min\_r_i + \sum_{j=1}^{n} min\_c_j, \tag{5.14}$$

where $min\_r_i = \min\{d_{i1}, d_{i2}, ..., d_{in}\}$ *is the smallest element in row* i *in the cost matrix* $[d_{ij}]_{n\times n}$*, and* $min\_c_j = \min\{d'_{1j}, d'_{2j}, ..., d'_{nj}\}$ *is the smallest element in column* j *in the cost reduction matrix* $[d'_{ij}]_{n\times n}$*, where* $d'_{ij} = d_{ij} - min\_r_i$*, for all* $i, j = 1, 2, ..., n$.

Proof. Investigating the Hungarian method in [48], after steps (a) and (b) of the method, we have a reduction of costs that is exactly equal to e. Let the total reduction of costs in the steps following step (b) be e'. Then we have

$$\min_{\mathbf{Q}}\{cost(\mathbf{Q})\} = e + e'. \tag{5.15}$$

Since $e' \geq 0$, (5.13) holds. $\square$

Theorem 5.1 provides us an approach to approximately obtain ρ_{ij} (and thus c_{ij}), by using (5.14) instead of the Hungarian method. Calculating e from an $n \times n$ matrix needs $O(n^2)$ time. With this estimate, the effort made in layer 1 now becomes $O(n^4)$, which is also the complexity of the two-layer assignment method.

5.4.2 Geometric Position Constraints

The derivation of the cost matrix $[c_{ij}]_{n \times n}$ consumes the most computational time in the two-layer assignment method. c_{ij} is the correspondence cost between segment i of a character and segment j of another character. In fact, with the help of the geometric position features (GPF) of a model character, which are defined in Definition 4.5,[5], we can reduce much computational time spent in layer 1. Look at Fig.5.4, the correspondences between segment 1 in character (a) and one of the segments 5–9 in character (b) is obviously unreasonable. Now we can use the GPF of a model character to decide whether a cost c_{ij} in the cost matrix in layer 2 needs to be estimated in layer 1. Let i be a segment of a model character and j be a segment of an input character. Let the ith element of the GPF of the model be (d_i, x_i, y_i). If $x_i \leq od(D_{d_i}(j)) \leq y_i$, i.e., the geometric position of segment i and that of segment j are compatible, then c_{ij} will be estimated in layer 1; otherwise, just set c_{ij} a sufficiently large positive value.

With the geometric position constraints, the computational complexity of the two-layer assignment method can be reduced further. In general, an integer

[5] In Definition 4.5, the primitives are strokes of Chinese characters. When it is applied to the segment-based recognition method, the primitives should be segments.

interval $[x_i, y_i]$ (see Definition 4.5) satisfies $y_i - x_i \leq K$ (say, 5). In this case, at most Kn elements in a cost matrix $[c_{ij}]_{n\times n}$ need further computation. Therefore, by using the above two complexity reduction schemes, the time required by the two-layer assignment method is $O(Kn^3) = O(n^3)$.

5.5 Experimental Results

The two-layer assignment method has been implemented in C on a PC/Pentium at 166MHz. The primitives of the method are segments. Before an input character is recognized, its segment types and the relations between its segments are extracted first in the preprocessing procedure. 54 Chinese characters, which have been used in the experiments for testing the segment-based state space search for graph matching method (see Section 4.5.2), are also used here. The parameter dr is chosen to be 5, which is equivalent to the cost of a segment deletion in graph matching.

The test data consists of more than 3000 Chinese characters written by 6 people. No stroke number and stroke order constraints were imposed on their writing. Fig. 5.6 shows a set of the test characters that are all recognized correctly. The recognition rate varies with the numbers of connected strokes appearing in the handwritten characters. The stroke numbers of the model characters are between 9 and 11. For the characters each having less than 3 connected strokes, the recognition rate is 96.3%. For the characters written each having 4 to 7 strokes, the recognition rate is 92.0%. The average recognition rate is 93.8%.

The average time for recognizing an input character is 0.085 second. If the two complexity reduction schemes are not used, the recognition time is about

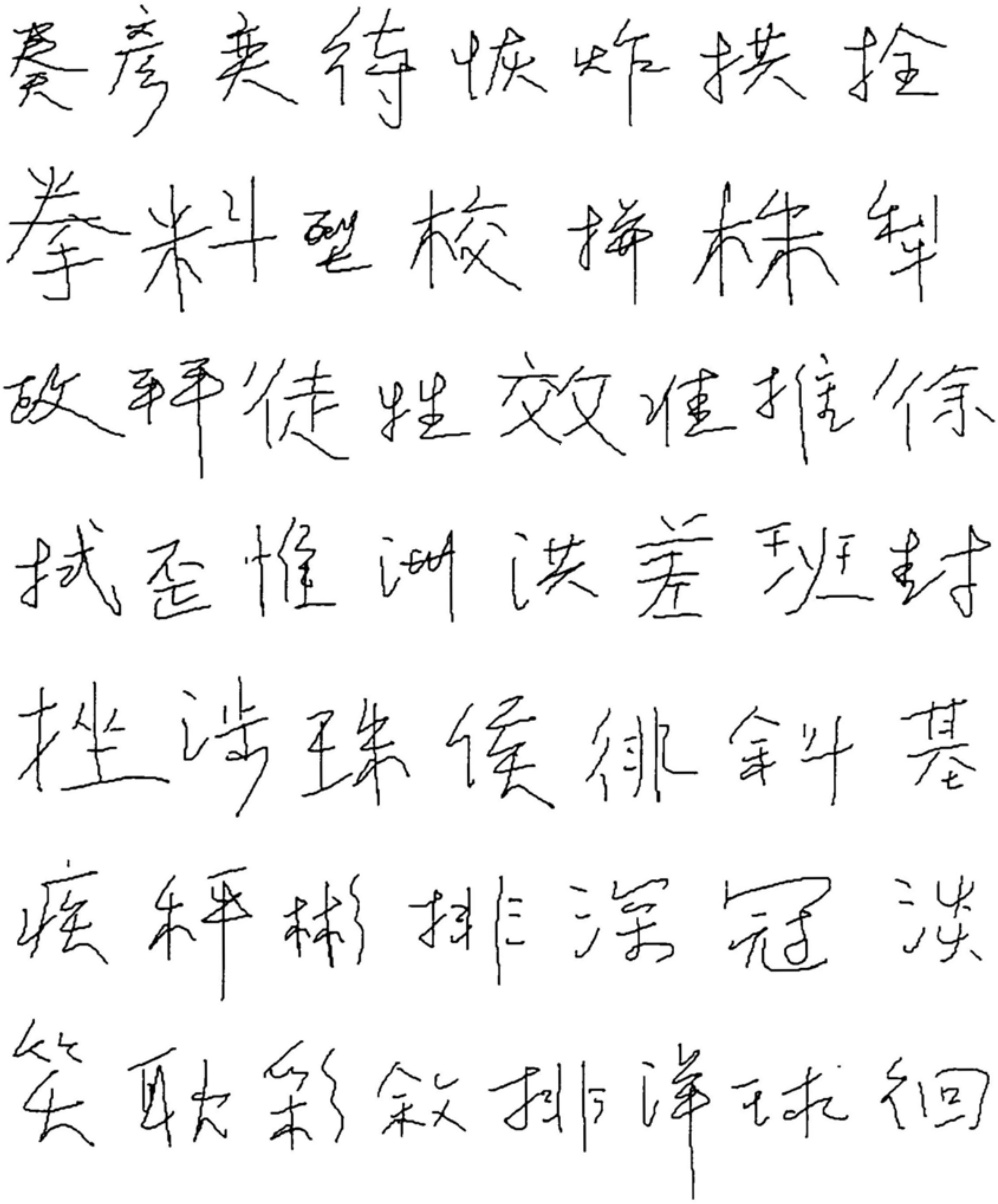


Figure 5.6: Some test data in the experiments.

Table 5.3: Performance comparison of three methods

method	average recognition rate	average recognition time
stroke-based state space search	90.9%	0.06 second
segment-based state space search	95.6%	0.09 second
segment-based two-layer assignment	93.8%	0.085 second

1 second, but the recognition rate almost remains the same. This is because (1) even if the lower bound estimate is not precise in some cases, the segment correspondences obtained by solving an assignment problem with an estimated cost matrix in layer 2 are still correct, and (2) the geometric position constraints of model characters are loose enough to tolerate most of the segment position variations in handwriting.

The set of test data was also used to test the stroke-based and segment-based state space search methods. Table 5.3 shows the results. Comparing these three methods, we can see that the stroke-based state space search method runs fastest but the recognition rate is lowest, and the segment-based state space search method runs slightly slower than the segment-based two-layer assignment method but has the highest recognition rate.

5.6 Summary

In this chapter, we have proposed a two-layer assignment method for on-line Chinese character recognition. Finding segment correspondences between two characters is formulated as a weighted bipartite graph minimum cost complete matching problem, which corresponds to an assignment problem of order n and can be solved by the Hungarian method in $O(n^3)$ time, where n is the greater number between the two segment numbers of the two characters. In order to derive the cost matrix of the assignment problem in layer 2, $n \times n$ assignment problems of order $n-1$ are created in layer 1. The costs of segment type correspondences and relation correspondences between the two characters are used to generate the cost matrixes in layer 1. To save the computational time, a lower bound estimate is employed to obtain the approximate value of the minimum total assignment cost of each assignment problem in layer 1. In addition, the geometric position features of model characters are used to avoid wasting computation on unreasonable segment correspondence costs. These two schemes reduce the complexity of the method from $O(n^5)$ to $O(n^3)$.

The experimental results are satisfactory. Compared with the segment-based state space search for graph matching method, the method in this chapter runs slightly faster and has a little lower recognition rate when they were used to recognize the characters with an approximate degree of deformation.

Chapter 6

A Fast String Matching Method

6.1 Introduction

We have proposed two methods for on-line Chinese character recognition in the last two chapters. They all use both the types of primitives (strokes or segments) of Chinese characters and relations between primitives to carry out recognition. The experimental results have shown that they have the ability to tolerate wide stroke order and stroke number deviations in handwriting. However, they require relatively large amounts of computation and are suitable to be implemented on relatively high-end CPUs such as a PC/486 or above. In various practical applications, many products such as portable electronic diaries, electronic Chinese-English dictionaries, multi-functional telephones and simple Chinese typewriters, may need fast small-memory-requirement recognition methods, due to their low-end CPUs and limited memory space equipped.

Each Chinese character has a standard stroke writing order and Chinese people write a Chinese character basically according to its standard stroke order.

On-line devices can capture the temporal information of the writing, including the order, number and direction changes of strokes. If model characters and input handwritten characters are all represented by their corresponding primitive (stroke or segment) strings, then the two-dimensional character recognition problem can be transformed to a relatively simple one-dimensional string matching problem. This fact makes some researchers study string matching based recognition methods [21, 22, 55, 56, 57, 92].

In this chapter, based on Wagner and Fischer's string matching (WFSM) algorithm [96], we propose a recognition method that incorporates the geometric position constraints of primitives into the WFSM algorithm. The method is very fast. Its running time is $O(mn)$ for matching two characters with primitive numbers m and n.

In Section 6.2, we briefly review the WFSM algorithm. Its application to on-line Chinese character recognition is presented in Section 6.3. Some experimental results are given in Section 6.4. In Section 6.5, we propose an extension of the string matching method when there may be several primitive strings to represent a model character. The summary in Section 6.6 ends this chapter.

6.2 The WFSM Algorithm

In [96], Wagner and Fischer discussed the string-to-string correction problem and suggested the application of the WFSM algorithm to spelling correction.

Let $S = s_1 s_2 ... s_m$ be a finite string of m symbols. A **null** symbol is denoted by λ. An edit operation is a pair $(a, b) \neq (\lambda, \lambda)$ and is written as $a \rightarrow b$, where a and b are two strings of length 0 or 1. Three edit operations on strings are as follows:

- code insertion: $\lambda \rightarrow a$
- code substitution: $a \rightarrow b$
- code deletion: $a \rightarrow \lambda$

The application of an edit operation $a \rightarrow b$ to string S results in string R, which is written as $S \Rightarrow R$ via $a \rightarrow b$. Let E be a sequence $e_1, e_2, ..., e_p$ of p edit operations. An edit transformation of string S to string R is a sequence of strings $S_0, S_1, ..., S_p$ such that $S = S_0$, $R = S_p$ and $S_{i-1} \Rightarrow S_i$ via e_i for $1 \leq i \leq p$.

In order to measure the similarity (or **distance**) between two strings. Costs associated with the edit operations are necessary. Let γ be a cost function that assigns to each edit operation $a \rightarrow b$ a nonnegative real number $\gamma(a \rightarrow b)$. γ can also be extended to a sequence of edit operations $E = e_1, e_2, ..., e_p$ by setting $\gamma(E) = \sum_{i=1}^{p} \gamma(e_i)$. If $p = 0$, i.e., no edit operation is applied, we define $\gamma(E) = 0$. The edit distance (or distance for short) between strings S and R is defined as

$$\begin{aligned}\delta(S, R) \quad = \quad & \min\{\gamma(E) | E \text{ is a sequence of edit operations} \\ & \text{that transforms } S \text{ to } R\}. \end{aligned} \tag{6.1}$$

To simplify the calculation of the edit distance $\delta(S_1, S_2)$ between two strings $S_1 = s_1 s_2 ... s_m$ and $S_2 = s'_1 s'_2 ... s'_n$, Wagner and Fischer defined a structure called a **trace** as follows. A trace from S_1 to S_2 is a triple (T, S_1, S_2) (or simply T when the strings S_1 and S_2 are understood), where T is any set of ordered pairs of integers (i, j) satisfying:

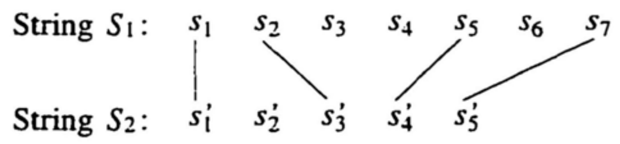
(a) $1 \leq i \leq m$ and $1 \leq j \leq n$;

(b) for any two distinct pairs (i_1, j_1) and (i_2, j_2) in T

(1) $i_1 \neq i_2$ and $j_1 \neq j_2$;

(2) $i_1 < i_2$ if and only if $j_1 < j_2$.

A pair (i, j) denotes a line joining the ith symbol of S_1 and the jth symbol of S_2. Condition (a) ensures that the lines touch the symbols of the respective strings. Condition (b1) ensures that each symbol of either string is touched by at most one line; and condition (b2) ensures that no two lines cross. Fig. 6.1 shows an example of a trace (T, S_1, S_2).



$T = \{(1,1),(2,3),(5,4),(7,5)\}$

Figure 6.1: A trace (T, S_1, S_2).

Let T be a trace from $S_1 = s_1 s_2 ... s_m$ to $S_2 = s'_1 s'_2 ... s'_n$. Let I and J be the sets of symbols in S_1 and S_2 respectively not touched by any line in T. The cost of T is defined by

$$cost(T) = \sum_{(i,j)\in T} \gamma(s_i \to s'_j) + \sum_{i\in I} \gamma(s_i \to \lambda) + \sum_{j\in J} \gamma(\lambda \to s'_j). \tag{6.2}$$

Wagner and Fischer proved that if the cost function γ is a metric, then

$$\delta(S_1, S_2) = \min\{cost(T) | T \text{ is a trace from } S_1 \text{ to } S_2\}. \tag{6.3}$$

We may call the process of finding the distance between two strings **string matching**. The following WFSM algorithm is used to calculate $\delta(S_1, S_2)$ of a trace from S_1 to S_2.[1]

The WFSM algorithm [96]

Input: String $S_1 = s_1 s_2 ... s_m$ and string $S_2 = s'_1 s'_2 ... s'_n$.

Output: The least cost $D[m, n]$ of a trace from S_1 to S_2.

begin

$D[0, 0] := 0;$

for $i = 1, 2, ..., m$ **do** $D[i, 0] := D[i-1, 0] + \gamma(s_i \to \lambda);$

for $j = 1, 2, ..., n$ **do** $D[0, j] := D[0, j-1] + \gamma(\lambda \to s'_j);$

for $i = 1, 2, ..., m$ **do**

for $j = 1, 2, ..., n$ **do**

begin

$d_1 := D[i-1, j-1] + \gamma(s_i \to s'_j);$

$d_2 := D[i-1, j] + \gamma(s_i \to \lambda);$

$d_3 := D[i, j-1] + \gamma(\lambda \to s'_j);$

$D[i, j] := \min\{d_1, d_2, d_3\};$

end

end

---

[1] We give the algorithm here again for the convenient description following it, although it has been shown in Section 2.3.2.

It is clear that the running time of the WFSM algorithm is $O(mn)$ for obtaining $\delta(S_1, S_2) = D[m, n]$ for string S_1 of length m and string S_2 of length n, and the memory space required is $O(mn)$. If the least cost trace T from S_1 to S_2 is required, the following algorithm with running time $O(m + n)$ will print the pairs in T using the information stored in array D of the above algorithm.

Least cost trace printing algorithm [96]

Input: Array D.

Output: Printed results of T.

begin

  $i := m;\ j := n;$

  while $(i \neq 0\ \&\ j \neq 0)$ **do**

    if $D[i, j] = D[i-1, j] + \gamma(s_i \rightarrow \lambda)$ **then** $i := i - 1;$

    else

      if $D[i, j] = D[i, j-1] + \gamma(\lambda \rightarrow s'_j)$ **then** $j := j - 1;$

      else

        begin

          print$((i, j));$

          $i := i - 1;\ j := j - 1;$

        end

end

6.3 Application of the WFSM Algorithm to on-Line Chinese Character Recognition

Wagner and Fischer suggested that the WFSM algorithm can be applied to spelling correction. In fact, it may also be applied to some pattern recognition problems such as the chain code string matching given in Section 2.3.2 and the on-line Chinese character recognition presented in this section.

For better understanding traces, the WFSM algorithm, and the extension of the WFSM algorithm to be presented in Section 6.5, we construct a network as shown in Fig. 6.2, in which each path from the source node (0,0) to the target node (m, n) corresponds to a trace from $S_1 = s_1 s_2 \ldots s_m$ to $S_2 = s'_1 s'_2 \ldots s'_n$. For example, the bold path P from node (0,0) to node (7,5) in Fig. 6.3 corresponds

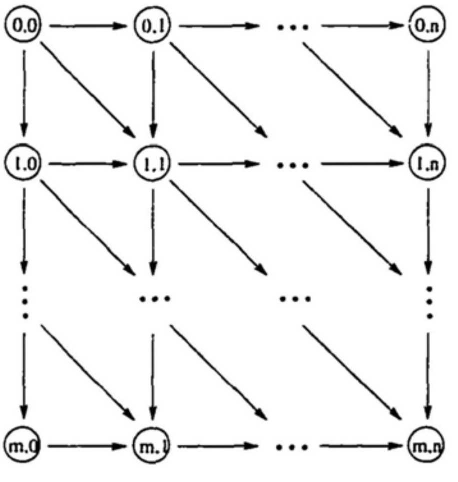
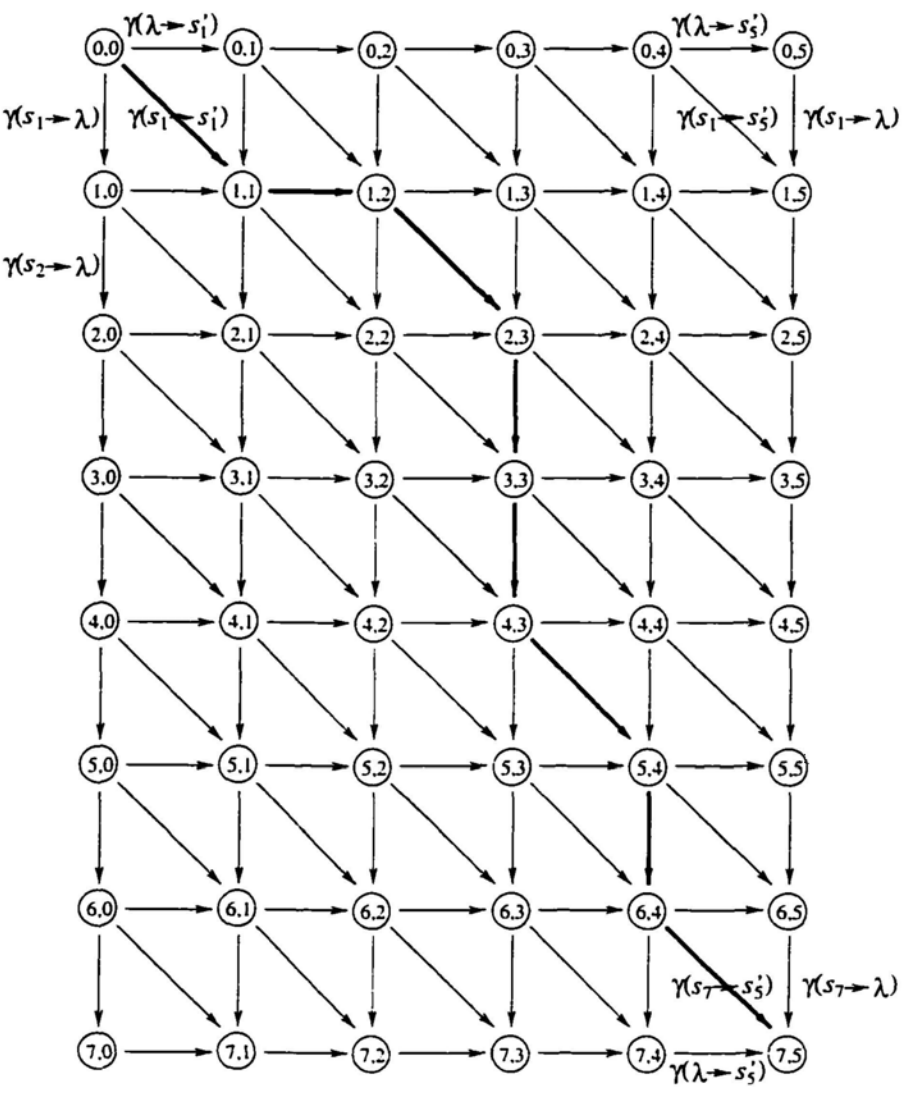


Figure 6.2: A network for calculation of the distance between a string of length m and a string of length n.

to the trace T in Fig. 6.1. We may assign a cost $\gamma(s_i \rightarrow \lambda)$ to the arc from node $(i-1, j)$ to node (i, j), a cost $\gamma(\lambda \rightarrow s'_j)$ to the arc from node $(i, j-1)$ to node (i, j), and a cost $\gamma(s_i \rightarrow s'_j)$ to the arc from node $(i-1, j-1)$ to node (i, j),



$P=\{(0,0),(1,1),(1,2),(2,3),(3,3),(4,3),(5,4),(6,4),(7,5)\}$

Figure 6.3: A network for calculation of the distance between two strings in Fig. 6.1. The bold path P corresponds to the trace $T = \{(1,1),(2,3),(5,4),(7,5)\}$.

for all $i = 1, 2, ..., m$ and $j = 1, 2, ..., n$, as shown in Fig. 6.3. If the cost $cost(P)$ of a path from the source node to the target node is defined as the sum of the costs of all the arcs in the path, it is easy to see that

$$cost(P) = cost(T), \tag{6.4}$$

where P corresponds to T. Therefore, the calculation of the distance between two strings is equivalent to finding the least cost of a path from the source node to the target node in the corresponding network. We call such a path the **shortest path**.

Considering Fig. 6.3, we see that the problem of finding the shortest path in the network can be divided into 8 stages in the vertical direction (or 6 stages in the horizontal direction). Such an optimization problem can be solved by a dynamic-programming algorithm with the following recursive relationship between two successive stages:

$$\begin{aligned} D[i,j] &= \min\{D[i-1,j-1] + \gamma(s_i \to s'_j), D[i-1,j] + \gamma(s_i \to \lambda), \\ &\quad D[i,j-1] + \gamma(\lambda \to s'_j)\}, \end{aligned} \tag{6.5}$$

where $D[i,j], D[i-1,j-1], D[i-1,j]$ and $D[i,j-1]$ are the least costs of the paths from node (0,0) to node (i,j), to node $(i-1,j-1)$, to node $(i-1,j)$, and to node $(i,j-1)$, respectively. Comparing (6.5) with the algorithmic equations in the two-layer **for**-loops in the WFSM algorithm, we will find that the WFSM is actually a dynamic-programming algorithm.

The application of the WFSM algorithm to the on-line Chinese character recognition problem is direct. We represent a model character with a string $S = s_1 s_2 ... s_m$, where $s_1, s_2, ..., s_m$ are the primitive (stroke or segment) types of the m primitives of the character, arranged in its standard order of writing. We also represent an input character having n primitives (after preprocessing) with a string $R = r_1 r_2 ... r_n$, where $r_1, r_2, ..., r_n$ are the primitive types of the n primitives of the character, arranged in its input order of writing. Then the comparison of similarity between S and R can be formulated as the problem of calculating the least cost of a path from the source node to the target node in the network formed with S and R. The WFSM algorithm can be used to carry out the calculation.

If the primitives are strokes (segments, respectively), then the stroke (segment, respectively) type correspondence costs in Table 3.2 (Table 3.4, respectively) may be used as the stroke (segment, respectively) substitution cost $\gamma(s_i \rightarrow s'_j)$, and the stroke (segment, respectively) insertion cost and the stroke (segment, respectively) deletion cost are defined as $\gamma(s_i \rightarrow \lambda) = \gamma(\lambda \rightarrow s'_j) = id1$ ($id2$, respectively).

However, using only the information of primitive (type) strings of Chinese characters is not sufficient to distinguish a character from the others when there are stroke type variations and connected strokes in handwriting, as mentioned in Section 3.3.1. We have found that it is true after we implemented the WFSM algorithm. Now we propose a scheme in the following to make the string matching method have better ability to do the recognition work.

Recall that we have defined the geometric position features (GPF) of a model character in Definition 4.5. In the string matching method, the GPF of a model character can help to assign a cost to the primitive substitution between a model primitive and an input primitive.

Let i denote the ith primitive of a model character and j a primitive of an input character. Let the ith element of the GPF of the model be (d_i, x_i, y_i). If $x_i \le od(D_{d_i}(j)) \le y_i$, i.e., the geometric position of segment i and that of segment j are compatible, then the primitive substitution cost $\gamma(s_i \rightarrow s'_j)$ is the same as that defined above, where s_i and s'_j are the primitive types of i and j respectively; otherwise, set $\gamma(s_i \rightarrow s'_j)$ a sufficiently large positive value.

This scheme incorporates partial 2D geometric primitive position information into the 1D string matching, but only slightly increases its running time. The WFSM algorithm with the geometric position constraints on input primitives is given as follows. Its running time is still $O(mn)$.

The WFSM algorithm with geometric position constraints

Input: A model string $S_1 = s_1 s_2 ... s_m$, an input string $S_2 = s'_1 s'_2 ... s'_n$, the GPF of the model: $GPF = \{(d_i, x_i, y_i) | i = 1, 2, ..., m\}$, and the $od(D_q(j))$, $j = 1, 2, ..., n$, $q = 0, 1, ..., 7$, of the input character (see Section 4.4.2).

Output: The least cost $D[m, n]$ of a path from node (0,0) to node (m, n) in the network formed with S_1 and S_2.

begin

$D[0, 0] := 0$;

for $i = 1, 2, ..., m$ **do** $D[i, 0] := D[i-1, 0] + \gamma(s_i \rightarrow \lambda)$;

for $j = 1, 2, ..., n$ **do** $D[0, j] := D[0, j-1] + \gamma(\lambda \rightarrow s'_j)$;

for $i = 1, 2, ..., m$ **do**

for $j = 1, 2, ..., n$ **do**

begin

if $x_i \leq od(D_{d_i}(j)) \leq y_i$

then $d_1 := D[i-1, j-1] + \gamma(s_i \to s'_j)$;

else $d_1 := M$ (a sufficient large positive value);

$d_2 := D[i-1, j] + \gamma(s_i \to \lambda)$;

$d_3 := D[i, j-1] + \gamma(\lambda \to s'_j)$;

$D[i, j] := \min\{d_1, d_2, d_3\}$;

end

end

6.4 Experimental Results

In this section, we give the experimental results to demonstrate the performance of the string matching method, respectively for stroke-based recognition and segment-based recognition. All algorithms are implemented in C. The parameters $id1$ and $id2$ are all set to be 4.

6.4.1 Stroke-Based Recognition

When the primitives are strokes and the stroke type string of a Chinese character is used to represent it, the recognition method is called stroke-based string matching method. 300 Chinese characters each with stroke number between 9 and 11 are selected to be models, which have been used in the experiments for testing the stroke-based state space search for graph matching method (see Section 4.5.1). The test data consist of about 3000 Chinese characters written by 6 people. The subjects were asked to write the characters in their own habitual stroke writing orders, but not in their cursive styles.

Fig. 6.4 shows a set of testing characters, all of which were recognized correctly. For these characters each having less than three connected strokes, the recognition rate is about 91.8%. If we consider the first three model candidates, we obtains a recognition rate of 93.6%. The recognition rate is not sensitive to $id1$ (the stroke insertion cost and stroke deletion cost). It almost remains unchanged when $id1$ varies between 3 and 5. If connected strokes in each input character increases, the recognition rate decreases quickly because of too many stroke type variations.

The average time for recognizing a character is about 0.017 second on a PC/Pentium at 166 MHz. It is about 0.068 second on a PC/486 at 50 MHz. Compared with the stroke-based graph matching method, the stroke-based string matching method is about 17 times faster.

Now let us see how the deviations of stroke order affect the recognition results. Fig. 6.5 shows a model character (a) and its handwritten characters (b)–(j) having different stroke orders. We denote by $x \leftrightarrow y$ that a standard stroke x in the model character corresponds to an input stroke y. Then the deviations of stroke order in these input characters are listed as follows:

character (b) $2 \leftrightarrow 3$, $3 \leftrightarrow 2$;

character (c) $2 \leftrightarrow 4$, $3 \leftrightarrow 2$, $4 \leftrightarrow 3$;

character (d) $1 \leftrightarrow 3$, $2 \leftrightarrow 4$, $3 \leftrightarrow 1$, $4 \leftrightarrow 2$;

character (e) $7 \leftrightarrow 8$, $8 \leftrightarrow 7$;

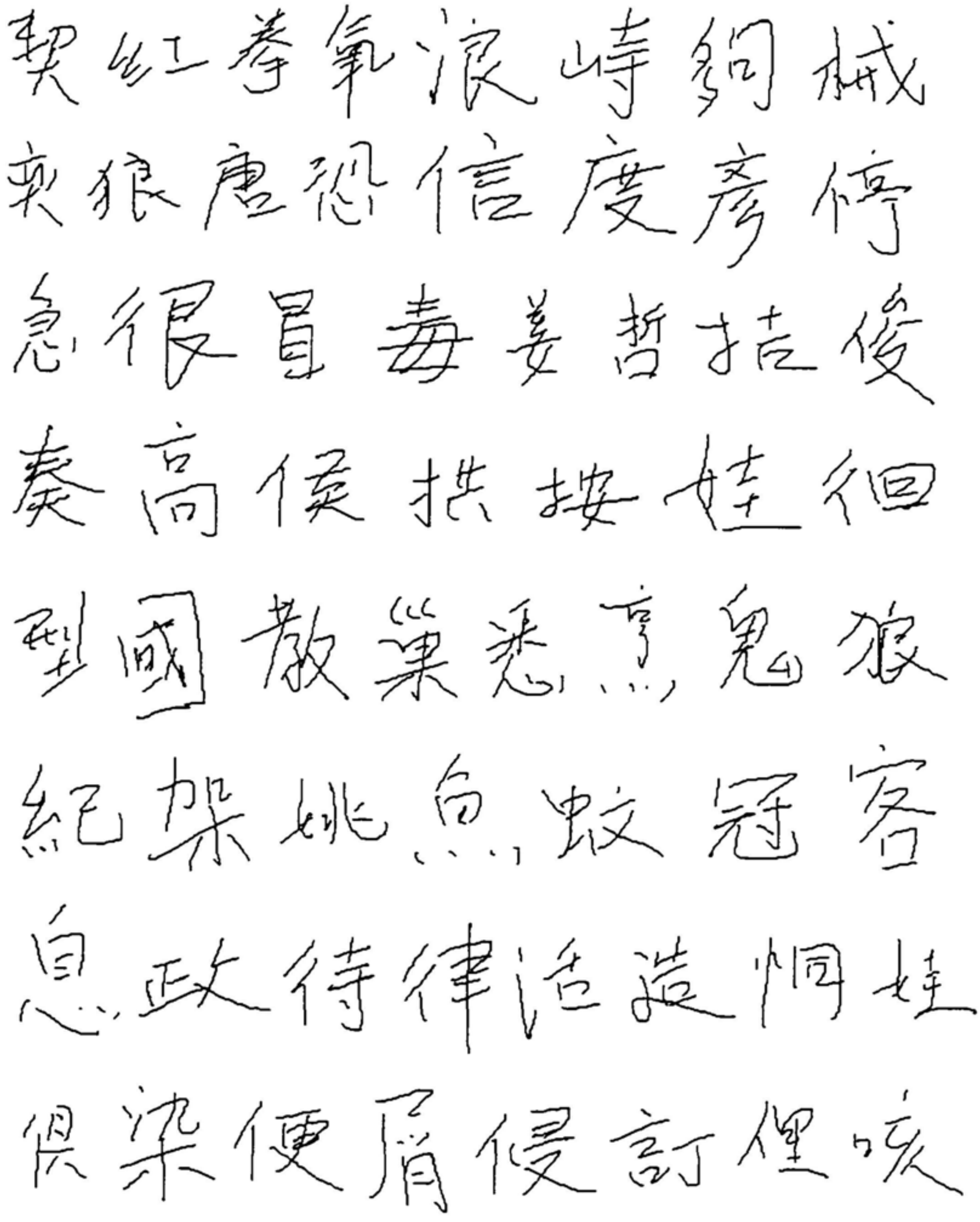


Figure 6.4: Some test characters each having less than three connected strokes.

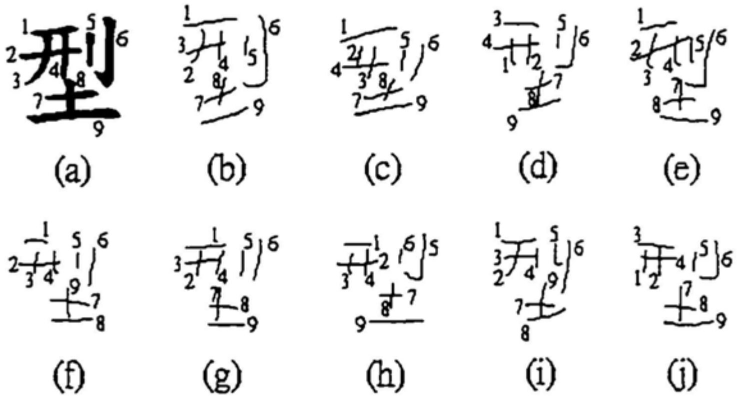


Figure 6.5: A model character (a) and a set of its input handwritten characters (b)–(j) having different stroke orders.

character (f) $8 \leftrightarrow 9$, $9 \leftrightarrow 8$;

character (g) $2 \leftrightarrow 3$, $3 \leftrightarrow 2$, $7 \leftrightarrow 8$, $8 \leftrightarrow 7$;

character (h) $5 \leftrightarrow 6$, $6 \leftrightarrow 5$,

character (i) $2 \leftrightarrow 3$, $3 \leftrightarrow 2$, $8 \leftrightarrow 9$, $9 \leftrightarrow 8$;

character (j) $1 \leftrightarrow 3$, $2 \leftrightarrow 4$, $3 \leftrightarrow 1$, $4 \leftrightarrow 2$, $7 \leftrightarrow 8$, $8 \leftrightarrow 7$;

Let $\delta(\text{model}\, p, \text{input}\, q)$ be the distance between model character p and input character q. Then we have

$$\begin{aligned}\delta(\text{model}\, a, \text{input}\, k) &= \min\{\delta(\text{model 1}, \text{input}\, k), \delta(\text{model 2}, \text{input}\, k), ..., \\ &\qquad \delta(\text{model 300}, \text{input}\, k)\},\end{aligned}$$

where model a is the model character in Fig. 6.5(a), and input k, $k \in \{b, c, d, e, f, g, h\}$ is one of the input characters in Figs. 6.5(b)–(h). This means that these characters are recognized correctly. However, $\delta(\text{model}\, a, \text{input}\, i)$ ranks the 3rd smallest among

$$\{\delta(\text{model 1}, \text{input}\, i), \delta(\text{model 2}, \text{input}\, i), ..., \delta(\text{model 300}, \text{input}\, i)\},$$

and $\delta(\text{model } a, \text{input } j)$ ranks the 5th smallest among

$$\{\delta(\text{model } 1, \text{input } j), \delta(\text{model } 2, \text{input } j), ..., \delta(\text{model } 300, \text{input } j)\}.$$

From these results we see that the string matching method can tolerate some stroke order deviations, but in general, too many stroke order deviations may cause incorrect classifications.

Note that if the WFSM algorithm without the stroke position constraints is used to perform the recognition, $\delta(\text{model } a, \text{input } c)$ will rank the 2nd smallest among

$$\{\delta(\text{model } 1, \text{input } c), \delta(\text{model } 2, \text{input } c), ..., \delta(\text{model } 300, \text{input } c)\},$$

$\delta(\text{model } a, \text{input } d)$ rank the 5th smallest among

$$\{\delta(\text{model } 1, \text{input } d), \delta(\text{model } 2, \text{input } d), ..., \delta(\text{model } 300, \text{input } d)\},$$

$\delta(\text{model } a, \text{input } i)$ rank after the 5th smallest among

$$\{\delta(\text{model } 1, \text{input } i), \delta(\text{model } 2, \text{input } i), ..., \delta(\text{model } 300, \text{input } i)\},$$

and $\delta(\text{model } a, \text{input } j)$ also rank after the 5th smallest among

$$\{\delta(\text{model } 1, \text{input } j), \delta(\text{model } 2, \text{input } j), ..., \delta(\text{model } 300, \text{input } j)\}.$$

Therefore, the geometric position constraints on input strokes are helpful for enhancing the recognition ability of the string matching method.

6.4.2 Segment-Based Recognition

When the primitives are segments and the segment type string of a Chinese character is used to represent it, the recognition method is called segment-based string matching method. 54 Chinese characters each with stroke number between 9 and 11 are selected to be models, which have been used in the experiments for testing the segment-based state space search for graph matching method (see Section 4.5.2). The test data consist of about 2500 Chinese characters written by 6 people. The subjects were asked to write the characters in their own habitual stroke writing orders.

For the input characters each having less than three connected strokes, as shown in Fig. 6.4, the recognition rate is 93.7%. For the character having more connected strokes, as shown in Fig. 6.6, the recognition rate is about 91.5%. The incorrect recognition results are caused mainly by too many stroke order deviations in input characters.

The average time for recognizing a input character is about 0.0036 second on a PC/Pentium at 166 MHz. If the number of model characters were 300, the time would be 0.02 second. The segment-based string matching method is just slightly slower than the stroke-based string matching method, and is about 25 times as fast as the segment-based graph matching method (see Section 4.5.2).

The stroke-based string matching method cannot recognize most of the characters in Fig. 6.6. The reason that the segment-based string matching method has better ability to recognize more freely-written characters is because (1) most connected strokes do not change the types of the segments (not including extra segments) in the connected strokes, and (2) the rules in the segment preprocessing are very efficient for breaking connected strokes and deleting some of the extra segments.

Figure 6.6: Some test characters written freely, all of which are recognized correctly.

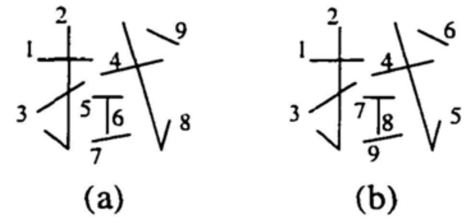


Figure 6.7: (a) A model character with the numbers labeling its standard stroke order. (b) The same character as (a) but with different stroke order.

6.5 Extension of the String Matching Method

A characteristic of string-matching-based approaches is that input characters are required to be written (basically) in their standard stroke orders. Our string matching method can tolerate some stroke order deviations, but too many stroke order deviations will cause incorrect recognition. Consider the character in Fig. 6.7(a), where the numbers labeling the strokes denote the standard order of writing of the character. However, some people write the character in the stroke order as illustrated in Fig. 6.7(b). Because of too many stroke order deviations in the character, it cannot be recognized by the string matching method.

A scheme to solve this problem is to represent the character with two primitive (say, stroke) type strings:

$$S_1 = s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9, \quad S_2 = s'_1 s'_2 s'_3 s'_4 s'_5 s'_6 s'_7 s'_8 s'_9,$$

where s_{1-9} are the stroke types of corresponding strokes in Fig. 6.7(a) and s'_{1-9} are the stroke types of corresponding strokes in Fig. 6.7(b). If the time for recognizing an input character with a low-level CPU is acceptable after enlarging the model string base, the string matching method can be used without any modification. If not, we have to seek some approaches to reduce computational

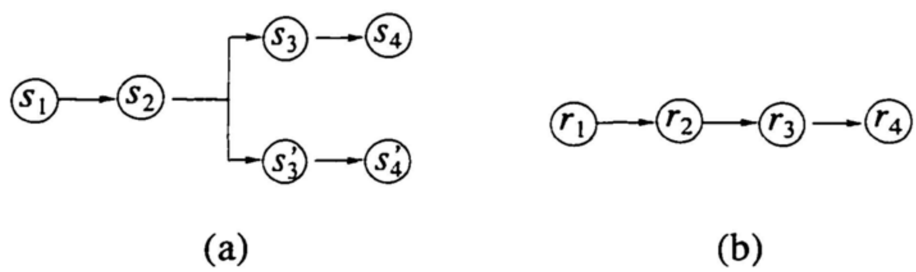


Figure 6.8: (a) Combining strings S_1' and S_2' together. (b) An input string R.

time. In the following, a scheme is proposed to reach this goal.

Considering the characters in Fig. 6.7 and the two strings S_1 and S_2, we have $s_i = s_i'$, $i = 1, 2, 3, 4$, i.e., a part of S_1 is the same as a part of S_2 at the same positions. For convenient description below, we use two shorter strings $S_1' = s_1 s_2 s_3 s_4$ and $S_2' = s_1 s_2 s_3' s_4'$ instead of S_1 and S_2. S_1' and S_2' can be combined together as shown in Fig. 6.8(a). Let $R = r_1 r_2 r_3 r_4$ be an input string (Fig. 6.8(b)). Two networks for finding $\delta(S_1', R)$ and $\delta(S_2', R)$ are shown in Figs. 6.9(a) and (b), respectively, where $id1$ is the stroke insertion or deletion cost. Using the dynamic-programming algorithm (the WFSM algorithm), the computational time for finding the least cost of a path among all the paths from node (0,0) to node (4,4) in network (a) and from node (0,0)' to node (4,4)' in network (b) is proportional to the number of arcs in the two networks, which equals 112.

Now we combine the two networks together to form a network as illustrated in Fig. 6.9(c), where not all the arc costs are given for simplicity, and the costs of the two arcs joining the end (target) node E are set to 0. Comparing Figs. 6.9(a) and (b) with Fig. 6.9(c), we see that for a path P_1 from node (0,0) to node (4,4) in network (a) or from node (0,0)' to node (4,4)' in network (b), there exists a corresponding path P_1' from node (0,0) to node E in network (c) such that

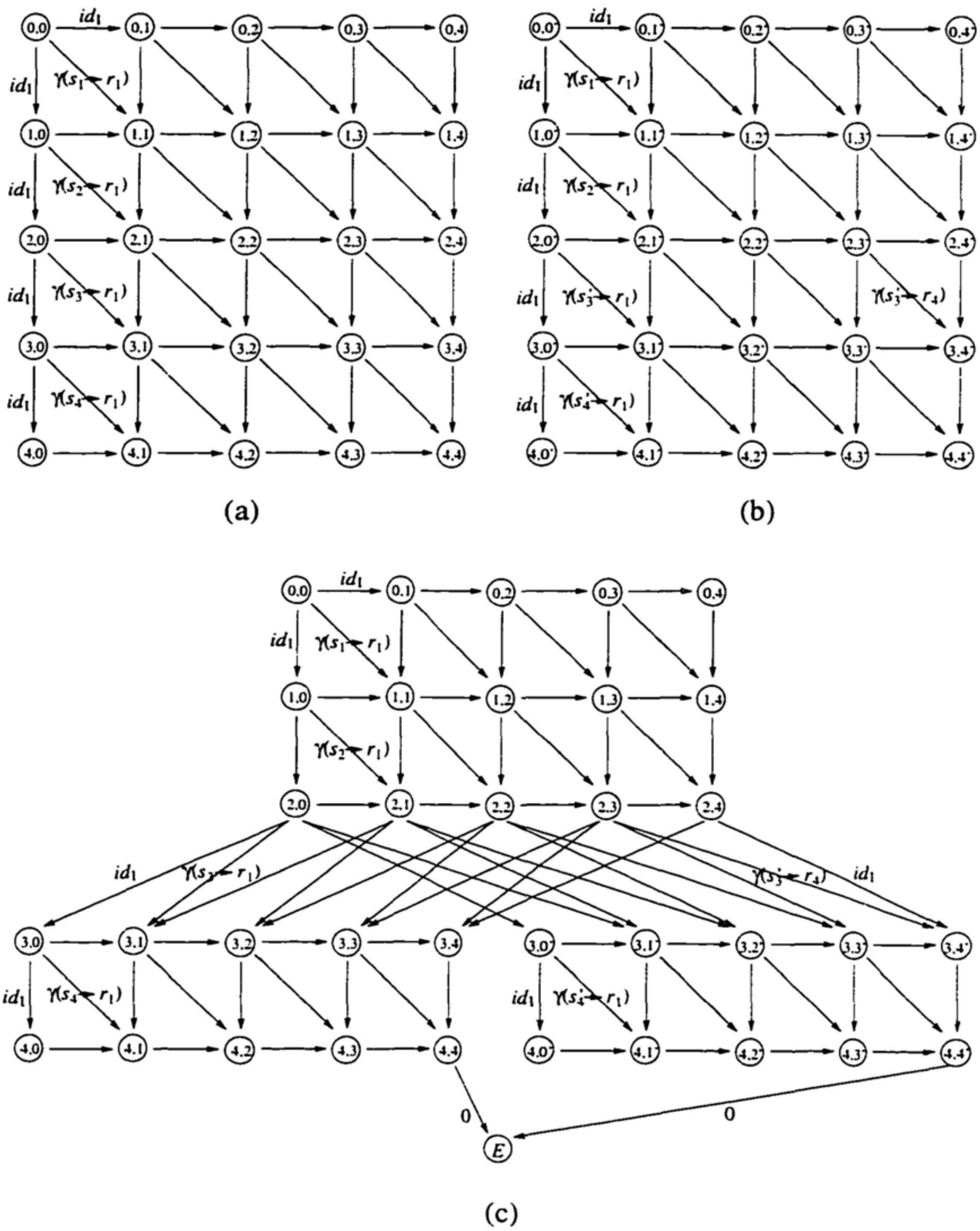


Figure 6.9: (a) A network for calculating $\delta(S'_1, R)$. (b) A network for calculating $\delta(S'_2, R)$. (c) A network obtained by combining (a) and (b).

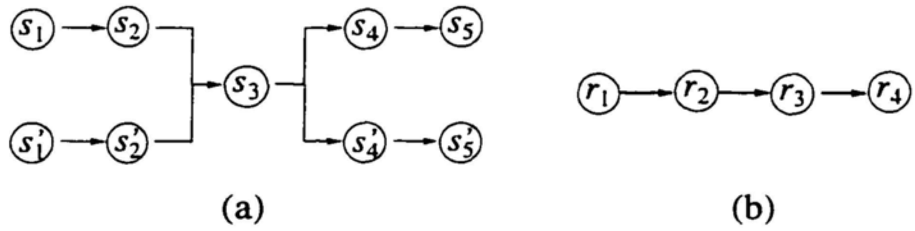


Figure 6.10: (a) Combining strings S_1, S_2, S_3 and S_4 together. (b) An input string R.

$cost(P_1) = cost(P'_1)$, where $cost(P'_1)$ is the sum of all the costs of the arcs in P'_1.

Let P' be any path from node (0,0) to node E in network (c). Then the problem of finding $\min\{\delta(S'_1, R), \delta(S'_2, R)\}$ respectively in networks (a) and (b) is now transformed to the problem of finding $\min\{cost(P')\}$ in network (c). Network (c) is also a multi-stage one. Obviously, this optimization problem can be solved by a dynamic-programming algorithm. As the arc number in network (c) is 84 (< 112), the new problem requires less time to be solved.

Let us consider a more complicated example. Suppose a model character is represented by the following four strings:

$$S_1 = s_1 s_2 s_3 s_4 s_5, \quad S_2 = s_1 s_2 s_3 s'_4 s'_5,$$

$$S_3 = s'_1 s'_2 s_3 s_4 s_5, \quad S_4 = s'_1 s'_2 s_3 s'_4 s'_5,$$

Combining S_{1-4} as shown in Fig. 6.10(a). Now we want to find

$$\min\{\delta(S_1, R), \delta(S_2, R), \delta(S_3, R), \delta(S_4, R)\},$$

where $R = r_1 r_2 r_3 r_4$ is an input string (Fig. 6.10(b)). The WFSM algorithm can be used to calculate $\delta(S_1, R), \delta(S_2, R), \delta(S_3, R)$ and $\delta(S_4, R)$, respectively. The corresponding four networks (a)–(d) are shown in Fig. 6.11. There are 276 arcs in the four networks. Combining these networks together, we obtain the

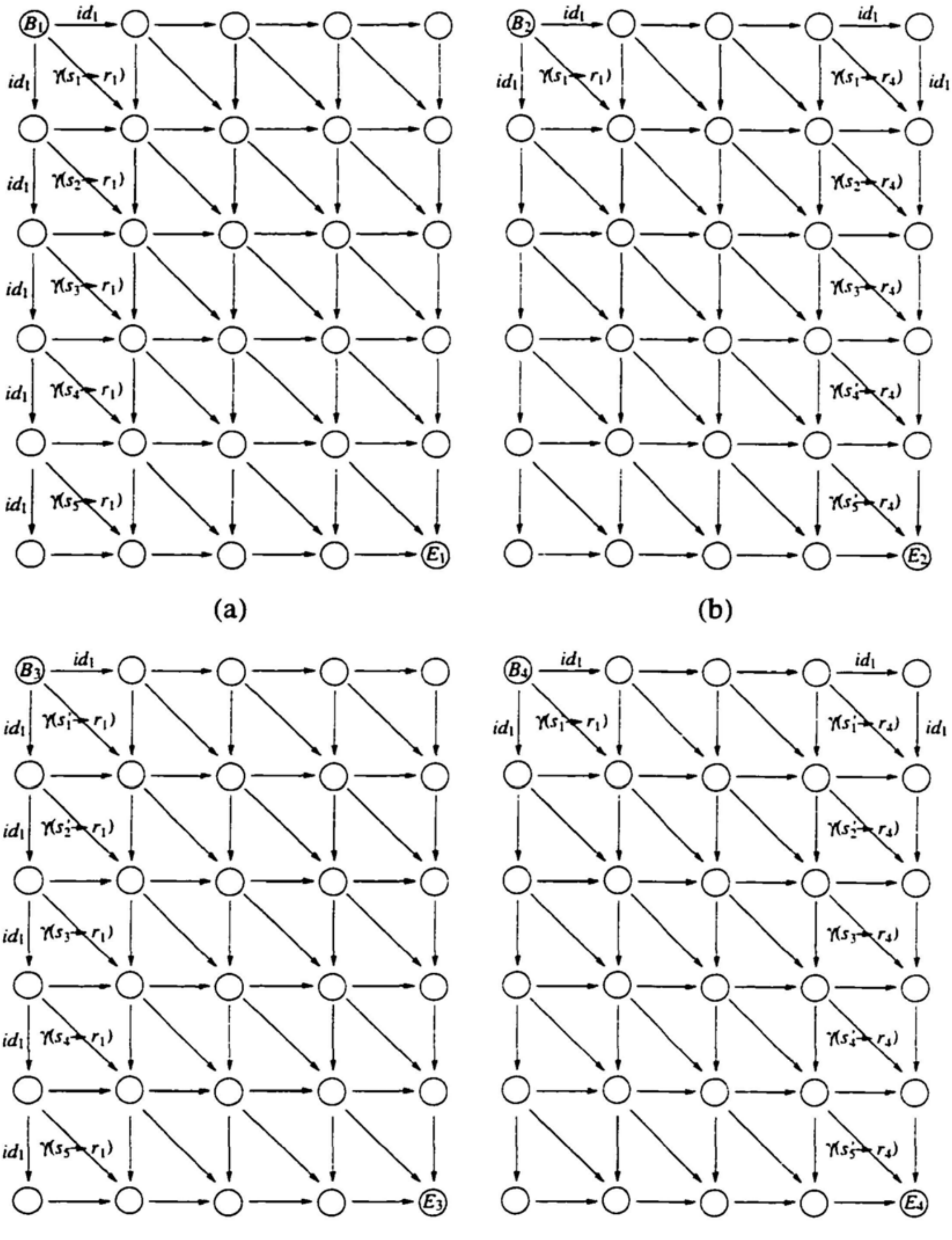


Figure 6.11: Four networks (a)–(d) for calculating $\delta(S_1, R)$, $\delta(S_2, R)$, $\delta(S_3, R)$, and $\delta(S_4, R)$, respectively.

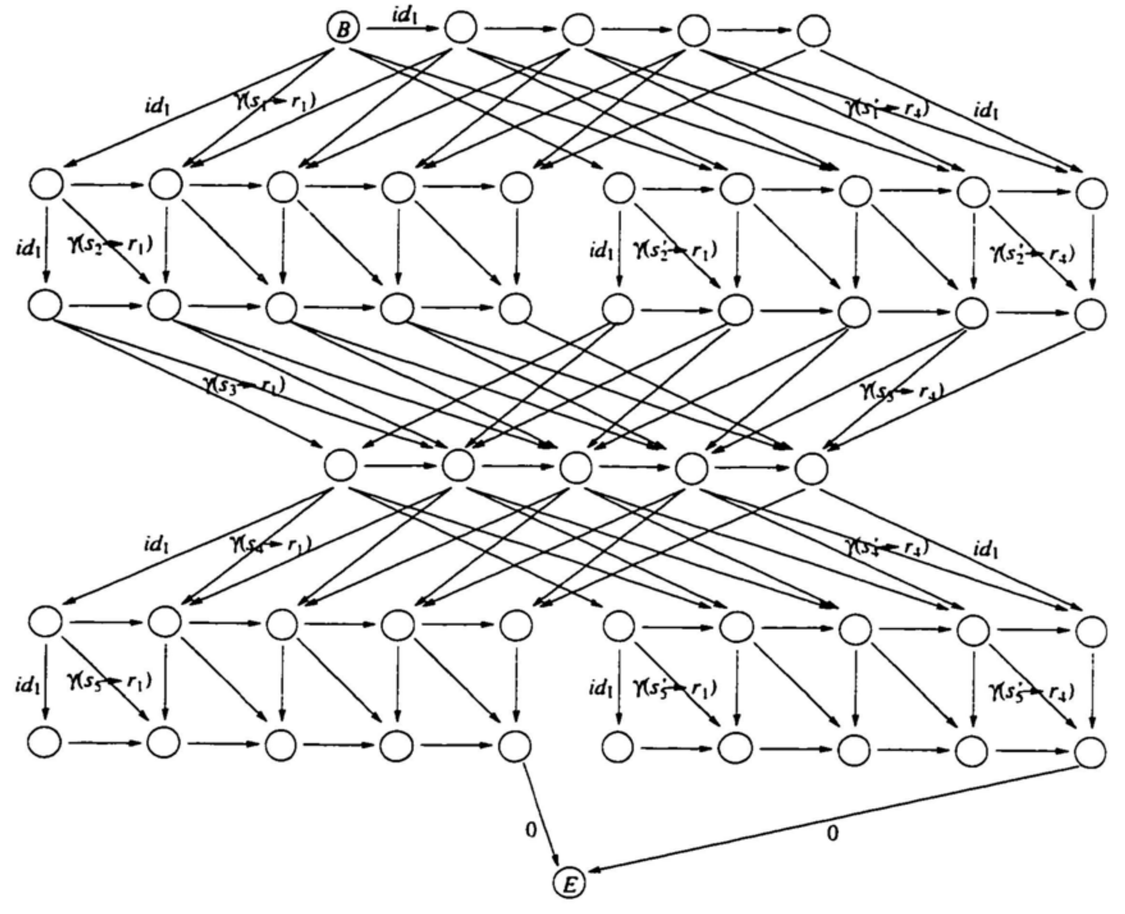


Figure 6.12: A network obtained by combining networks (a)–(d) in Fig. 6.11.

network in Fig. 6.12, in which there are 132 arcs.

Comparing Fig. 6.11 with Fig. 6.12, it is not difficult to see that for a path P_2 from node B_1 to node E_1 in network (a), from node B_2 to node E_2 in network (b), from node B_3 to node E_3 in network (c), or from node B_4 to node E_4 in network (d), there exists a path P_2' from node B to node E in the network in Fig. 6.12 such that $cost(P_2) = cost(P_2')$. Therefore, using a dynamic-programming algorithm and the network, we can also obtain the solution to the problem of finding $\min\{\delta(S_1, R), \delta(S_2, R), \delta(S_3, R), \delta(S_4, R)\}$.

6.6 Summary

In this chapter, we have proposed a fast string matching method for on-line Chinese character recognition, which incorporates the geometric position constraints of primitives of Chinese characters into Wagner and Fischer's string matching algorithm. The experiments show that when input characters are written not having great stroke order deviations, the method can obtain good recognition results. Moreover, the segment-based method may tolerate more cursive handwriting than the stroke-based one. To allow more stroke order deviations for some characters, using two or more strings to represent one of these model characters is necessary. In this case, we suggest a scheme to save computational time, which combines two or more separate networks into one and employs a dynamic-programming procedure to solve the shortest path problem.

The string matching method is very fast and requires small memory space. When the primitives used are segments, it runs 25 times as fast as the segment-based graph matching method does. Therefore, this method can be implemented on low-end CPUs.

Chapter 7

Conclusions and Suggestions

7.1 Contributions of this Thesis

The aim of this thesis is to derive efficient methods for on-line Chinese character recognition. We have addressed the three aspects: preprocessing of input handwriting, representations of Chinese characters, and recognition methods. Now we summarize our major contributions and results as follows.

1. **Preprocessing of input handwriting**

    - In order to facilitate the recognition of the types of strokes and segments, an input stroke is represented with a polyline by using the efficient polyline fitting algorithm and the line merging algorithm. This approach can handle some handwriting noise (such as wild points and hooks) well.

    - A method for recognizing the types of strokes with more than two segments is proposed, which consists of three procedures: normalization of strokes, extraction of stroke chain code strings, and matching between input code strings and model code strings. The experimental results show that the method works well. It can be used not only in stroke-based but also in segment-based on-line recognition of Chinese characters.

- Some rules are presented to detect most of frequently-occurred connected strokes and then delete the extra segments in such strokes. These rules make our recognition methods have the ability to recognize more freely-written Chinese characters

2. **Representations of Chinese characters**

we have formally defined the complete relational graphs and the distances for measuring the similarity between two graphs. With such graphs, we have proposed the stroke-based and segment-based spatially-temporally relational representations for on-line inputted Chinese characters. We have also dealt with assigning costs to node and arc correspondences for calculating the graph matching distances.

The stroke-based representations may be used to recognize relatively neat Chinese character handwriting while the segment-based representations will ease the recognition of more freely written characters but make a recognition method need more computational time and memory space. These representations have the following advantages:

- The representations incorporate the human knowledge of Chinese characters and can reflect their features well (except some very similar character pairs). The novel "don't care", "should" and "must" relational features allow us to represent unstable, stable and very stable primitive relations conveniently. Relations between **any** two primitives give much information and are very beneficial to the matching procedures.

- The proposed complete graph representations are directly based on strokes or segments. To obtain the representations, examining whether a stroke or segment belongs to some component is not required. However, the graph representations in [18, 68] need to correctly extract components of Chinese characters first. The recognition method based on the graph representation in [13] also needs to find components before performing recognition of a character. In fact, wide stroke type variations and connected strokes make it very difficult to extract components of Chinese characters at a high success rate. In [16], the authors adopted only the relations between segments within the same components in their graph representation. This results in two shortcomings: (1) some relations represented in an input graph may not appear in its corresponding model graph, and vice versa; (2) most of the relation information between segments are not utilized.

- The spatial and temporal relations between primitives are, at the first time, unified into the graph representations, which fully captures the on-line information of handwriting. The use of the primitive order relations enhances the discrimination ability of the representations and helps to speed up the graph matching. Because of the "don't care" feature, the representations can tolerate common stroke order deviations.

- If the weight w_4 in (3.11) is set to 0 in graph match, then the stroke order relations will be ignored and our recognition methods will allow writing a Chinese character in any stroke orders.

The disadvantage of our representations is that the creation of a model character base is a relatively heavy task. It can be eased by constructing the graphs of components of Chinese characters first and then combining several component graphs to form the whole graph of a character. We will discuss this problem in the next section.

3. A state space search method

We have formulated the graph matching as a state space search problem. The optimal matching between two graphs is then equivalent to finding the best goal node in a search tree. To obtain good search efficiency, we have used the A^* algorithm to perform heuristic search, and proposed the following schemes to speed up the A^*.

- A heuristic function h, which has been proved to be a lower bound on h^* and monotonic, is defined to make the A^* expand fewer nodes in a search tree.

- A tree pruning strategy, which employs the geometric position features of strokes (or segments) of Chinese characters to prune a search tree, is proposed to let the A^* have more or less the function of a bird's eye view, in other words, to let the A^* avoid searching the nodes that have very little chance to be located in the optimal path from the initial node to the best goal node in a tree.

- Two new criteria, together with the original one, are presented to stop the A^* by utilizing the monotone of the evaluation function of the A^*. They are based on the fact that in Chinese character recognition, finding the final optimal matching between two dissimilar characters is not necessary if we have known their distance is great enough.

The experimental results show that the recognition speeds of our stroke-based and segment-based recognition methods are sufficiently fast in practical applications, even if the frequently-used 5000 or more Chinese characters are added. In common recognition of input Chinese characters (the first phase), the methods can tolerate most of the stroke order variations due to the "don't care" temporal relations between strokes (segments). To deal with a character with great stroke order deviations, the re-classification stage (the second phase) can be effected (without the need to write the character again), which ignores the stroke (segment) order information and takes a little more time to perform a recognition. Therefore, the methods are stroke order free. The results also show that the segment-based method can recognize the handwritten characters having many connected strokes, so it is stroke number free too.

We have made some comparisons between our segment-based method and several other studies published recently in international journals. Considering their recognition rates, recognition time and tolerances of stroke order and stroke number variations, we see that out method is very promising.

4. **A two-layer assignment method**

Finding segment correspondences between two characters is formulated as a weighted bipartite graph minimum cost complete matching problem, which corresponds to an assignment problem (in layer 2) and can be solved by the Hungarian method. The cost matrix of this assignment problem is derived by the assignment problems in layer 1. The costs of segment type correspondences and relation correspondences between two characters are used to generate the cost matrixes in layer 1. To save the computational time, a lower bound estimate is proposed to approximately solve each assignment problem in layer 1. In addition, the geometric position features of model characters are used to avoid wasting computation on unreasonable segment correspondence costs. These two schemes reduce the complexity of the method from $O(n^5)$ to $O(n^3)$.

The experimental results are satisfactory. Compared with the segment-based state space search method, the two-layer assignment method runs slightly faster and has a little lower recognition rate when they were used to recognize the characters with an approximate degree of deformation. The two-layer assignment method is also stroke order and stroke number free.

5. A fast string matching method

Incorporating the geometric position constraints of strokes (or segments) of Chinese characters into Wagner and Fischer's string matching algorithm, we have proposed a fast string matching method for on-line Chinese character recognition. The experiments show that when input characters are written not having great stroke order deviations, the method can obtain good recognition results. Moreover, the segment-based string matching method can recognize cursive handwriting, so it is stroke number free.

To allow more stroke order deviations for some characters, using two or more strings to represent one of these model characters is necessary. In this case, we have proposed a scheme to save computational time, which combines two or more separate networks into one and employs a dynamic-programming procedure to solve the shortest path problem.

The string matching method is very fast and requires small memory space. When the primitives are segments, it runs 25 times as fast as the segment-based graph matching method does. Therefore, this method can be used in the products that are required to be able to recognize on-line inputted Chinese characters but equipped with low-end CPUs and small memory, such as portable electronic diaries, electronic Chinese-English dictionaries, and multi-functional telephones.

7.2 Suggestions for Further Research

Several methods for on-line Chinese character recognition have been proposed in this thesis. They are ready for practical applications. However, to develop a whole perfect system, more work needs to be done. Below we would suggest some directions for further research.

1. **Creation of a model graph base**

The creation of a model graph base for out state space search method or two-layer assignment method is a relatively heavy task. There may be three approaches to this goal.

- Completely based on the human knowledge of Chinese characters, this work is done by the people who are familiar with Chinese character handwriting. In this case, machine learning is not necessary, but the creation and modification of a base are heavy and boring.
- The second scheme consists of two steps. In step 1, first, for each model character, collect a set of learning samples which are written in correct

stroke orders and without connected strokes, and then with each set of learning samples, build corresponding model graph by examining the primitive types and spatially-temporally relations between primitives using a simple program. The relation features generated in step 1 only contain the "don't care" and "should" features. In step 2, some of the "should" features are changed to "must" features by people who are familiar with Chinese character handwriting. This scheme needs to collect a large set of Chinese characters. Changing and modifying the features by people are also a heavy and boring task.

- The third approach is based on the fact that Chinese characters are constructed by a set of components (radicals). Because the number (< 250) of the components are much less than that of frequently-used Chinese characters and the stroke number of each component is less than seven,[1] building the complete relational graphs of these components is much easier.

  Using the components, we can form a Chinese character on the screen of a computer. Then by moving the components, we obtain the "don't care" and "should" relations between the primitives of two components. To further obtain the "must" features, we may select two sets of primitives and then choose one of the "must" features ("left of", "right of", "above", and "below") to be the relations between the primitives in the two sets. The movement of components can be completed by one using a mouse or by a program according to some rules. We have estimated that a model graph can be generated within a minutes. This scheme makes the creation

[1]Traditionally, there are components with more than six strokes. These components can be constructed by the components with fewer strokes.

of a model base easier. In addition, if we change the relations of some component, the modification of relations for all the characters containing this component may be made automatically. We are now developing a tool for this goal.

2. **Preclassification and detailed recognition problems**

There are about 4000 Chinese characters which cover more than 99.9% of the daily-used ones [102]. To extend our state space search method or two-layer assignment method to recognizing 4000 or more Chinese characters, a preclassification stage is required for saving computational time. The numbers of segments of Chinese characters is a useful features for choosing primary model character candidates. Estimating some possible components (radicals) in input characters is also a common approaches to preclassification. Many methods have been proposed for the preclassification purpose in on-line or off-line Chinese character recognition [14, 20, 21, 22, 50, 52, 57, 67, 93, 102, 103]. By comparing the performances of these methods and combining some features of them, it is not difficult to obtain a good preclassification method.

There are some pairs of very similar characters, such as (土,士), (日,曰) and (己,已). The methods proposed in this thesis, like other general recognition methods, cannot distinguish them. Adding an *ad hoc* detailed recognition stage, we may solve this problem.

3. **Improvement of performance of the proposed methods**

- In the graph representations of Chinese characters, using more spatial relations between some primitives will enhance its ability to distinguish between very similar characters. The new relations may be the relative spatial relations between the beginning points of two primitives, between the beginning point of a primitive and the end point of another primitive, and so on. Of course, that will increase the work load to build a graph base.

- Finding more precise heuristic function h is a way to further speed up the A^* algorithm. Besides the primitive types, we might use spatial and temporal information among primitives to estimate h^*. In this case, we have to test whether the total search time is reduced because more computation for calculating h is needed.

- In the two-layer assignment method, the spatial and temporal relations between segments are employed to estimate the cost matrixes of the assignment problems in layer 1. An alternative way is to use the information of segment coordinates. It is worth making a comparison between them to see which is better in the future.

4. **Extension to off-line Chinese character recognition**

Off-line Chinese character recognition is a more difficult task than on-line Chinese character recognition. The existing methods can only recognize very neatly-written Chinese characters. By ignoring the temporal relations between primitives, our state space search method and two-layer assignment method can be extended to off-line Chinese character recognition if an approach for extracting the segments of a Chinese character is available. Many researchers have investigated the segment or stroke extraction problem [1, 2, 15, 24, 49, 53, 73].

Therefore, we expect that our methods will also yield good results when they are applied to off-line Chinese character recognition.

Bibliography

[1] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta. Processing of off-line handwritten text: polygonal approximation and enforcement of temporal information. *CVGIP: Graphical models and Image Processing*, 56:324–335, 1994.

[2] I. S. I. Abuhaiba, M. J. J. Holt, and S. Datta. Processing of binary images of handwritten text documents. *Pattern Recognition*, 29:1161–1177, 1996.

[3] H. Arakawa. On-line recognition of handwritten characters — Alphanumerics, Hiragana, Katakana, Kanji. *Pattern Recognition*, 16:9–16, 1983.

[4] H. Arakawa, K. Odaka, and I. Masuda. On-line recognition of handwritten characters — Alphanumerics, Hiragana, Katakana, Kanji. In *Proc. 4th Int. Joint Conf. Pattern Recognition*, pages 810–812, Nov. 1978.

[5] E. Balas and C. S. Yu. Finding the maximum clique in an arbitrary graph. *SIAM J. Comput.*, 15:1054–1068, 1986.

[6] D. H. Ballard and C. M. Brown. *Computer Vision*. Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[7] P. Berman and G. Schnitger. On the complexity of approximating the independent set problem. In *Lecture Notes in Computer Science 349*, pages 256–267. Springer-Verlag, 1989.

[8] D. P. Bertsakas. The auction algorithm: a distributed relaxation method for the assignment problem. *Annals of Operations Research*, 14:105–123, 1988.

[9] L. Bolc and J. Cytowski. *Search Methods for Artificial Intelligence*. Academic Press, San Diego, CA, 1992.

[10] C. Bron and J. Kerbosch. Finding all cliques of an undirected graph. *Comm. ACM*, 16:575–577, 1973.

[11] H. Bunke and G. Allermann. Inexact graph matching for structural pattern recognition. *Pattern Recognition Letters*, 1:245–253, 1983.

[12] R. Carraghan and P. M. Pardalos. An exact algorithm for the maximum clique problem. *Operations Research Letters*, 9:375–382, 1990.

[13] K. P. Chan and Y. S. Cheung. Fuzzy-attribute graph with application to Chinese character recognition. *IEEE Trans. Syst. Man Cybern.*, 22:153–160, 1992.

[14] H. D. Chang and J. F. Wang. Preclassification for handwritten Chinese character recognition by a peripheral shape coding method. *Pattern Recognition*, 26:711–719, 1993.

[15] H.-D. Chang and J.-F. Wang. A robust stroke extraction method for handwritten Chinese characters. In H. Bunke, P. S. P. Wang, and H. S.

Baird, editors, *Document Image Analysis*, pages 1223–1239. World Scientific Publishing, Singapore, 1994.

[16] J. W. Chen and S. Y. Lee. On-line handwritten Chinese character recognition via a fuzzy attribute representation. *Image and Vision Computing*, 12:669–681, 1994.

[17] K.-J. Chen, K.-C. Li, and Y.-L. Chang. A system for on-line recognition of Chinese characters. *Computer Processing of Chinese & Oriental Languages*, 3:309–318, 1988.

[18] L. H. Chen and J. R. Lieh. Handwritten character recognition using a 2-layer random graph model by relaxation matching. *Pattern Recognition*, 23:1189–1205, 1990.

[19] W.-T. Chen and T.-R Chou. A hierarchical deformation model for on-line cursive script recognition. *Pattern Recognition*, 27:205–219, 1994.

[20] K.-S. Chou, K.-C. Fan, and T.-I. Fan. Horizontal segments and peripheral features for use in coarse classification of Chinese characters. In *Proc. Int. Symp. Multi-Technology Information Processing*, pages 343–348, Taiwan, 1996.

[21] K. S. Chou, K. C. Fan, T. I. Fan, C. K. Lin, and B. S. Jeng. Knowledge model based approach in recognition of on-line Chinese characters. *IEEE J. Selected Areas Communi.*, 12:1566–1574, 1994.

[22] S.-L. Chou and W.-H. Tsai. On-line Chinese character recognition through stroke-segment matching using a new discrete iteration scheme. *Computer Processing of Chinese and Oriental Languages*, 7:1–20, 1993.

[23] W. J. Christmas, J. Kittler, and M. Petrou. Structural matching in computer vision using probabilistic relaxation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17:749–764, 1995.

[24] C.-T. Chuang and L. Y. Tseng. A heuristic algorithm for the recognition of printed Chinese characters. *IEEE Trans. Syst. Man Cybern.*, 25:710–717, 1995.

[25] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* The MIT Press, MA, 1994.

[26] F. Depiero, M. Trivedi, and S. Serbin. Graph matching using a direct classification of node attendance. *Pattern Recognition*, 29:1031–1048, 1996.

[27] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* Wiley-Interscience, New York, 1973.

[28] M. A. Eshera. *Image Understanding by Hierarchical Symbolic Representation and Inexact Matching of Attributed Graphs.* PhD thesis, Electrical Engineering, Purdue University, 1985.

[29] L. R. Foulds. *Graph Theory Applications.* Springer-Verlag, New York, 1992.

[30] K. S. Fu. *Syntactic Pattern Recognition and Applications.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[31] K. S. Fu, Z. X. Cai, and G. Y. Xu. *Artificial Intelligence and its Applications.* The Publishing House of the Tsinghua University, Beijing, China, 1987.

[32] H. N. Gabow and R. E. Tarjan. Faster scaling algorithms for network problems. *SIAM J. Comput.*, 18:1013–1036, 1989.

[33] S. Gold and A. Rangarajan. A graduated assignment algorithm for graph matching. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18:377–388, 1996.

[34] L. Goldfarb. A unified approach to pattern recognition. *Pattern Recognition*, 17:575–582, 1984.

[35] L. Goldfarb. A new approach to pattern recognition. In L. N. Kanal and A. Rosenfeld, editors, *Progress in Pattern Recognition 2*, pages 241–402. Elsevier Science Publishers B. V., North-Holland, 1985.

[36] V. K. Govindan and A. P. Shivaprsad. Character recognition — a review. *Pattern Recognition*, 23:671–683, 1990.

[37] S. Hanaki, T. Temma, and H. Yoshida. An on-line character recognition aimed at a substitution for a billing machine keyboard. *Pattern Recognition*, 8:63–71, 1976.

[38] Y. Hidai, K. Ooi, and Y. Nakamura. Stroke re-ordering algorithm for on-line handwritten character recognition. In *Proc. 8th Int. Conf. Pattern Recognition*, pages 934–936, 1986.

[39] F. S. Hillier and G. J. Lieberman. *Introduction to Operations Research.* McGraw-Hill, New York, 1990.

[40] A. J. Hsieh, K. C. Fan, and T. I. Fan. Bipartite weighted matching for on-line handwritten Chinese character recognition. *Pattern Recognition*, 28:143–151, 1995.

[41] R. A. Hummel and S. W. Zucker. On the foundations of relaxation labeling processes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 5:267–287, 1983.

[42] K. Ikeda, T. Yamamura, Y. Mitamura, S. Fujiwara, Y. Tominaga, and T. Kiyono. On-line recognition of hand-written characters utilizing positional and stroke vector sequences. In *Proc. 4th Int. Joint Conf. Pattern Recognition*, pages 813–815, Nov. 1978.

[43] S. Impedovo, B. Marangelli, and V. L. Plantamura. Real-time recognition of handwritten numerals. *IEEE Trans. Syst. Man Cybern.*, 6:145–148, 1976.

[44] K. B. Irani and S. I. Yoo. A methodology for solving problems: problem modeling and heuristic generation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 10:676–686, 1988.

[45] M. S. Kamel, H. C. Shen, A. K. C. Wong, and R. I. Campeanu. System for the recognition of human faces. *IBM Systems Journal*, 32:307–320, 1993.

[46] R. M. Karp. Reducibility among combinatorial problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. Plenum Press, New York, 1972.

[47] J. Kobler, U. Schoning, and J. Toran. *The Grapg Isomorphism Problem: its Structural Complexity*. Birkhauser, Boston, 1993.

[48] B. Kreko. *Linear Programming*. Pitman, London, 1968.

[49] K.-M. Ku and P. P. K. Chiu. Fast stroke extraction method for handwritten Chinese characters by cross region analysis. *Electronics Letters*, 30:1210–1212, 1994.

[50] T. Kumamoto, K. Toraichi, T. Horiuchi, K. Yamamoto, and H. Yamada. On speeding candidate selection in handprinted Chinese character recognition. *Pattern Recognition*, 24:793–799, 1991.

[51] P. Kuner and B. Ueberreiter. Pattern recognition by graph matching — combinatorial versus continuous optimization. *Int. J. Pattern Recognition and Artificial Intelligence*, 3:527–542, 1988.

[52] S.-R. Lay, C.-H. Lee, N.-J. Cheng, C.-C. Tseng, B.-S. Jeng, P.-Y. Ting, Q.-Z. Wu, and M.-L. Day. On-line Chinese character recognition with effective candidate radical and candidate character selections. *Pattern Recognition*, 29:1647–1659, 1996.

[53] S. Lee and J. C. Pan. Offline tracing and representation of signatures. *IEEE Trans. Syst. Man Cybern.*, 22:755–771, 1992.

[54] S. Z. Li. Matching: invariant to translations, rotations and scale changes. *Pattern Recognition*, 25:583–594, 1992.

[55] C.-K. Lin and B.-S. Jeng. On-line recognition of handwritten Chinese characters and alphabets. In *Proc. Int. Conf. Acoustics, Speech, Signal Processing*, pages 2029–2032, 1990.

[56] C. R. Lin, K. C. Fan, and F. T. P. Lee. On-line recognition by deviation-expansion model and dynamic programming matching. *Pattern Recognition*, 26:259–268, 1993.

[57] T. Z. Lin and K. C. Fan. Coarse classification of on-line Chinese characters via structure feature-based method. *Pattern Recognition*, 27:1365–1377, 1994.

[58] J.-Z. Liu, W. K. Cham, and M. M. Y. Chang. Online Chinese character recognition using attributed relational graph matching. *IEE Proc. Vision Image Signal Process.*, 143:125–131, 1996.

[59] J.-Z. Liu, W. K. Cham, and M. M. Y. Chang. Stroke order and stroke number free on-line Chinese character recognition using attributed relational graph matching. In *Proc. Int. Conf. Pattern Recognition*, pages 259–263, 1996.

[60] J.-Z. Liu, W. K. Cham, and M. M. Y. Chang. On-line Chinese character recognition by incorporating human knowledge. *Int. J. of Uncertainty, Fuzziness, and Knowledge-Based Systems*, 5:13–29, 1997.

[61] J.-Z. Liu, W. K. Cham, and M. M. Y. Chang. A spatial-temporal approach to on-line Chinese character recognition. In *Proc. Int. Conf. Computer Processing Oriental Languages*, pages 258–261, 1997.

[62] J.-Z. Liu, W. K. Cham, and M. M. Y. Chang. A spatial-temporal method for on-line Chinese character recognition. *Communications of COLIPS*, 7:31–39, 1997.

[63] J.-Z. Liu, W. K. Cham, and M. Y. Chang. On-line Chinese character recognition with attributed relational graph matching. In R. T. Chin, H. H. S. Ip, A. C. Naiman, and T. C. Pong, editors, *Image Analysis Applications and Computer Graphics*, pages 189–196. Springer, 1995.

[64] Y. J. Liu and J. W. Tai. An on-line Chinese character recognition system for handwritten in Chinese calligraphy. In *From Pixel to Features III — Frontiers in Handwriting Recognition*, pages 87–99. Elsevier Science Publishers B. V., 1992.

[65] S.-C. Loh, C.-W. Chan, and S.-C Chan. On-line recognition of handwritten Chinese characters. In *Int. Conf. Computer Processing Chinese & Oriental Languages*, pages 58–61, 1988.

[66] W. W. Loy and I. D. Landau. An on-line procedure for recognition of handprinted alphanumeric characters. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4:422–427, 1982.

[67] H. Lu and P. Yang. A preclassification method for Chinese character recognition based on peripheral stroke structure. *Communications of COLIPS*, 2:73–79, 1992.

[68] S. W. Lu, Y. Ren, and C. Y. Suen. Hierarchical attributed graph representation and recognition of handwritten Chinese characters. *Pattern Recognition*, 24:617–632, 1991.

[69] A. Marzal and E. Vidal. Computation of normalized edit distance and applications. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15:926–932, 1993.

[70] K. G. Murty. *Network Programming.* Prentice-Hall, Englewood Cliffs, New Jersey, 1992.

[71] N. J. Nilsson. *Principles of Artificial Intelligence.* Tioga Publishing Co., Palo Alto, CA, 1980.

[72] K. Odaka, H. Arakawa, and I. Masuda. On-line recognition of handwritten characters by approximating each stroke with several points. *IEEE Trans. Syst. Man Cybern.*, 12:898–903, 1982.

[73] H. Ogawa and K. Taniguchi. Thinning and stroke segmentation for handwritten Chinese character recognition. *Pattern Recognition*, 15:299–308, 1982.

[74] J. B. Orlin and R. K. Ahuja. New scaling algorithms for the assignment and minimum cycle mean problems. *Mathmatical Programming*, 54:41–56, 1992.

[75] C. H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization — Algorithms and Complexity.* Prentice-Hall, Englewood Cliffs, New Jersey, 1982.

[76] J. Pearl. Knowledge versus search: a quantitative analysis using a*. *Artificial Intelligence*, 20:1–13, 1983.

[77] J. Pearl. *Heuristics: Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley, Reading, MA, 1984.

[78] J. Pearl. Some recent results in heuristic search theory. *IEEE Trans. Pattern Anal. Mach. Intell.*, 6:1–13, 1984.

[79] A. Sanfeliu and K.-S. Fu. A distance measure between attributed relational graphs for pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13:353–362, 1983.

[80] S. Sarkar and K. L. Boyer. Computional structure for preattentive perceptual organization: graphical enumeration and voting methods. *IEEE Trans. Syst. Man Cybern.*, 24:246–267, 1994.

[81] R. Schalkoff. *Pattern Recognition: Statistical, Structural and Neural Approaches.* John Wiley & Sons, Inc., New York, 1992.

[82] D. S. Seong, H. S. Kim, and K. H. Park. Incremental clustering of attributed graphs. *IEEE Trans. Syst. Man Cybern.*, 23:1399–1411, 1993.

[83] L. G. Shapiro, J. D. Moriarty, R. M. Haralick, and P. G. Mulgaonkar. Matching three-dimensional objects using a relational paradigm. *Pattern Recognition*, 17:385–405, 1984.

[84] P. N. Suganthan, E. K. Teoh, and D. P. Mital. Pattern recognition by graph matching using the Potts MFT neural networks. *Pattern Recognition*, 28:997–1009, 1995.

[85] P. N. Suganthan, E. K. Teoh, and D. P. Mital. Pattern recognition by homomorphic graph matching using hopfield neural networks. *Image and Vision Computing*, 13:45–60, 1995.

[86] J. W. Tai and Y. J. Liu. Chinese character recognition. In *Syntactic and Structural Pattern Recognition, Theory and Applications*, pages 415–452. World Scientific, 1989.

[87] C. C. Tappert. Speed, accuracy, flexibility trade-offs in on-line character recognition. Technical report, IBM Res. Rep. RC13228, 1987.

[88] C. C. Tappet, C. Y. Suen, and T. Wakahara. The state of the art in on-line handwriting recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 12:787–808, 1990.

[89] C. Thornton and B. D. Boulay. *Artificial Intelligence through Search.* Kluwer Academic Publishers, The Netherlands, 1992.

[90] W.-H. Tsai and K.-S. Fu. Error-correcting isomorphisms of attributed relational graphs for pattern analysis. *IEEE Trans. Syst. Man Cybern.*, 9:757–768, 1979.

[91] W.-H. Tsai and K.-S. Fu. Subgraph error-correcting isomorphisms for syntactic pattern recognition. *IEEE Trans. Syst. Man Cybern.*, 13:48–62, 1983.

[92] Y. T. Tsay and W. H. Tsai. Attributed string matching by split-and-merge for on-line Chinese character recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15:180–185, 1993.

[93] C.-C. Tseng, S.-L. Lay, B.-S. Jeng, and K.-S. Chou. Candidate selection in on-line Chinese character recognition system using voting scheme. In *Proc. Int. Symp. Multi-Technology Information Processing*, pages 355–360, Taiwan, 1996.

[94] J. Tsukumo. Handprinted Kanji character recognition based on flexible template matching. In *Proc. 11th Int. Conf. Pattern Recognition*, pages 483–486, 1992.

[95] J. Tsukumo and H. Tanaka. Classification of handprinted Chinese characters using non-linear normalization and correlation methods. In *Proc. 9th Int. Conf. Pattern Recognition*, pages 168–171, 1988.

[96] R. A. Wagner and M. J. Fischer. The string-to-string correction problem. *J. ACM*, 21:168–173, 1974.

[97] T. Wakahara, H. Murase, and K. Odaka. On-line handwritten recognition. *Proc. IEEE*, 80:1181–1194, 1992.

[98] J. T. L. Wang, K. Zhang, and G.-W. Chirn. Algorithms for approximate graph matching. *Information Sciences*, 82:45–74, 1995.

[99] R. C. Wilson and E. R. Hancock. A bayesian compatibility model for graph matching. *Pattern Recognition Letters*, 17:263–276, 1996.

[100] A. K. C. Wong, S. W. Lu, and M. Rioux. Recognition and shape synthesis of 3-d objects based on attributed hypergraphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11:279–290, 1989.

[101] A. K. C. Wong, M. You, and S. C. Chan. An algorithm for graph optimal monomorphism. *IEEE Trans. Syst. Man Cybern.*, 20:628–636, 1990.

[102] Y. Wu and X. Ding. *Chinese Character Recognition—Principles, Methods and Implementations.* The Higher Education Publishing House, China, 1992.

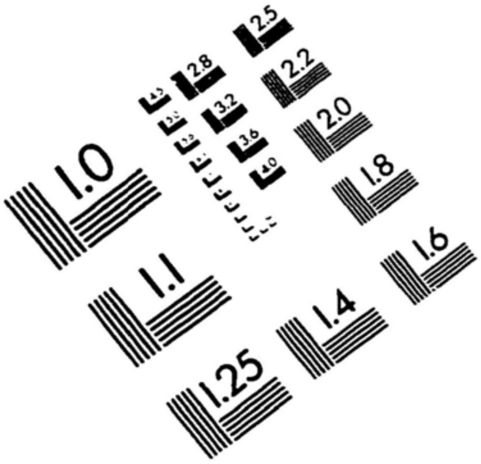
[103] Y. Wu, N. Xu, and X. Ding. A new clustering method for Chinese character recognition system using neural networks. *ACTA Electronica Sinica*, 22:1–8, 1994.

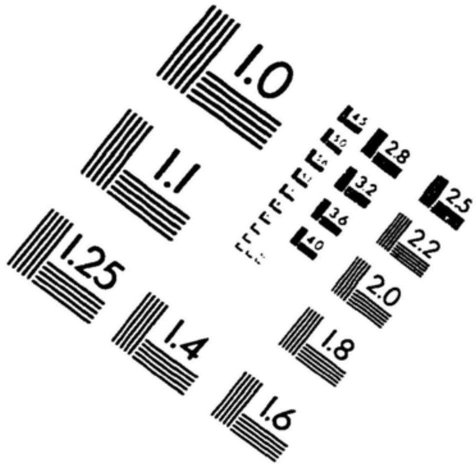
[104] B. Zhang and L. Zhang. A new heuristic search technique — algorithm SA. *IEEE Trans. Pattern Anal. Mach. Intell.*, 7:103–107, 1985.

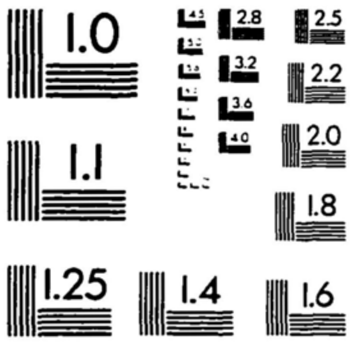
[105] B. Zhang and L. Zhang. Statistical heuristic search. *J. Computer Science Technology*, 2:1–11, 1987.

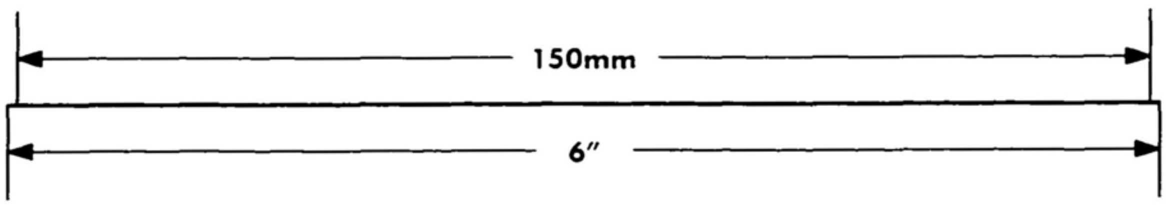
[106] S. W. Zucker, E. V. Krishnamurthy, and R. L. Haar. Relaxation processes for scene labeling: convergence, speed, and stability. *IEEE Trans. Syst. Man Cybern.*, 8:41–48, 1978.

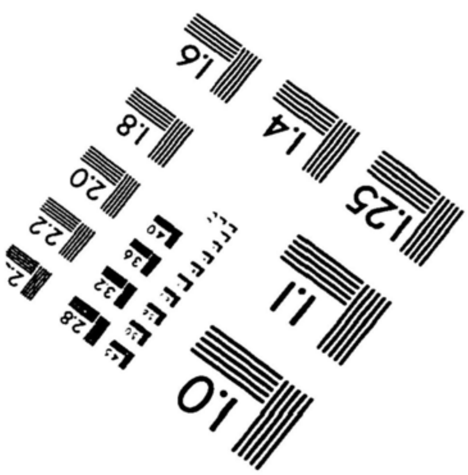
IMAGE EVALUATION
TEST TARGET (QA-3)











APPLIED IMAGE . Inc
1653 East Main Street
Rochester, NY 14609 USA
Phone: 716/482-0300
Fax: 716/288-5989