l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Jinpeng Li

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Université de Nantes
sous le label de l'Université de Nantes Angers Le Mans

Discipline : Informatique
Spécialité : Automatique et Informatique Appliquée
Laboratoire : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN)

Soutenue prévue le 23 Octobre 2012

École doctorale : 503 (STIM)
Thèse n° :

Extraction de connaissances symboliques et relationnelles appliquée aux tracés manuscrits structurés en-ligne

Symbol and Spatial Relation Knowledge Extraction Applied to On-Line Handwritten Scripts

JURY

Rapporteurs :	**M. Eric Anquetil**, Professeur des Universités, Institut National des Sciences Appliquées de Rennes **M. Salvatore-Antoine Tabbone**, Professeur des Universités, Université de Lorraine
Examinateur :	**M. Jean-Marc Ogier**, Professeur des Universités, Université de La Rochelle
Directeur de thèse :	**M. Christian Viard-Gaudin**, Professeur des Universités, Université de Nantes
Co-encadrant de thèse :	**M. Harold Mouchère**, Maître de conférences, Université de Nantes

Acknowledgements

The subject of this thesis was originally proposed by my supervisors, Prof. C. Viard-Gaudin and Doct. H. Mouchère. Therefore, specially thank you for my supervisors' thesis, suggestions, correcting of all my papers, funding of "Allocation de Recherche du Ministère" and "Projet DEPART", etc. It is a really new and challenging subject, which has been largely unexplored until now. Knowledge extraction is not only for prediction, but also for human understanding. In addition, thanks to many chances of international conference participations (CIFED2010, ICDAR2011, KDIR2011, DRR2012, CIFED2012, ICFHR2012), I earned experiences to talk to people from different countries and known what the other people are doing. Thanks to an opportunity to organize a small seminar "séminaire au vert" with Sofiane and help a seminar "franco-chinois" in Polytech'Nantes. Thanks also to academic suggestions from Guoxian, Montaser, Sofiane, and Zhaoxin. Furthermore, thank you for all the people from the team IVC of IRCCyN, e.g. for small delicious drinkings ("pot" in French) from different countries. I am grateful for juries and examinator, Prof. Anquetil, Prof. Tabbone and Prof. Ogier, to take the time to review my thesis.

Concerning living in Nantes, thank you for my friends who helped me: Jiefu, Hongkun, Junle, LiJing, Jiazi, Dingbo, Zhujie, Baobao, Chuanlin, Fengjie, Hugo, Baptiste, Zhangyu, Wenzi, He HongYang, Junbin, Zhangyang, Biyi, Shuangjie, Zhaoxin, Tan Jiajie, Cédric, Dan, Zeeshan, Pierre, Emilie, and so on.

At the end, really thank you for my wife Huanru coming at Nantes for the end of my thesis, and allowing me that always focus on my interesting works.

Contents

1

Introduction

Since paper and pen invention, we human begin to write traces on pieces of paper to save information using different graphical language forms, e.g. text lines including letters and characters, flowcharts, mathematical equations, ideograms, schema, etc. The graphical language forms are understandable for human being. With computer emergence, this information is saved in human-predefined “bit” format data (e.g. *Unicode* and *UTF* for letters and characters, LaTeX for mathematical equations, markup language standards), which are machine “understandable”. A machine can easily use bit-format data to display corresponding handwritten traces. The contrary process is a more-challenging handwriting recognition process, which automatically translates human handwriting into bit-format data via the machine.

A traditional handwriting recognition system (machine) [1, 2] usually takes advantage of a training dataset, referred as a ground-truth dataset, to perform some machine learning algorithms. These algorithms are in charge of two main tasks, one is to segment the ink traces in relevant segments (segmentation task), then the second task is to recognize the corresponding segments [2] by assigning them a label from a set of symbols defined by a given graphical language (classification task). The problem of symbol segmentation is by itself a very though job. Usually, to alleviate the difficulties, segmentation and classification tasks are tied so that the classifier can optimize the output of the segmentation stage. In our case, we do not want to rely on such schema since at that point we ignore the underlying graphical language, and thus no symbol classifier can be invoked. Our work concerns knowledge extraction from graphical languages whose symbols are a priori unknown. We are assuming that the observation of a large quantity of documents should allow to discover the symbols of the considered language. The difficulty of the problem is the two-dimensional and handwritten nature of the graphical languages that we are studying.

To deal with the segmentation problem, a naïve approach would rely on the connected strokes. However, a simple symbol equal "=" is composed of two non-connected strokes. More elaborated works, (symbol relation tree [3], recursive horizontal and vertical projection profile cutting [4], recursive X-Y cut [5], grouping strokes by a maximizing confidence level [6], etc.) are proposed to study the segmentation problem.

Concerning the recognition of isolated segmented symbols, many classifiers can be applied: KNN (K-Nearest Neighbor) [7], ANN (Artificial Neural Networks) [8], SVM (Support Vector Machine) [9], HMM (Hidden Markov Model) [10], etc.

With a traditional approach, and if we are considering the example of mathematical expressions as displayed in Fig. 1.1, the graphical symbols are defined beforehand in the ground-truth dataset. Classifiers are trained to recognize graphical symbols. After that, unlabeled graphical documents on the left side of Fig. 1.1 can be segmented and recognized as labeled symbol sets shown on the right side.

In other words, many existing recognition systems [1] require the definition of the character or symbol set, and rely on a training dataset which defines the ground-truth at the symbol level. Such datasets are essential for the training, evaluation, and testing stages of the recognition systems. However, collecting all the ink samples and labeling them at the symbol level is a very long and tedious task, especially for an unknown language. Hence, it would be very interesting to be able to assist this process, so that most of the tedious work can be done automatically, and that only a high-level supervision needs to be done to conclude the labeling process.

Without knowing any symbol of an unknown two-dimensional graphical language, creating the high-level supervision is an unsupervised symbol learning procedure. For instance, given the set of expressions as shown on the left side of

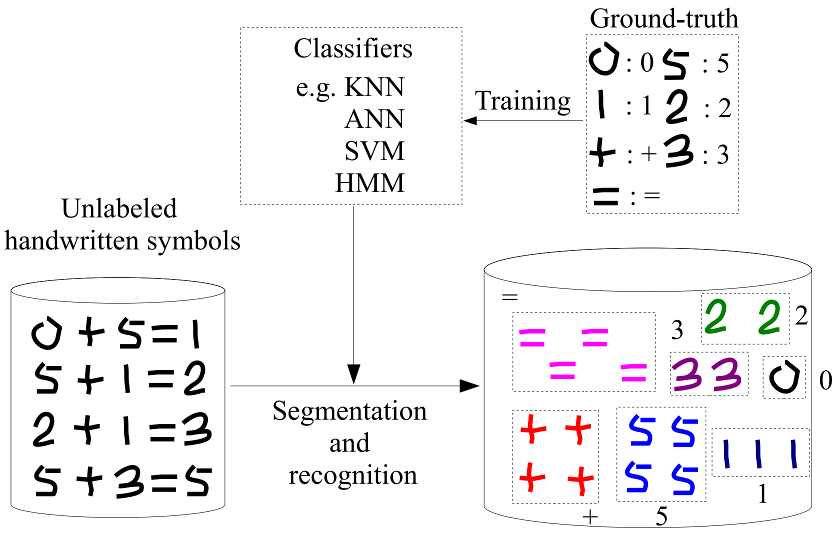


Figure 1.1: Traditional handwriting recognition

Fig. 1.2, we would like to extract the presence of 7 different symbols, represented by 20 different instances. Thus, only 7 symbol sets instead of 20 symbols have to be labeled so that symbol labeling workload can be reduced in this example.

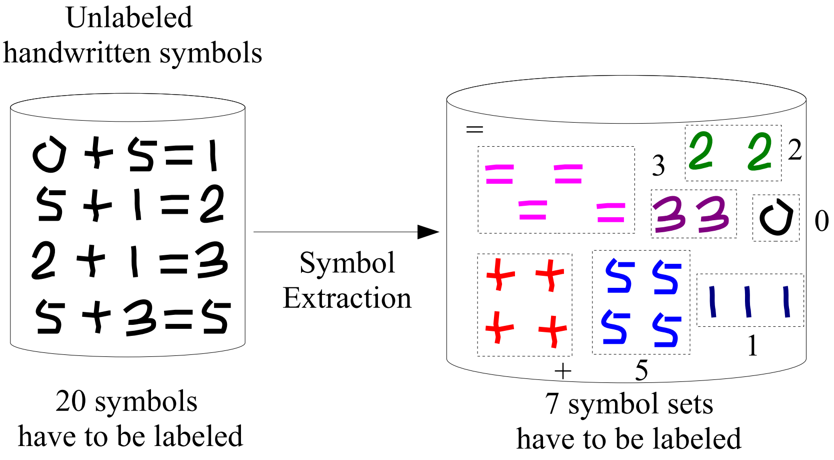


Figure 1.2: Extracting the symbol set from a graphical language

But the unsupervised symbol extraction procedure is quite difficult. First of all, no symbol segmentation is defined in a dataset. We consider that a stroke, a sequence of points between a pen-down and a pen-up, is the basic unit. Should this assumption not be verified, then an additional segmentation process will have to be undergone, so that every basic graphical unit, termed as a grapheme, belongs to a unique symbol. Conversely, a symbol can be made of one or several strokes, which are not necessarily drawn consecutively, i.e. we do not exclude interspersed symbols.

Fig. 1.3 displays a horizontal stroke which may belong to a single symbol (minus, "$-$"), or belong to a part of symbol with the same horizontal stroke (equal, "$=$") or another stroke (plus, "$+$"). The difficulty is to find out which combination of strokes is a symbol. In other words, we first require an unsupervised symbol segmentation method. It is obvious to observe that symbols are somehow "frequent" spatial compositions of strokes in handwritten equations. For instance, "$=$", "$+$", "5" are repeated four times, and "1" is present three times in Fig. 1.2, while there is a total of 8 vertical strokes including those belonging to the five "+". Comparing a repetitive pattern "$+1$" (three strokes repeated twice) and its sub-part "$+$" (two strokes repeated four times), both of them can be a symbol. But which of them is more likely to be a symbol? In this thesis, we will introduce a criterion, the Minimum Description Length (MDL) principle [11], to determine which of them is more likely to be a symbol.

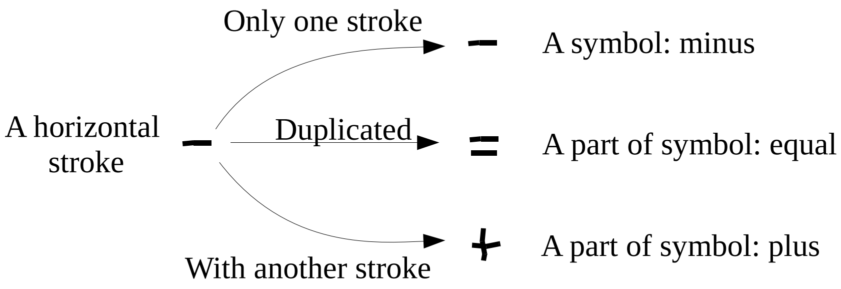


Figure 1.3: A stroke may be a symbol or a part of symbol

As the MDL criterion depends on symbol frequencies, it is necessary to count the number of symbol occurrences. In order to be able to count (or search) efficiently how many instances of single-stroke symbol or multi-stroke symbol (e.g. a combination of two strokes "$+$" has four instances in Fig. 1.2), we propose to organize the two-dimensional graphical language as relational graphs between strokes. In a graphical language, the symbol counting (searching) problem therefore becomes a sub-graph searching problem.

For example, the first two mathematical expressions of Fig. 1.2 could be represented by the two graphs in Fig. 1.4, and the symbols are sub-graphs in the graphs. To avoid an ambiguity with the strokes coded by the same representative grapheme, all the strokes are indexed by a different number $(.)$. A relative problem is how to learn relationships (e.g., *Right*, *Intersection*, *Below*, etc.) between strokes, called *spatial relation*, in the relational graphs. If such relations are exhibited in a graph, we can see that multi-stroke symbols “+”, “5”, and “=” will be present. The multi-stroke symbols could be sought in graphs. Hence, with the help of a discovery criterion based on a graph description, we should be able to produce a symbol segmentation.

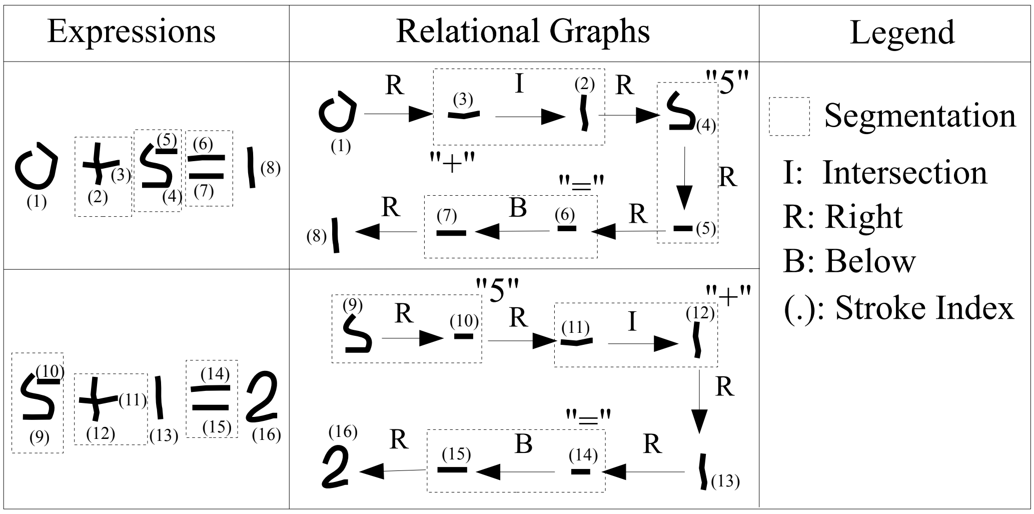


Figure 1.4: Expressions and corresponding relational graphs

Once the segmentation is available, as shown in Fig. 1.5, a clustering technique is needed for sorting symbols according to different shapes (right side of Fig. 1.5). In order to implement the clustering technique, a distance should be developed between two graphical symbols. According to the number of symbol strokes, we can divide the distance computation approaches into two kinds of problems, two single-stroke symbol comparison (a simple case) and two multi-stroke symbol comparison (a more complicated case). Two single-stroke symbols comparison can be well solved by Dynamic Time Warping (DTW) [12].

Nevertheless computing the distance between two multi-stroke symbols is more difficult since two writers may write a visually same symbol with different stroke orders, different stroke directions, and even different stroke numbers. It seems appropriate that the distance should be independent of these variations. To reach this goal, we will introduce a stroke-order-free, stroke-direction-free, and stroke-number-free distance.

Taking a two-stroke symbol “+” as an example, it could be written with four

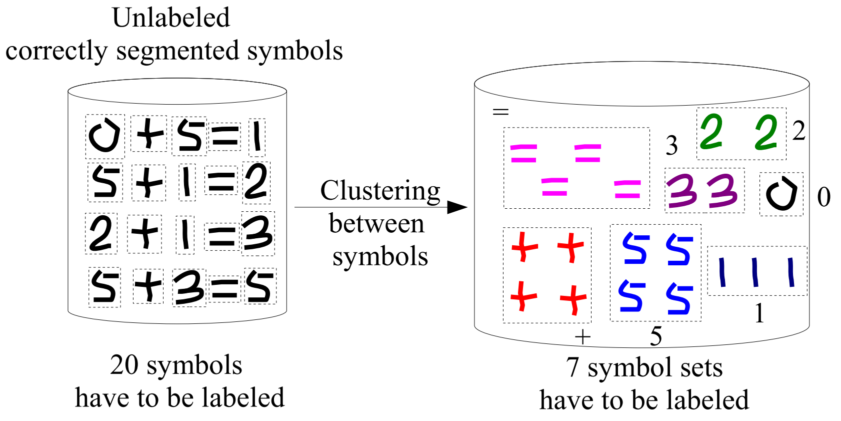


Figure 1.5: Correctly segmented symbols are grouped into clusters

different ways (four instances of a symbol "+") as shown in Fig. 1.6. The number of handwritten ways increase fast in terms of stroke number. Traditionally in online handwriting, we will concatenate strokes in a symbol instance by a natural written order, and then a distance between two symbol instances can be obtained by Dynamic Time Warping (DTW) algorithm [12]. However, with different written orders and written directions, the distance between same symbols will become large. We will discuss the complexity of this problem later in this thesis.

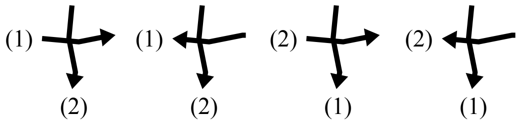


Figure 1.6: Four different handwriting trajectories for a two-stroke symbol "+"

Once we have selected a distance between symbols, we use a clustering technique for grouping symbols into several sets. From each set, we choose a representative sample. The representative samples are stored in a visual codebook as a visual interface for human being, in which we can manually annotate symbols. Nevertheless unsupervised symbol segmentation is non-trivial. It is difficult to generate perfect symbol segments; each of them containing exactly one symbol instance. A segment may contain several symbol instances, or a symbol instance and half in case of under segmentation problem. In the visual codebook, a user (human being) can easily separate symbol instances. Nevertheless the system has to find a correct stroke mapping from labeled samples to raw samples.

Fig. 1.7 shows the case where a frequent pattern "+1" is considered as a symbol instance. In fact, this segment contains two symbol instances. A visual codebook produced from this segmentation is illustrated in Fig. 1.8. A user can easily separate the segment "+1" into two isolated symbol instances, "+" and "1" in the visual codebook. This will require a multi-symbol mapping from labeled samples to raw samples.

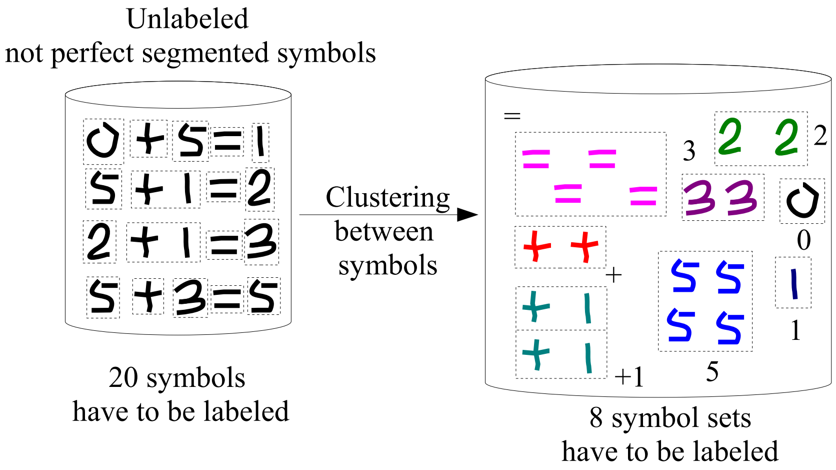


Figure 1.7: A not perfect symbol segmentation ("+1" is defined as a symbol)

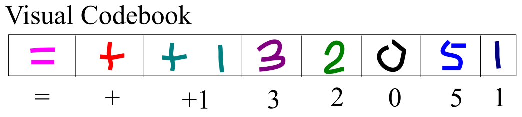


Figure 1.8: A visual codebook for user labeling

After the presentation of the problems that we want to address with the help of this example of mathematical expressions, we can introduce the general scheme that will be developped all along this document. It is presented in Fig. 1.9, and we will refer to this figure to position the different contributions that are described in the following chapters of this document.

In this thesis, Chapter 2 discusses works relevant to our topic. It contains several parts: symbol segmentation using the MDL principle, spatial relation modeling, clustering techniques and its evaluation, codebook extraction. In a graphical language, we need to first segment symbols, and then give labels to them. The MDL principle will be introduced to extract symbols. We propose to model the graphical language as relational graphs by defining nodes as strokes and edges as spatial relations. After that, we group symbols into sets (a codebook) with similar shapes using clustering techniques.

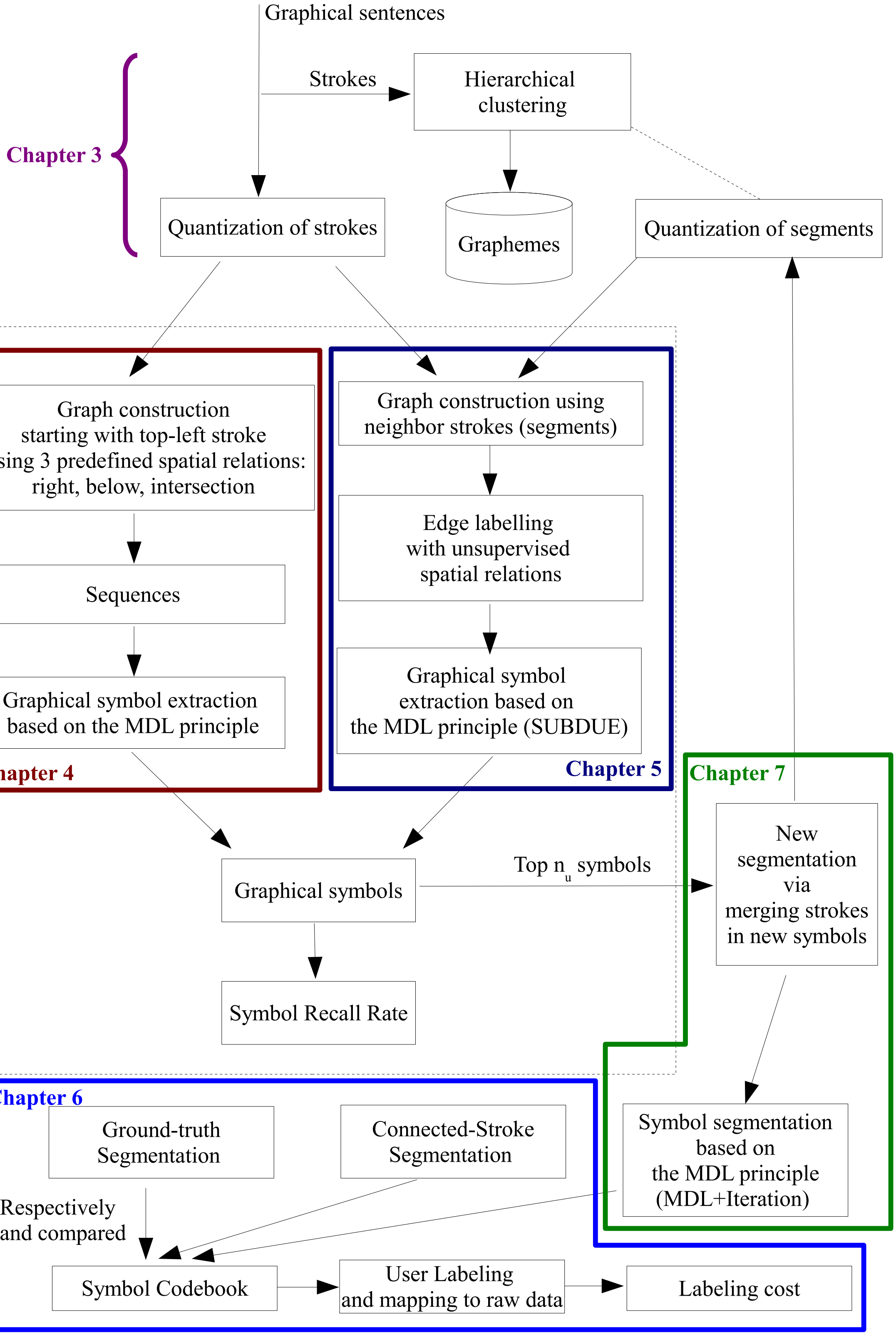

Graphical sentences
Strokes
Hierarchical clustering
Chapter 3
Quantization of strokes
Graphemes
Quantization of segments
Graph construction starting with top-left stroke Using 3 predefined spatial relations: right, below, intersection
Sequences
Graphical symbol extraction based on the MDL principle
Chapter 4
Graph construction using neighbor strokes (segments)
Edge labelling with unsupervised spatial relations
Graphical symbol extraction based on the MDL principle (SUBDUE)
Chapter 5
Chapter 7
New segmentation via merging strokes in new symbols
Graphical symbols
Top n_u symbols
Symbol Recall Rate
Symbol segmentation based on the MDL principle (MDL+Iteration)
Chapter 6
Ground-truth Segmentation
Connected-Stroke Segmentation
Respectively and compared
Symbol Codebook
User Labeling and mapping to raw data
Labeling cost

Figure 1.9: Thesis global view

Chapter 3 introduces the problem of quantifying isolated graphical symbols. We first choose a clustering technique for quantifying graphical symbols. The clustering technique requires a similarity between two symbols. Feature extraction in our system will be discussed for the similarity between two single-stroke symbols (a simple case) and two multi-stroke symbols (a complex case).

We try to set up a graphical language as relational graphs using predefined spatial relations, which are limited as Directed Acyclic Graphs (DAG). DAG are then transformed into sequences in Chapter 4 so that text mining technique can be applied. Chapter 4 shows some encouraging results where some lexical units are successfully extracted.

In order to be capable to process a more general two-dimension graphical language, we extract spatial relation features within three levels in Chapter 5: distance relation, orientation relation, and topological relation. The spatial relations can be embedded into a fix-length feature space. In feature space, we can cluster spatial relations into several prototypes. Furthermore, a more general relational graph (not limited to a DAG) can be produced. Chapter 5 shows how to extract sub-graphs in these produced relation graphs using the Minimum Description Length (MDL) principle, which is an algorithm that minimizes the description length of an extracted lexicon and relational graphs using the extracted lexicon. The lexical units could have a hierarchical structure.

We can use this unsupervised symbol learning method for an application that reduces symbol labeling workload. During symbol extraction, a symbol segmentation will be generated. We can use the symbol segmentation to generate a codebook. A tentative test with ready-made segmentations is shown in Chapter 6. We also propose a multi-symbol mapping method to solve the situation where several symbols are mixed in a cluster, and propose a labeling cost to evaluate how much work has been reduced. Chapter 7 finally presents an experiment very closed to a real context: it uses the unsupervised symbol segmentation to reduce symbol labeling workload. In this chapter we show that the spatial relations and symbol definitions are linked and we propose an iterative extraction of them.

In the next chapter, we will present the state of the art about relative works.

2

State of the Art

In this chapter, we will present relevant works linked to the background of this thesis. As mentioned in the introduction, traditional handwriting recognition systems rely on ground-truth datasets, which contain correctly segmented and correctly labeled symbols. However, the tasks consisting in the segmentation and annotation of a document at the stroke level is non trivial. The Minimum Description Length (MDL) principle is a possible solution to produce an unsupervised segmentation with a labeling at the symbol level. Thus, we will first present in Section 2.1 the MDL principle. Since as mentioned in the introduction, a graphical language will be modeled as relational sequences and relational graphs, the MDL principle will be explained with two simple examples on relational sequences and relational graphs respectively. These two examples are inspired by [11] and [13]. Secondly, the modeling of sequences and of graphs need spatial relations. Section 2.2 presents how to model spatial relations between objects used in current recognition systems. After the unsupervised symbol segmentation using the MDL principle, we propose to group segmented symbols into a codebook using a clustering technique. Several clustering techniques and their evaluation will be discussed in Section 2.3. We can therefore label symbols from the codebook rather than each symbol in the dataset. The codebook generation will be presented in Section 2.4. Consequently, more symbol labeling workload could be saved, and the ground-truth dataset could be built more easily.

2.1 Symbol Segmentation Using the MDL principle

In offline handwritten annotation, [14] proposes a similar concept that helps to give labels to Lampung characters from an Indian language. Few people know this language. During the creation of training datasets which contains labeled characters, it is time-consuming to assign large-scale characters with corresponding correct labels by limited number of people who understand Lampung characters. The proposed system in [14] groups Lampung characters into several clusters according to different shapes. We can therefore give labels to clusters rather than to each character. The experiment results show that this procedure can save most of human work. However, the critical problem of symbol segmentation has not been discussed[15]; all the isolated characters were correctly segmented in advance. In this thesis, an important contribution is to automatically generate a segmentation at the symbol level so that we can give labels on each cluster to reduce human symbol labeling cost.

To tackle the symbol segmentation problem, [16] uses convolutional deep belief networks for a hierarchical representation (segmentation) on two-dimensional images. Some meaningful frequent patterns (faces, cars, airplanes, etc.) are extracted at different levels. Moreover, several other works are using heuristic approaches [17] for text segmentation. One famous approach is using the Minimum Description Length (MDL) principle [18]. The MDL principle's fundamental idea is that any regularity can be used for compressing a given data [19]. In our case, we would like to extract lexical units that compress a graphical language. An iterative algorithm is proposed in Marcken's thesis [11, 20] to build the lexicon from texts, which are character sequences. The principal idea of this algorithm is to minimize the description length of sequences by iteratively trying to add and delete a word, in terms of the MDL principle [18]. Ref. [11] reports a recall rate of 90.5% for text words [20] on the Brown English corpus [21], which is a text dataset. **We propose to extend this kind of approach on real graphical languages where not only left to right layouts have to be considered.**

Formally, given an observation U, we try to choose the lexical unit u which minimizes the description length:

$$DL(U,u) = I(u) + I(U|u) \tag{2.1}$$

where $I(u)$ is the number of bits to encode the lexical unit u and $I(U|u)$ is the number of bits to encode the observation U using the lexical unit u. To understand the MDL principle on texts, in the next section, we will give an example showing the general idea.

2.1.1 A Sequence Case

We describe a simple example inspired by [11] to give the general idea of the MDL principle. The aim is to find a lexicon [20] using the MDL principle. We analyze the expression "$1234 - 2/1234$" as a sequence of graphemes:

$$U = (1, 2, 3, 4, -, 2, /, 1, 2, 3, 4). \tag{2.2}$$

For simplicity, spatial relations are omitted, but they have to be taken into account in a real algorithm. The description length of U in MDL can be represented by $DL(U) = I(U)$, where $I(U)$ is a code length function that is equal to the number of characters, e.g. $I(U) = 11$. We assume that u is a lexical unit, obtained by a simple concatenation of elementary symbols of the language alphabet. $DL(U|u) = I(U|u) + I(u)$ represents the sum of the code length after U is compressed by replacing instances of u (Viterbi representation in Fig. 2.1 [11]), and the code length of u.

To have a better understanding, we try to analyze the description length with three different lexicons, (1) a lexicon without any lexical unit, (2) a lexicon including the discovered lexical unit

$$LU\_2 = (1, 2, 3, 4),$$

and (3) the lexicon including the discovered lexical unit (the whole expression):

$$LU\_3 = (1, 2, 3, 4, -, 2, /, 1, 2, 3, 4).$$

In Tab. 2.1, we have three lexicons, L_1, L_2 and L_3 to interpret U by Viterbi representation [11]. Intuitively L_2 is the best lexicon since L_2 contains a word "1234".

Table 2.1: Three lexicons for the sequence of graphemes $U = (1, 2, 3, 4, -, 2, /, 1, 2, 3, 4)$

L_1	$\{\}$
Description Length of U:	$DL(U) = 11$
L_2	$\{LU\_2 = (1, 2, 3, 4)\}$
Viterbi representation $(U \mid LU\_2)$:	$LU\_2 \circ (-) \circ (2) \circ (/) \circ LU\_2$
Code length of $(U \mid LU\_2)$	$I(U \mid LU\_2) = 5$
Code length of (LU_2):	$I(LU\_2) = 4$
Description length:	$DL(U \mid LU\_2) = I(U \mid LU\_2) + I(LU\_2) = 9$
L_3	$\{LU\_3 = (1, 2, 3, 4, -, 2, /, 1, 2, 3, 4)\}$
Viterbi representation $(U \mid LU\_3)$:	LU_3
Code length of $(U \mid LU\_3)$:	$I(U \mid LU\_3) = 1$
Code length of (LU_3):	$I(LU\_3) = 11$
Description length:	$DL(U \mid LU\_3) = I(U \mid LU\_3) + I(LU\_3) = 12$

The Viterbi representation is used to interpret U by matching the longest sequence in L_2 shown in Fig. 2.1. For example, U is interpreted by L_2 as $(1, 2, 3, 4) \circ (-) \circ (2) \circ (/) \circ (1, 2, 3, 4)$ where $\circ$ is a concatenation. Comparing the three lexicons in Tab. 2.1, we found that L_2 reports the minimum description length, which means $(1, 2, 3, 4)$ is the best lexical unit.

An algorithm to build the optimal lexicon is presented in [20] using the MDL principle. In this algorithm, a word is iteratively added or removed in order to minimize the description length until the lexicon cannot be changed. Thus, we get an optimal lexicon L on the training handwriting database containing the sequences of graphemes/relations. In the next section, we will introduce an example using the MDL principle on graphs.

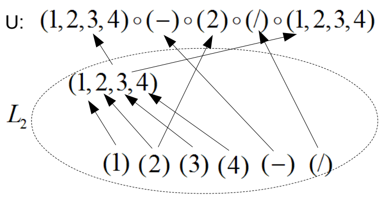


Figure 2.1: Viterbi representation

2.1.2 A Graph Case

A graph is an interesting data structure to describe documents at different levels. For instance, in off-line data (images), [22] first groups basic units (pixels) into regions, and then defines graphs between regions. A colour segmentation process is therefore proposed based on the graphs. Similarly, we can describe an on-line graphical language with a graph approach; strokes, which are the basic graphical units, define the nodes and they are connected by edges according to some spatial relations. In this situation, unlike the sequence case, the search space for the combination of units which makes up possible lexical units is much more complex since it is no longer a linear one. Thus, a graph mining technique is required to extract repetitive patterns in the graphs. To perform such as task, SUBDUE (SUBstructure Discovery Using Examples) system [23] will be introduced. It is a graph based knowledge discovery method which extracts substructures (sub-graphs) in graphs using the MDL principle. Ref. [13] gives the precise definition of $DL(G, u)$ on graphs. The system SUBDUE iteratively extracts the best lexical unit (substructure) using the MDL principle. A unit could be a hierarchical structure [24] built with a recursive approach.

Formally, we assume that $I(u)$ denotes the sum of the number of nodes and edges for encoding (description length) a graph u. $(G|u)$ represents a graph whose instances of sub-graph u are replaced by a new node. $I(G|u)$ means the sum of the number of nodes and edges from a graph $(G|u)$. The strokes in the expressions as shown in Fig. 1.4 are labeled with a codebook defined in Fig. 2.2. Note that a label (grapheme) from the codebook may be used for several strokes. For instance, the label "b" will be used for several different strokes of the expressions, a piece of the '$+$' sign, a bar of the '5', the '=' symbol. The corresponding labeled graphs G are displayed in Fig. 2.3. We can find that $DL(G) = I(G) = 30$. Now we try to compress G by replacing instances of a symbol u"=" as shown in Fig. 2.4 ($I(u)$=3), and then a compressed graph $(G|u)$ is obtained in Fig. 2.5 where $I(G|u) = 26$. Thus $DL(G, u) = I(u)+I(G|u) = 29$. We have reduced by 1. Hence, the token $u =$"=" could be taken into account as a lexical unit.

In order to assess two extraction methods (in the sequence case and in the graph case), two graphical languages will be presented in Section 3.6: a single-line mathematical expression corpus and a more general two-dimension flowchart corpus.

Codebook

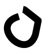
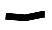
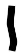
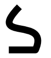
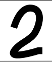
Shape					
Label	a	b	c	d	e

Figure 2.2: Example of codebook used for coding expressions of Fig. 1.4

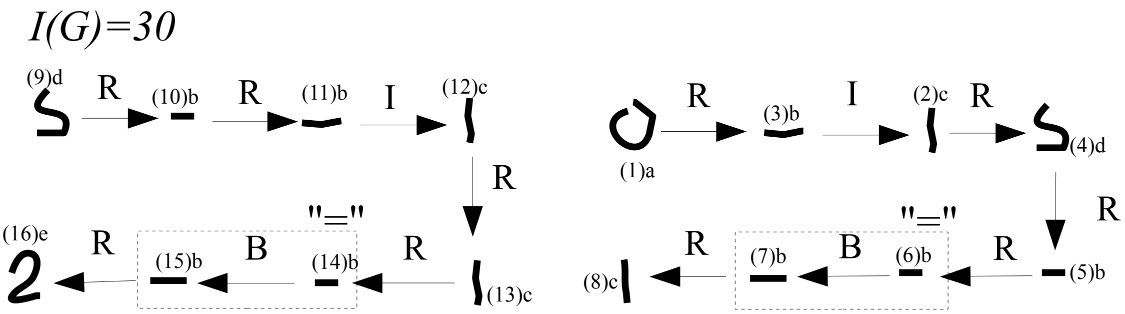


Figure 2.3: Original Graphs

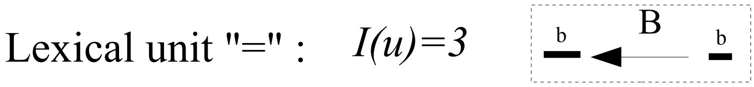


Figure 2.4: An extracted lexical unit

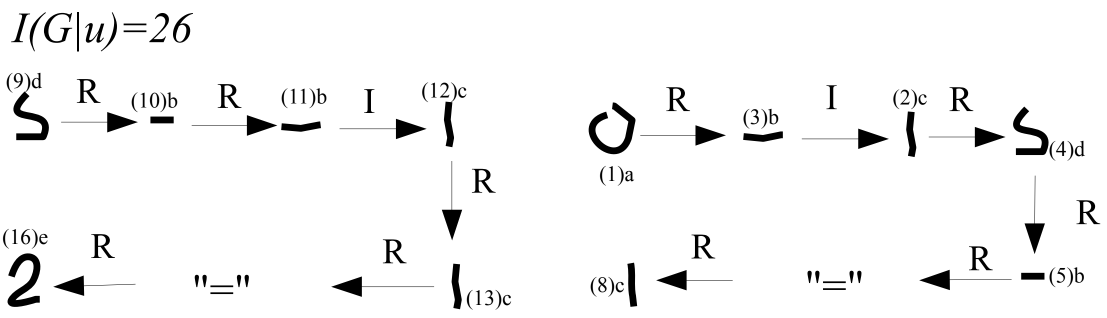


Figure 2.5: Compressed Original Graphs

Single-line mathematical expressions are suitable for sequence mining. Flowcharts are more challenging, and they will require to develop graph mining approaches. As a preliminary step, it is necessary to transform the set of strokes into a relational graph. To obtain such a representation, we need to define and model the spatial relations that link strokes together. This points will be presented in the next section.

2.2 Spatial Relations

All communication is based on a fact that participants share conventions that determine how messages are constructed and interpreted. For graphical communication these conventions indicates how arrangements, or layouts, of graphical objects encode information. For instance, graphical languages (sketch, mathematical or chemical expressions, etc.) are composed of a set of symbols within some constraints. These constraints could be the grammar of this language, the layout of symbols, and so on. Furthermore, the symbols are also composed of a layout of strokes. The layout means that elements (symbols, strokes) are arranged in the two dimensional space, so that we can build a coherent document. Fig. 2.6 illustrates two handwritten documents, a handwritten mathematical expression and a handwritten flowchart. Spatial relations specify how these elements are located in the layout.

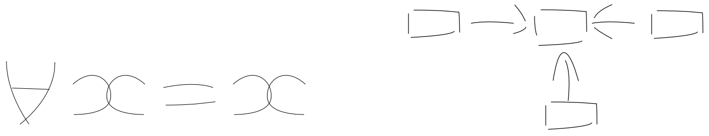

(a) (b)

Figure 2.6: Two different handwritten graphical documents: (a) a handwritten mathematical expression, (b) a handwritten flowchart.

As an example, suppose that we have a set of two different shapes of strokes called *graphemes* $\{\backslash, /\}$. We assume that these graphemes are well detected by a clustering algorithm as discussed in Chapter 3. Using these two graphemes, we can compose two different symbols:“$\wedge$” and “$\vee$”. A difference between “$\wedge$” and “$\vee$” is the spatial relation; “$\backslash$” is put on the right side in “$\wedge$” and on the left side in “$\vee$”. These spatial relations, left and right, are easily defined manually.

With more graphemes and more spatial relations, it is possible to design new symbols. For instance, using a set of graphemes $\{\backslash, -, /\}$, we can compose a symbol "$\forall$" with three strokes "$\backslash_{(1)}$", "$-_{(2)}$", and "$/_{(3)}$". We can say "$-_{(2)}$" is *between* "$\backslash_{(1)}$" and "$/_{(3)}$". In this case, *between* implies a relationship among three strokes, which is a cardinality of this spatial relation [25]. In this work, we limit the cardinality of spatial relation to two strokes, from a reference stroke to an argument stroke. However, with only three strokes, we have to consider six different stroke pairs to envisage all appropriate alternatives, for example "$\backslash_{(1)} \rightarrow -_{(2)}$", "$-_{(2)} \rightarrow \backslash_{(1)}$", "$\backslash_{(1)} \rightarrow /_{(3)}$", etc. The number of spatial relation couples will grow rapidly with an increasing number of strokes in the layout [26].

A traditional modeling of spatial relation is represented at three levels [25, 27]: topological relations, orientation relations, and distance relations. The topological characteristics are preserved under topological transformations, for example translation, rotation, and scaling [28]. The orientation relations calculate directional information between two strokes [29]. For instance a stroke A is on the right of another stroke B. The distance relations describe how far two strokes are.

Most of existing systems dealing with handwriting need some spatial relations between strokes. For instance, [29] uses a fuzzy relation position (orientation relations) for an analysis of diacritics on on-line handwritten text. In [30], authors add a distance information to design a structural recognition system for Chinese characters. In the context of handwritten mathematical expression recognition in [31, 32], authors use the three levels of spatial relations to create a Symbol Relation Tree (SRT) using six predefined spatial relations: inside, over, under, superscript, subscript and right. Spatial relations are also useful in automatic symbol extraction as in our work [26, 33]. In short, we automatically extract graphical symbols from a graphical language with a simple set of predefined spatial relations. Our approach was successfully tested on a simple mathematical expression database. We predefined three domain specific relations (right, below, and intersection) to create a relational graph between strokes. The creation of this relational graph starts with the top-left stroke because of the left to right handwriting orientation. In the relational graph, repetitive sub-graphs composed of graphemes and predefined spatial relations are considered graphical symbols.

Using a simple set of predefined spatial relations obviously is not enough for describing “a new” (or “an unknown”) complex graphical language. We may lose some unknown spatial relations which are important for a specified graphical language. Let us consider differences between 9 different layouts of the 2 previous strokes $\{\backslash, /\}$: “\/” “∨”, “/\”, “∧”, “<”, “<”, “>”, “>” and “×”. We want to distinguish these 9 layouts. We assume “\” as the reference stroke and “/” as the argument stroke. If we categorize these layouts by intersection, two groups will be obtained: {“\/”,“/\”, “<”, “>”} and {“∨”, “∧”, “<”, “>”, “×”}. If we categorize these layouts by four predefined directions (right, left, above, and below) of “\”, four groups will be obtained: {“\/”,“∨”}, {“∧”, “/\”}, {“<”, “<”}, and {“>”, “>”} with the confusing layout “×”. The combination of left (directional relations) and intersection (topological relations) allows the distinction of these 9 layouts. However, there are many combinations of spatial relations in a complex graphical language. It is hard to manually predefine all the useful combinations of spatial relations.

As mentioned in this section, topological relations, orientation relations, and distance relations are the three levels of traditional modeling of spatial relation. In order to understand this traditional modeling, we will introduce the three levels of spatial relations in the next three sub-sections.

2.2.1 Distance Relations

A distance relation denotes how far apart two objects are. In a simple case, we can assume that two objects are considered as two points $pt_1 = (x_1, y_1)$ and $pt_2 = (x_2, y_2)$. In analytic geometry [34], the distance between two points is given by the Euclidean distance:

$$dist(pt_1, pt_2) = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2}. \tag{2.3}$$

However, when two objects are very near, their shapes cannot be ignored. Each object in on-line handwriting is a set of points. To describe how far apart two objects are, we need a distance between two point sets. The Hausdorff distance $HD(.,.)$ is a metric between two point sets $obj_i = \{pt_i\}$ and $obj_j = \{pt_j\}$ [35]:

$$HD(obj_i, obj_j) = \max(hd(obj_i, obj_j), hd(obj_j, obj_i)) \tag{2.4}$$

where $hd(obj_x, obj_y) = \max_{pt_x \in obj_x} \min_{pt_y \in obj_y} (dist(pt_x, pt_y))$. Nevertheless in general, Hausdorff distance is used for matching two pattern shapes instead of measuring how far apart two objects are. In this thesis, we will meet many graphical symbols arrows which connect symbols. If the Hausdorff distance is used, it will generate a large distance. We prefer a distance by choosing the closest point pair $CPP(.,.)$ between two point sets:

$$CPP(obj_i, obj_j) = \min_{pt_i \in obj_i} \min_{pt_j \in obj_j} (dist(pt_i, pt_j)). \tag{2.5}$$

Fig. 2.7 shows an example for explaining why we choose the closest point pair. Three graphical symbols in a flowchart are illustrated in Fig. 2.7. For a logical, or functional interpretation of this flowchart, from the circle symbol we have to move to the arrow and then to the rectangle. To obtain this sequence, it will be necessary to consider that the arrow is closer to the circle than the rectangle. It will be the case if CPP(,) is used as the distance instead of HD(,) since

$$CPP(\text{“}Circle\text{”}, \text{“}Arrow\text{”}) < CPP(\text{“}Circle\text{”}, \text{“}Rectangle\text{”}) \tag{2.6}$$

while

$$HD(\text{“}Circle\text{”}, \text{“}Arrow\text{”}) > HD(\text{“}Circle\text{”}, \text{“}Rectangle\text{”}), \tag{2.7}$$

we can choose “Arrow” as the next symbol.

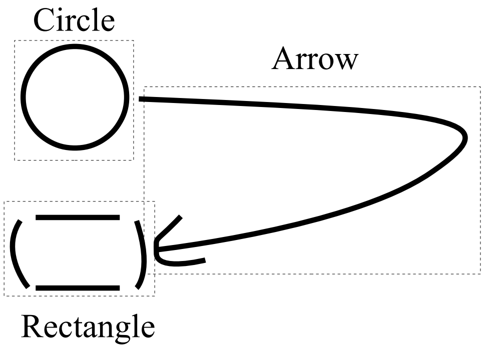


Figure 2.7: Which is the closest symbol from the symbol “Circle”?

In the next section, we will introduce orientation relation describing directional information between objects.

2.2.2 Orientation Relations

Orientation represents some directional information, e.g. *east*, *west*, *south*, *north*, etc. We can say a symbol is located at the *south* side of another symbol [25]. In this section, we will introduce a fuzzy directional relation between two graphical symbols [29, 36]. Ref. [29] shows that the directional relation not only depends on the positions of two symbols, but also on the shapes of two symbols.

In a fuzzy directional relation, we have to define a reference symbol R and an argument symbol A with respect to a reference direction $\overrightarrow{u}_\alpha$. The angle function varying in the boundary $[0, \pi]$ is defined by:

$$\beta(P, Q) = \arccos(\frac{\overrightarrow{QP} \cdot \overrightarrow{u}_\alpha}{||\overrightarrow{QP}||}) \tag{2.8}$$

where $\beta(P, P) = 0$. Thus taking an argument point P into account, we use the minimum value $\beta_{\min}$ among all the reference points $Q \in R$ for defining a directional angle:

$$\beta_{\min}(P) = \arg\min_{Q \in R} ||\beta(P, Q)||. \tag{2.9}$$

In order to normalize it from $[0, \pi]$ to $[0, 1]$, a simple linear function is applied:

$$\mu_\alpha(R)(P) = \max(0, 1 - \frac{2\beta_{\min}(P)}{\pi}). \tag{2.10}$$

Considering the whole argument point set A, we can accumulate $\mu_\alpha(R)(P)$ for all the points, and then normalize it. Thus, we can compute a relative direction value $M_\alpha^R(A)$ between a reference symbol R and an argument symbol A, with respect to a reference direction $\overrightarrow{u_\alpha}$:

$$M_\alpha^R(A) = \frac{1}{|A|} \sum_{x \in A} \mu_\alpha(R)(x). \tag{2.11}$$

Taking a reference symbol “2” comparing an argument symbol “5” as an example in Fig. 2.8 and a reference point P, we search for the minimum β angle. After that, we accumulate $\mu_\alpha(R)(P)$ for all the points in an argument symbol A using Eq. (2.11).

Eq. (2.11) works well for comparing two objects in raster graphics (image) [36].

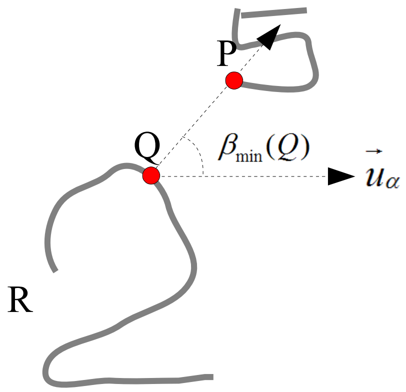


Figure 2.8: Fuzzy relative directional relationship from a reference symbol to an argument symbol with respect to a reference direction in Ref. [29].

In images, each object is composed of pixels. Two consecutive points in each stroke is connected. Nevertheless in on-line handwriting, in each stroke, two consecutive discrete points have some space between them according to resampling frequency. Ref. [29] points out that using Eq. (2.9) between two consecutive points will generate a “comb effect”.

In order to avoid this effect, Ref. [29] redefines β (see Eq. (2.9)) from $\vec{u}_\alpha$ to $\overrightarrow{QP}$ by the counter-clockwise direction. Hence, β is located in the range $[0, 2\pi]$. Considering a stroke composed of only one point, we just use the original definition β shown in Eq. (2.9). Usually a stroke are composed of several points. Each time, we consider a pair of consecutive points. We go through all the pairs of consecutive points Q_1 and Q_2 to compute two angles β_1 and β_2 respectively. If one β is in $[0, \pi/2]$ while the other is in $[\pi/2, \frac{3\pi}{2}]$, $\beta_{min}(P)$ will be zero. Otherwise, $\beta_{min}(P)$ will be computed as usual using Eq. (2.9).

Fig. 2.9 shows two general cases for β computation. The first case is when β_2 is in the range $[0, \pi/2]$ while β_1 is in the range $[\pi/2, \frac{3\pi}{2}]$ on left side of Fig. 2.9. It means P is located between two consecutive points at the reference direction $\vec{u}_\alpha$ side. It implies that there is a middle point between Q_1 and Q_2. The middle point makes that $\beta_{min}(P) = 0$. The second case is that both β are in the range $[\pi/2, \frac{3\pi}{2}]$ and then β_{min} is computed as usual using Eq. (2.9).

In this subsection, the orientation relation have been introduced. In the next section, we will study the last topological relation.

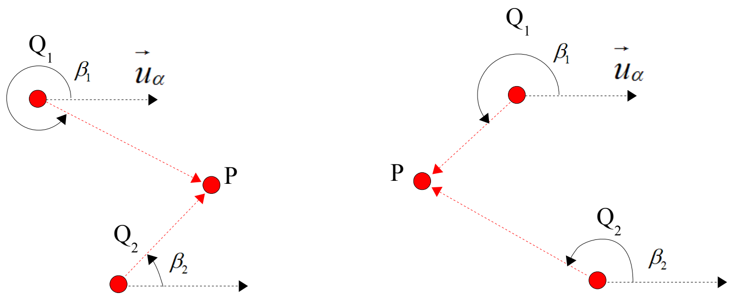


Figure 2.9: New β function to avoid a comb effect Ref. [29]

2.2.3 Topological Relations

The topological characteristics are preserved under topological transformations for example translation, rotation, and scaling [28]. A simple example of topological relation in Fig. 2.10 is the intersection of two strokes.

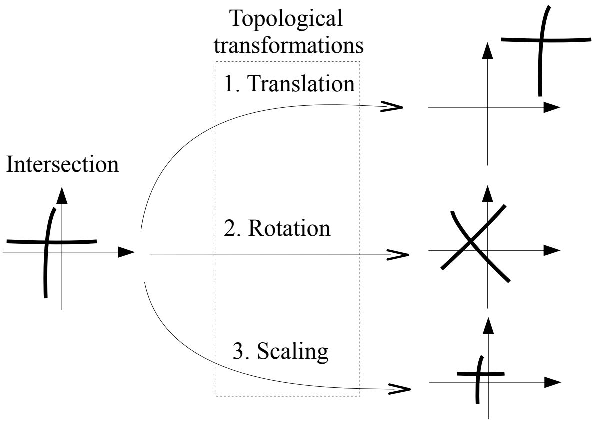


Figure 2.10: Topological transformations

To automatically generate topological relations, [37] develops formal categorization of binary topological relations between regions, lines, and points. Given a geometric object A, we can define the set-theoretic closure as $\overline{A}$, the boundary as ∂A, the exterior $A^{-} = U - \overline{A}$ (where U is the universe), and the interior $A^{\circ} = \overline{A} - \partial A$. A could be any geometric object, e.g. regions, lines, points, etc. In on-line handwriting, we will analyze spatial relations between strokes (lines). We use an example for explaining how to categorize binary topological relations between two lines. Eq. (2.12) shows a binary relation matrix between two objects (lines) A and B where $\emptyset$ means no intersection between them while they are intersected using $\neg\emptyset$. Fig. 2.11 shows a corresponding topological relation between two lines. Ref. [37] deduces 33 relations which can be realized between simple lines using the binary relation matrix.

$$\begin{array}{cc} & \begin{array}{ccc} B^\circ & \partial B & B^- \end{array} \\ \begin{array}{c} A^\circ \\ \partial A \\ A^- \end{array} & \begin{pmatrix} \neg\emptyset & \emptyset & \neg\emptyset \\ \emptyset & \emptyset & \neg\emptyset \\ \neg\emptyset & \neg\emptyset & \neg\emptyset \end{pmatrix} \end{array} \tag{2.12}$$

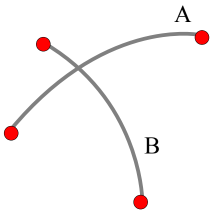


Figure 2.11: Corresponding topological relations between two lines in [37]

In our work [38], we first define spatial relation features at the three levels, and then use a clustering technique to discover spatial relations rather than some predefined spatial relations. The learned spatial relations (edges) are applied to discover the graphical symbols in relational graphs. In the next section, we will discuss the clustering techniques which will be used to generate spatial relation prototypes and to generate graphical symbol prototypes.

2.3 Clustering Techniques

It exists many clustering methods in the state of the art: *k-means* [39], Self-Organizing Map (SOM) [40], Neural Gas (NG) [41], Growing Neural Gas (GNG), hierarchical clustering [42], etc. The clustering algorithm *k-means* consists in iteratively seeking k mean feature vectors (prototypes), and then n samples are partitioned into k clusters in which each sample belongs to the cluster with the nearest prototype (center). The sample space is divided into Voronoi cells. However, the k prototypes are independent from each other. SOM, NG, and GNG contain some topological relationships between prototypes. SOM, a kind of artificial neural network, can produce a discretized representation as a fixed lattice in a low-dimension (typically two-dimension) space of the input space of training samples. We can see the lattice as a map for data visualization. Rather than the fixed lattice, NG has a more flexible topological relationship between prototypes. In the fixed lattice, neighbors of a prototype are fixed while neighbors of a prototype in NG can be changed. A modified version of NG is GNG whose prototype number can be changed. It starts with a small prototype number, and prototypes could be added or be removed in each iteration. Hierarchical clustering seeks to build a hierarchy of clusters. Two general approaches exist: agglomerative (bottom up) and divisive (top down). Divisive clustering is conceptually more complex than agglomerative clustering [43]. In this thesis, only two clustering techniques will be used, k-means for spatial relation learning and agglomerative hierarchical clustering for multi-stroke symbol learning. In the next section, *k-means* will be presented in detail.

2.3.1 K-Means

The algorithm *k-means* is seeking k mean vectors (prototypes) $M = (\mu_1, \mu_2, ...\mu_k)$, which correspond to k classes, $\Omega = \{C_1, C_2, ...C_k\}$. We assume n samples $X = (x_1, x_2, ..., x_n)$ and know the number of clusters k in advance. Firstly, k samples are randomly selected for the cluster centers to initialize the *k-means* iterative procedure. For each iteration, we have to update each sample membership and recalculate the prototypes M.

In order to simplify the description of this problem, only the square Euclidean distance has been considered. We define the membership function $P(C_i|x_p)$ that determines whether $x_p \in X$ belongs to the class C_i with the mean vector μ_i using the nearest squared Euclidean distance $||x_p - \mu_i||^2$.

$$P(C_i|x_p) = \begin{cases} 1 \quad if\ \mu_i = \arg\min\limits_{\mu_j \in M} ||x_p - \mu_j||^2 \\ 0 \qquad\qquad otherwise \end{cases} \tag{2.13}$$

Eq. (2.13) is applied for allocating each sample for its cluster. After each sample gains a cluster, we have to update all the mean values M using an iterative equation Eq. (2.14).

$$\mu_i = \frac{\sum\limits_{k=1}^{n} P(C_i|x_k)x_k}{\sum\limits_{k=1}^{n} P(C_i|x_k)} \tag{2.14}$$

A pseudo algorithm for k-means clustering is defined as:

1. Begin.
2. Initializing a number of clusters k, and $(M = \mu_1, \mu_2, ...\mu_k)$ randomly.
3. Allocating all the samples with its cluster using the square Euclidean metric via Eq. (2.13).
4. Updating mean values via Eq. (2.14).
5. If mean values are changed, we go to the step 3.
6. End.

Consequently, k prototypes are attained until means values are stable. The algorithm k-means can be easily implemented. Once we embed data into a fixed-length feature space, k-means can be used for clustering. In the next section, we will introduce agglomerative hierarchical clustering.

2.3.2 Agglomerative Hierarchical Clustering

Rather than embedding data into a fixed-length feature space, agglomerative hierarchical clustering require only a pair-wise distance matrix between all the data. It starts a clustering procedure with singleton clusters; every single data is a cluster. Given data $X = \{x_1, x_2, ..., x_n\}$, each of them is a cluster $\Omega = \{C_1, C_2, ..., C_n\}$ where $C_i = \{x\}$. The algorithm of agglomerative hierarchical clustering is described as:

1. Begin.
2. Searching for the cluster pair with the closest distance, $(i, j) = \underset{C_i, C_j \in \Omega}{\arg\min}\, dist(C_i, C_j)$.
3. Merging two sets C_i and C_j becomes a new cluster C_k, and $\Omega = (\Omega - C_i - C_j) \cup C_k$.
4. Return to the step 2 until clustering results satisfy a criterion.
5. End.

In this algorithm, $dist(C_i, C_j)$ represents a distance between two clusters. At the end, we will get a *dendrogram* that records each distance when two clusters are merged. In this thesis, six distances will be used, *Single*, *Average*, *Complete*, *Centroid*, *Median*, and *Ward*.

1. *Single*: *Single* metric computes the smallest distance between two clusters.

$$dist(C_i, C_j,' Single') = \min_{x \in C_i, x' \in C_j} ||x - x'|| \tag{2.15}$$

2. *Average*: *Average* metric computes the average distance between two clusters.

$$dist(C_i, C_j,' Average') = \frac{1}{n_i n_j} \min_{x \in C_i, x' \in C_j} ||x - x'|| \tag{2.16}$$

3. *Complete*: *Complete* metric computes the largest distance between two clusters.

$$dist(C_i, C_j,' Complete') = \max_{x \in C_i, x' \in C_j} ||x - x'|| \tag{2.17}$$

4. *Centroid*: *Centroid* metric computes the Euclidean distance between the centroids of two clusters.

$$dist(C_i, C_j,' Centroid') = ||m(C_i) - m(C_j)|| \tag{2.18}$$

   where $m(C_i) = \frac{1}{|C_i|} \sum\limits_{x \in C_i} x$.

5. *Median*: *Median* metric computes the Euclidean distance between weighted centroids of two clusters.

$$dist(C_i, C_j,' Median') = ||\tilde{x} - \tilde{x}'|| \tag{2.19}$$

   where $\tilde{x}$ is created by a fusion of two clusters p and q and $\tilde{x}$ is recursively defined as:

$$\tilde{x} = \frac{1}{2}(\tilde{x}_p + \tilde{x}_q) \tag{2.20}$$

6. *Ward*:

   Ward uses an increase of sum of squares as a result of joining two clusters.

$$dist(C_i, C_j,' Ward') = \sqrt{\frac{2|C_i||C_j|}{(|C_i| + |C_j|)}}||m(C_i) - m(C_j)|| \tag{2.21}$$

In this thesis, we will use these six metrics for grouping multi-stroke symbols. In the next section, two criteria will be introduced for evaluating clustering results.

2.3.3 Evaluating Clusters

After generating clusters, we need measures for evaluating quality of the distances mentioned in this chapter. A better clustering method should attain a high intra-cluster similarity and a low inter-cluster similarity. In this thesis, we will apply both *Purity* and *Normalized Mutual Information* (NMI) [43, 44] for quality assessment.

We accumulate the biggest numbers of major class in each cluster, and *Purity* is equal to a ratio of the sum of accumulation and the total data number. A formal definition of *Purity* is described as following. Given a set of clusters $\Omega = \{w_1, w_2, ..., w_{n_p}\}$ and a set of classes $\mathbb{C} = \{c_1, c_2, ..., c_J\}$, *Purity* is defined by:

$$purity(\Omega, \mathbb{C}) = \frac{1}{N} \sum_{w_k \in \Omega} \max_{c_j \in \mathbb{C}} |w_k \cap c_j|, \tag{2.22}$$

where N is the number of all the data (symbols). We can see that more clusters, and a higher purity. Singleton clusters reach the highest purity of 1.

Rather than a simple criterion *Purity*, NMI will be penalized by an increasing number of clusters. NMI is defined by:

$$NMI(\Omega, \mathbb{C}) = \frac{I(\Omega; \mathbb{C})}{[H(\Omega) + H(C)]/2}. \tag{2.23}$$

I is the mutual information between Ω and $\mathbb{C}$ [1] defined by:

$$\begin{aligned} I(\Omega; \mathbb{C}) &= \sum_{w_k \in \Omega} \sum_{c_j \in \mathbb{C}} P(w_k \cap c_j) \log \frac{P(w_k \cap c_j)}{P(w_k)P(c_j)} \\ &= \sum_{w_k \in \Omega} \sum_{c_j \in \mathbb{C}} \frac{|w_k \cap c_j|}{N} \log \frac{N|w_k \cap c_j|}{|w_k||c_j|} \end{aligned} \tag{2.24}$$

where N is the number of all the symbols, and $P(w_k \cap c_j)$ denotes a probability of a symbol being in the cluster w_k and in the class c_j (the intersection of w_k and c_j). H is the entropy as defined by:

$$\begin{aligned} H(\Omega) &= - \sum_{w_k \in \Omega} P(w_k) \log P(w_k) \\ &= - \sum_{w_k \in \Omega} \frac{|w_k|}{N} \log \frac{|w_k|}{N} \end{aligned} . \tag{2.25}$$

NMI in Eq. (2.23) is in a range between 0 and 1 [43]. NMI$=0$ means a random

[1] In this thesis, *log* uses a base of 2 as default.

choice on classes. A higher value is preferable. We can see that NMI in Eq. (2.23) is normalized by $[H(\Omega)+H(\mathbb{C})]/2$, a function in terms of the cluster number where $H(\mathbb{C})$ is invariant with the cluster number and $H(\Omega)$ reaches the maximum value $\log N$ for $n_p = N$.

However, for two clustering results with the same number of clusters $|\Omega_1| = |\Omega_2|$, a higher *Purity*($Purity(\Omega_1, \mathbb{C}) > Purity(\Omega_2, \mathbb{C})$) does not mean a higher NMI. It is possible that: $NMI(\Omega_1, \mathbb{C}) < NMI(\Omega_2, \mathbb{C})$.

For example, Fig. 2.12 displays $|\Omega_1| = 3$ clusters containing digits with $|\mathbb{C}| = 3$ labels (classes), $\mathbb{C} = \{$ “2”, “4”, “7” $\}$. In total, there are $N = 18$ instances of symbol.

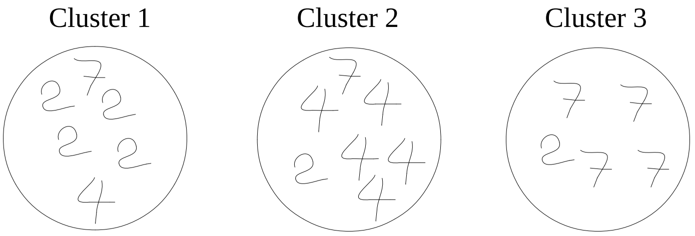


Figure 2.12: Example of three clusters for three classes (three handwritten digits “2”, “4”, and “7”)

Purity can be easily computed by:

$$Purity(\Omega_1, \mathbb{C}) = \frac{4+5+4}{18} = 0.72, \tag{2.26}$$

and the mutual information between Ω_1 and $\mathbb{C}$ is defined by:

$$\begin{aligned} I(\Omega_1; \mathbb{C}) = \frac{1}{18}(4 \times \log\frac{18\times 4}{6\times 6} + 1\times\log\frac{18\times 1}{6\times 6} + 1\times\log\frac{18\times 1}{6\times 6} \\ +1\times\log\frac{18\times 1}{7\times 6} + 5\times\log\frac{18\times 5}{7\times 6} + 1\times\log\frac{18\times 1}{7\times 6} \\ +1\times\log\frac{18\times 1}{5\times 6} + 0 + 4\times\log\frac{18\times 4}{5\times 6}) = 0.33. \end{aligned} \tag{2.27}$$

The entropies of Ω_1 and $\mathbb{C}$ are computed by:

$$H(\Omega_1) = -1\times\frac{6}{18}\log(\frac{6}{18}) - 1\times\frac{7}{18}\log(\frac{7}{18}) - 1\times\frac{5}{18}\log(\frac{5}{18}) = 1.57, \tag{2.28}$$

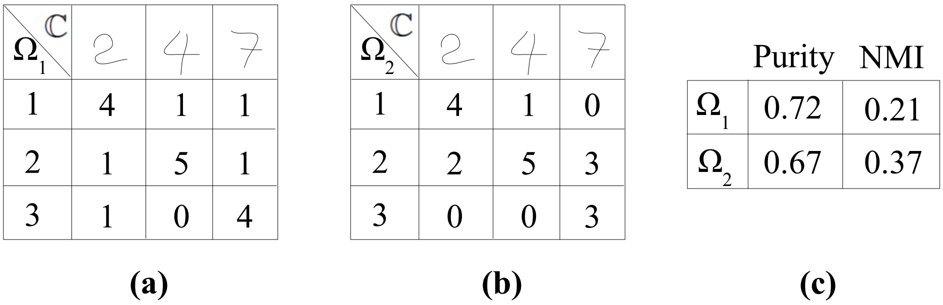
and

$$H(\mathbb{C}) = -1 \times \frac{6}{18}\log(\frac{6}{18}) - 1 \times \frac{6}{18}\log(\frac{6}{18}) - 1 \times \frac{6}{18}\log(\frac{6}{18}) = 1.59. \quad (2.29)$$

At the end, the normalized mutual information is obtained by:

$$NMI(\Omega_1, \mathbb{C}) = \frac{0.33}{(1.57 + 1.59)/2} = 0.21. \quad (2.30)$$

Fig. 2.13 shows two distribution matrix for two clustering results respectively. The example in Fig. 2.12 is represented by the distribution matrix in Fig. 2.13a. We change the distribution from Ω_1 to Ω_2 in Fig. 2.13b by keeping 3 clusters. *Purity* and NMI are obtained in Fig. 2.13c. The scatter non-major classes result in lower $NMI(\Omega_1, \mathbb{C})$ even Ω_1 has a higher purity.



Ω_1 \ $\mathbb{C}$	2	4	7
1	4	1	1
2	1	5	1
3	1	0	4

(a)

Ω_2 \ $\mathbb{C}$	2	4	7
1	4	1	0
2	2	5	3
3	0	0	3

(b)

	Purity	NMI
Ω_1	0.72	0.21
Ω_2	0.67	0.37

(c)

Figure 2.13: Two clustering results (a) and (b) with a same number of 3 clusters

In the next section, we will present relative works on generating a codebook in handwriting using clustering. The interest of this codebook, when available, is to allow the user to manually labeled each symbol, eventually containing several strokes, and then to propagate this labeling to the raw data.

2.4 Codebook Extraction in Handwriting

In this section, we discuss the codebook generation at two levels: the single-stroke symbol level and the multi-stroke symbol level.

Extracting a codebook from single strokes in the field of both offline and on-line handwriting has gained increased attention. Many offline biometric systems [45–47] generate a codebook using a clustering method (e.g. *k-means*, self-organizing map, etc.) so that a codebook-based probability distribution function can be employed to identify or verify the writers. It is necessary to cut the offline ink (to segment it) beforehand [45, 46], then basic components are extracted for generating the codebook. In these cases the codebook aims to be representative of a writer, but not to match to language symbols. Considering the on-line handwriting, basic elements are the strokes, so we can build directly a codebook at the single-stroke level in this thesis.

A multi-stroke codebook based on the *k-means* algorithm is built in [39] for clustering the different allographs of the same letter but beforehand a complete segmentation and recognition tool is applied on the text document. Varied features are used for this clustering. The two-dimensional graphical symbols containing several strokes are re-sampled into a fixed number of points, and then embedded in a feature vector space so that we can compute a distance between two graphical symbols. However, the order of strokes has not been discussed; the embedding is stroke-order-sensitive. We need a stroke-order-free algorithm to obtain the distance between two graphical symbols composed of many strokes; writers may copy a same symbol with different stroke orders and different stroke directions. Moreover, segments comprising different numbers of strokes would be the same symbol (stroke-number-free problem). In our work, we make use of a Modified Hausdorff Distance (MHD), which is widely used in contour matching on offline data (images) [35, 48], in order to avoid the problem of stroke-order, stroke-direction, and stroke-number (see Section 3.5.3 in detail).

After codebook generation, we have to manually label symbols. [14, 15] only give a label to a correctly segmented character. In reality, it is difficult to build clusters that contain only well segmented symbols. Many clusters would mix the symbols with the sub-parts of other symbols. We have to create a symbol mapping from labeled symbol instances to raw symbol instances. Furthermore, we need a criterion for evaluating how much work we have been reduced.

2.5 Conclusion

Building the ground-truth dataset at the symbol level is a tedious work. Two main steps exist in the ground-truth dataset creation: symbol segmentation and symbol labeling. We propose to organize the graphical language as relational sequences and relational graphs. The MDL principle on sequences and on graphs have been presented to automatically generate the symbol segmentation. We can group segmented symbols into a codebook using clustering algorithms. A codebook will be generated so that we can label symbols at the codebook level, which can save symbol annotation workload. In the next chapter, we start to discuss several similarities between two isolated symbols and symbol quantization using hierarchical clustering. The multi-stroke symbol codebook can be generated for labeling.

3

Quantifying Isolated Graphical Symbols

As every instance of a handwritten graphical symbol is different one from the other because of the variability of human handwriting, it is of prime importance to be able to compare two patterns and further more to define clusters in the feature space of patterns which are very similar. In this chapter, we introduce a hierarchical clustering which is convenient in a context of unsupervised clustering. This clustering algorithm requires a pairwise distance matrix between all the graphical symbols. We mainly discuss a distance between two graphical symbols which are two sets of sequences. According to the composition of graphical symbol, we can divide discussion into two categories, a distance between two single-stroke symbols and a distance between two multi-stroke symbols. With respect to the first category, the famous Dynamic Time Warping (DTW) based distance allows an elastic matching between two strokes. It is considered as an efficient algorithm for single stroke patterns. To deal with multi-stroke patterns, a first solution consists in a simple concatenation of the strokes respecting a natural order, which is most of the time the temporal order. However, as we will see with some examples introduced later in this chapter, this solution is not always satisfying. To allow more flexibility in the construction of the stroke sequence we will propose a novel algorithm, called DTW-A* (DTW A star). As it turns out to be a very time consuming method, we have also proposed a modified Hausdorff distance (MHD) to allow even more flexibility in the matching process while reducing the computation time. In that last case, all the temporal constraints of the point sequences are ignored. To analyze the behavior of these distances and of the hierarchical clustering algorithm, two datasets have been considered. One is the *Calc* dataset, it is composed of single line mathematical expressions, the second one is more challenging, it is the *FC* dataset with handwritten flowcharts.

3.1 Introduction

In on-line handwriting, basic elements are strokes. Each stroke contains a sequence of points, from a pen-down point to a pen-up point. Hence, the stroke is oriented. A graphical symbol is composed of one or several strokes. To automatically quantify graphical symbols, a clustering technique is required for grouping symbol shapes. As reminded in the state of the art section, it exists many clustering methods, hierarchical clustering [42], *k-means* [39], self-organizing map [40, 49], neural gas [41], etc. For implementing the clustering, a common necessary condition is to be able calculate a distance (or a similarity) between two symbols. In this chapter, we will discuss the distance between two isolated multi-stroke graphical symbols, equivalently between two sets of point sequences.

Different people may write a visually same symbol with different stroke directions and different stroke orders. In writer identification, these characteristics can efficiently distinguish writers [39]. However, to understand or communicate the same symbol written by different writers, stroke direction and stroke order should be ignored. We human read handwritten symbols without knowing the stroke direction and the stroke order. For instance, a symbol containing a horizontal stroke "−" can be written by two different approaches, from left to right "→" or an inverse way "←".

DTW (Dynamic Time Warping) is an algorithm which computes a distance between two single-stroke symbols. It obeys a continuity constraint and a boundary constraint during point-to-point matching [12]. These two constraints will be elaborated in Section 3.4.1. Comparing two opposed direction strokes, the distance DTW $dist_{DTW}(\rightarrow, \leftarrow)$ naturally produce a large value because of two inverse directions. A simple solution is to choose the smallest distance between two possible directions of one stroke: $min(dist_{DTW}(\rightarrow, \leftarrow), dist_{DTW}(inv(\rightarrow), \leftarrow))$ where $inv(.)$ is an operator of reversing stroke trajectory direction.

However, when comparing two multi-stroke symbols, the number of possible directions and orders increases very fast in terms of a growing stroke number. Tab. 3.1 illustrates an example of how to write "E" within four strokes. With this example, 384 different writing sequences are possible. This example shows the complexity of combinations of different stroke directions and stroke orders. In general, the number of different temporal writing ways for a symbol is given by:

$$S_N = N! \times 2^N = 2 \times N \times S_{(N-1)} \qquad (3.1)$$

where N is the stroke number of a symbol. For calculating the distance DTW between two multi-stroke symbols, a simple solution is to concatenate the strokes using different stroke directions and stroke orders.

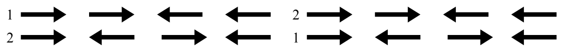
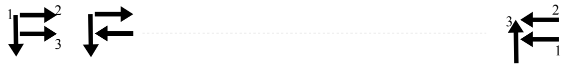
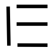
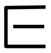
Stroke Number (N)	Example	Combination Number (S)	Writing Method Temporal Illustration
1		2	
2		8	
3		48	
4		384	

Table 3.1: Variability of stroke order and direction in an on-line handwritten symbol

For example, for the distance DTW between  (4 strokes) and  (2 strokes), we should calculate 384 × 8 = 3092 possible combinations. This large combination number is due to different writing orders of N strokes (N!) and due to the two directions of each written order (2^N).

In a more extreme case, we can get rid of all the temporal information and consider the symbols as a set of points ignoring the sequences they produce. This leads to use the Hausdorff distance [35]. This metric is used in image processing domain

[35]. More formally the Hausdorff distance is defined in Eq. (2.4). Furthermore, another varied version is a modified Hausdorff distance [48]. But the matching of these two distances does not verify the intra-sequence continuity constraint.

The continuity constraint mentioned before concerns the temporal dimension. Many works [50–52] extend the temporal continuity constraint (temporal sequence warping) to a spatial continuity constraint in two spatial dimensions (two-dimension warping). But the temporal continuity constraint of sequences is ignored. In this chapter, we proposed a distance DTW A* between two multi-stroke symbols by keeping the temporal continuity constraints.

Once a distance is selected, hierarchical clustering is used to group symbols into n_p clusters. At the end, we will introduce cluster quality criteria for comparing performance between distances.

In this chapter, we will first present the clustering technique for grouping symbols in Section 3.2. To implement the clustering technique, we will study three distances using proposed local features between two symbols in Section 3.4 at a single-stroke symbol level and in Section 3.5 at a multi-stroke symbol level, and then the distances will be compared in Section 3.7 using cluster quality criteria introduced in Section 2.3.3.

3.2 Hierarchical Clustering

At that stage, we assume that all the symbol instances are well segmented in the handwritten documents. The goal of this chapter is to quantify these symbol instances; grouping the instances into n_p symbols. A clustering technique is used for producing a codebook (a symbol set), which is then brought into play for computing the membership of each symbol instance. We have chosen an agglomerative hierarchical clustering [42] since it only needs a pairwise distance matrix between all the segments. Furthermore, we can also easily tune the number of prototypes from the dendrogram. We use the Lance-Williams formula [42] which provides an efficient computational algorithm to generate a dendrogram. Then, the membership of symbol instance is generated: all the instances are grouped into n_p clusters (symbols). In the next section, we will present local features for each point, and then three distances using these features will be introduced for implementing hierarchical clustering.

3.3 Extracting Features for Each Point

In this section, for designing a distance between two symbols, 12 local features for each point will be extracted. We assume a multi-stroke symbol which is a set of point sequences (strokes) $sym = \{str_1, str_2, ...\}$. Each stroke is a point sequence, $str = (p(1), ..., p(i-1), p(i), p(i+1), ...)$ where a point p is defined by its coordinates (x, y). For being size independent, the symbol should be normalized into a reference bounding box $\{x \in [-1, 1], y \in [-1, 1]\}$ by keeping an original ratio, and re-sampled into a fixed number of points. During re-sampling, handwriting velocity information is lost since we only need a shape characteristic.

The designed features should be independent of a written trajectory direction. In addition to raw data (x, y), we use local orientation features, a local curvature (cosine) and a binary pen-up and pen-down information to have a 12-feature local description of a point. We assume three consecutive points: $p(i-1)$, $p(i)$, and $p(i+1)$. Three levels of features are defined: coordinate level, orientation level, and curvature level. The coordinate level uses only x and y coordinates.

We assume a reference orientation. Usually between a written orientation and the reference orientation, two supplementary angles can be easily found: one is in the range $[0, \pi/2]$ and another is in the range $[\pi/2, \pi]$. The sum of these two angles is equal to π. We define the angle located in the range $[0, \pi/2]$ as θ. A similarity is computed as $\cos(\theta)$ for a reference orientation. We can observe that when $\theta = 0, \cos(0) = 1$ (the same orientation), and when $\theta = \pi/2, \cos(\pi/2) = 0$ (two orthogonal orientations). Hence, concerning a reference orientation, a directional feature will be obtained.

Fig. 3.1 shows an example for the written direction "$p(i-1) \rightarrow p(i+1)$" and a reference orientation. Two supplementary angles are found, $\theta \in [0, \pi/2]$ and $\theta' \in [\pi/2, \pi]$. We choose θ to calculate the similarity as $cos(\theta)$ with respect to the reference orientation. One reference orientation corresponds to one similarity of written direction. In this example, we can see the reference orientation separates the directional space into two symmetry spaces, left and right. Obviously, for each written direction, we can find another written direction with the same similarity value. For instance, two different written directions, “$p(i-1) \to p(i+1)$” and “$p(i-1)' \to p(i+1)'$”, have the same similarity of the reference orientation.

Thus, to avoid this ambiguity, we add another orthogonal reference orientation as shown in Fig. 3.2. In this figure, two reference orientations (Ref1, and Ref2) are used to measure the written direction “$p(i-1) \to p(i+1)$”. We assume that $\theta = \pi/4$. Considering the two reference orientations, a feature vector $(cos(\pi/4), cos(\pi/4))$ will be obtained. However, we can also find another symmetry written direction, which can contribute the same feature vector $(cos(\pi/4)$, $cos(\pi/4))$. This written direction is illustrated in Fig. 3.3. We can use “$p(i-1) \to p(i+1)$” and “$q(i-1) \to q(i+1)$” as the two new reference orientations. Hence, this ambiguity can be avoided.

Similarly, adding more reference orientations can avoid this ambiguity. In this thesis, we use eight reference orientations to define 8 features. The reference orientations are defined in Fig. 3.4. Each reference orientation shifts to another by $\pi/8$.

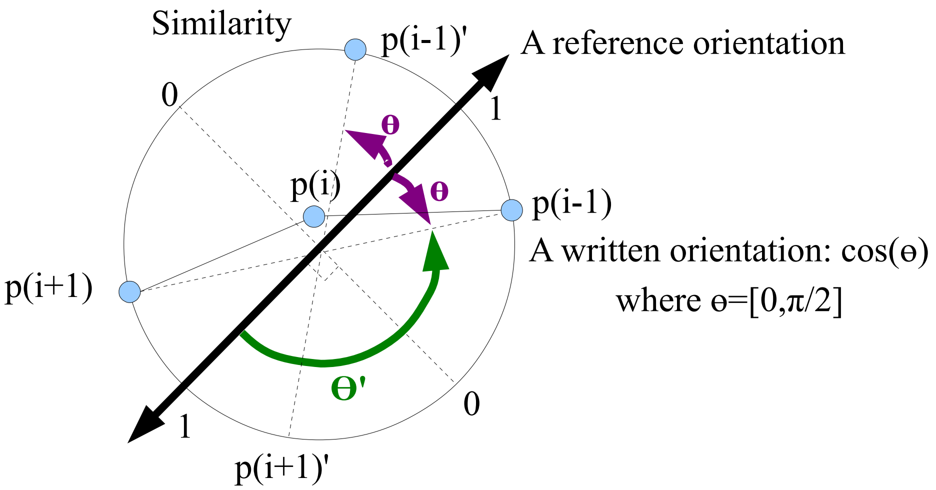


Figure 3.1: Defining a reference orientation, and its similarity value between the reference orientation and a written orientation

The curvature angle φ in Fig. 3.5 is between “$p(i) \to p(i-1)$” and “$p(i) \to p(i+1)$”. The curvature angle here is only expressed in cosine, but not in sine. Since the curvature angle varies only in $[0, \pi]$ and its corresponding cosine value range is in $[-1, 1]$ which is a bijective function.

A binary pen-up (-1) and pen-down (1) information is the last feature. To compare two multi-stroke symbols, the basic DTW algorithm first links the different

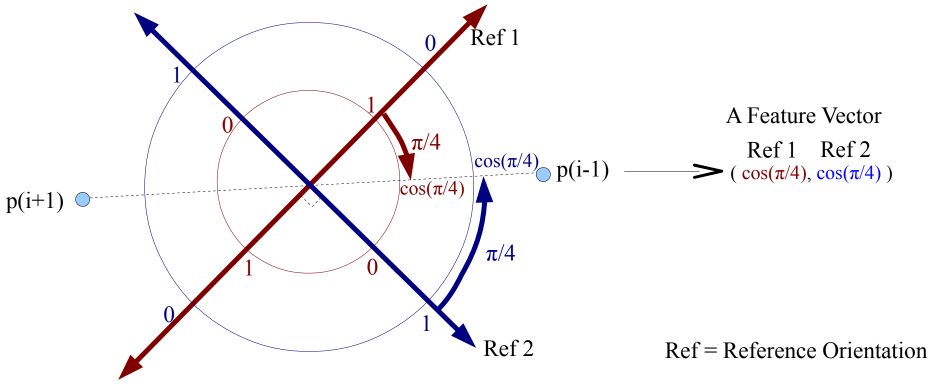


Figure 3.2: Two orthogonal reference orientations are defined to get a feature vector

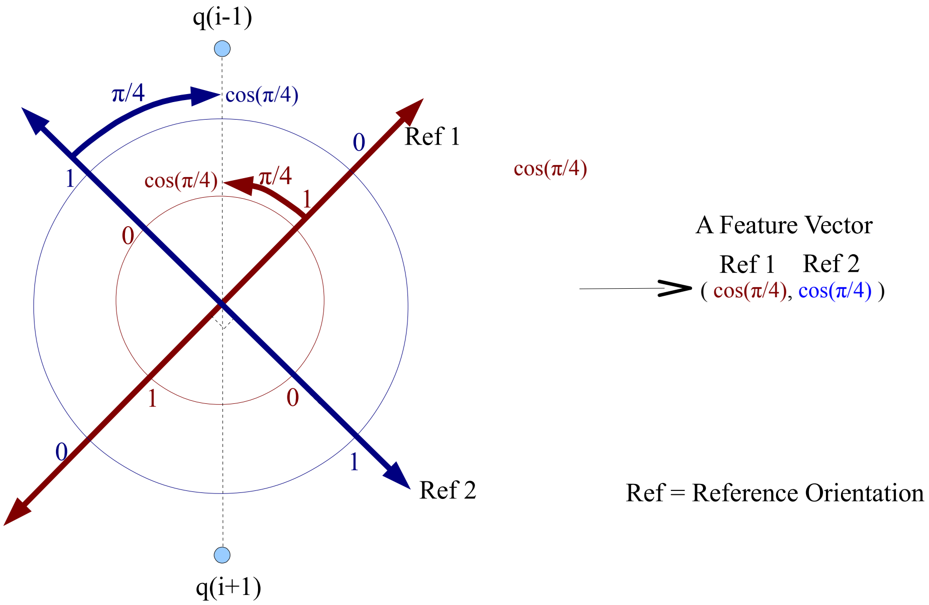


Figure 3.3: The symmetry written direction with the written direction as shown in Fig. 3.2

strokes together to form a unique stroke. Fig. 3.6 shows an example of a two-stroke symbol “4”. The two strokes are concatenated to one stroke. A new stroke is interpolated. Points in the two original strokes are marked with a pen-down (+1), and interpolated points in the new stroke are marked as a pen-up feature (-1).

Using these 12 features, the Euclidean distance $dist(pt_1, pt_2)$ between two points pt_1 and pt_2 can be calculated in a 12-dimension feature space. In the next two sections (Section 3.4 and Section 3.5), we will discuss distances between two symbols (single-stroke and multi-stroke) applying the Euclidean distance $dist(pt_1, pt_2)$.

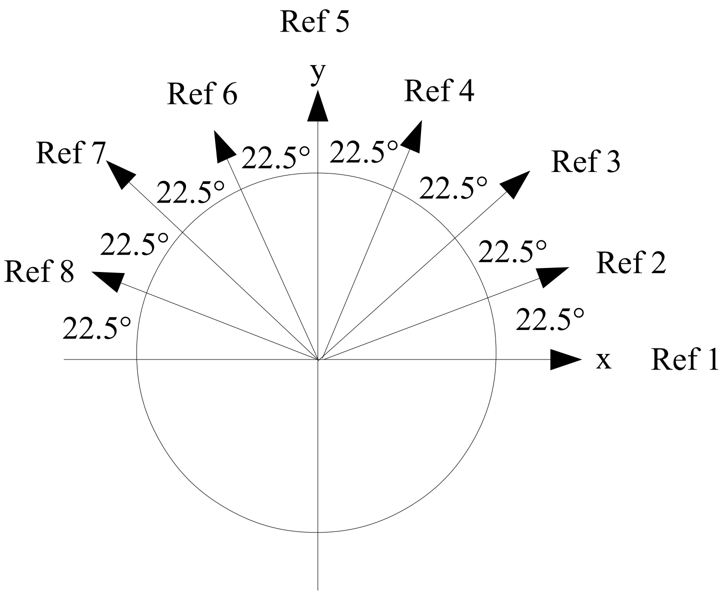


Figure 3.4: Eight reference orientations

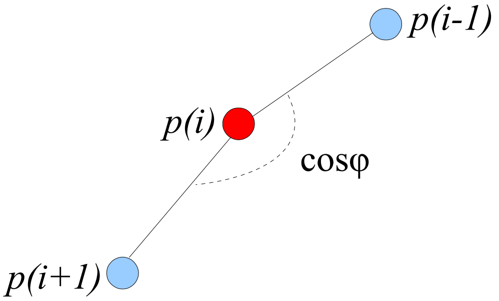


Figure 3.5: A curvature feature of the point $p(i)$

3.4 Matching between Two Single-Stroke Symbols

In on-line handwriting, basic elements are strokes. Each stroke is a point sequence. A simple case is when each symbol contains only one stroke. Therefore, we can consider each stroke as a temporal sequence. Calculating a distance between two strokes is then possible using the DTW (Dynamic Time Warping) algorithm [12]. In this section, we will give a brief description of DTW as an introduction for the multi-stroke symbol matching case.

3.4.1 Dynamic Time Warping

We assume that two strokes (two time-varying-data sequences) denoted as:

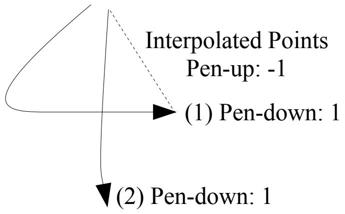


Figure 3.6: A binary pen-up (-1) and pen-down (1) feature

$$S_1 = (p_1(1), ..., p_1(N_1))$$

and

$$S_2 = (p_2(1), ..., p_2(N_2))$$

will be compared. The algorithm DTW can be used for calculating the distance between two sequences whose data vary in time. This method has been applied first in speech processing aiming at matching two temporal varing acoustic samples. Analogously, on-line handwritten strokes contain temporal varing information, and can be matched by the algorithm DTW. Even it is time-consuming, many works [12, 53, 54] have shown its efficiency.

Main principles of the algorithm DTW [12] are summarized as follows. Given a warping path $P(h) = (i(h), j(h)), 1 \leqslant h \leqslant H$ defining point-to-point associated pairs where h is a pair index from the $i(h)$*th* point in S_1 and from the $j(h)$*th* point in S_2.

$P(h)$ should respect a boundary constraint and a continuity constraint. The first boundary constraint is defined by:

$$\begin{aligned} &P(1) = (i(1), j(1)) = (1, 1),\\ &P(H) = P(i(H), j(H)) = (N_1, N_2). \end{aligned} \tag{3.2}$$

It means that the first two beginning points should be matched in the two strokes, and so do the two ending points. Eq. (3.3) explains the second temporal continuity constraint:

$$(\triangle\, i(h), \triangle\, j(h)) = (i(h) - i(h-1), j(h) - j(h-1)) = \begin{cases} (1,0) or \\ (0,1) or \\ (1,1). \end{cases} \tag{3.3}$$

This constraint implies the point-to-point matching shift between two sequences is at most and at least one. In addition, all the points are matched at least once. Calculating the distance between two sequences involves the search of a warping path (a point-to-point associated pair sequence) that minimizes the sum of the point-to-point associated cost function:

$$D(S_1, S_2) = \min_{P(h)} \sum_{h=1}^{H} dist(p_1(i(h)), p_2(j(h))), \tag{3.4}$$

where $dist(.,.)$ is the Euclidean distance in the point feature space as defined in Section 3.3. This classical DTW distance should be normalized by the number of couples:

$$DTW(S_1, S_2) = \frac{1}{H} \min_{P(h)} \sum_{h=1}^{H} dist(p_1(i(h)), p_2(j(h))). \tag{3.5}$$

The solution for Eq. (3.4) can be resolved by dynamic programming. The dynamic programming searches the minimum warping path from a cumulative distance matrix :

$$D(i, j; h) = d(p_1(i), p_2(j)) + \min \begin{cases} D(i-1, j; h-1) \\ D(i, j-1; h-1) \\ D(i-1, j-1; h-1), \end{cases} \tag{3.6}$$

with $D(i, j; 0) = 0$ for initialization. Once the cumulative distance matrix is computed, we can use backtracking to find the minimum warping path.

Fig. 3.7 illustrates an example of matching two single-stroke symbols. The starting point couple is marked with two red circles. We search the minimum warping path with Eq. (3.6). We first compute a cumulative distance matrix as explained in Fig. 3.8. The best warping path can be found by backtracking [55] from the ending point couple to the starting point couple to obtain: P(1),...,P(9) = (1,1), (2,2), (3,3), (3,4), (4,5), (5,5), (6,6), (6,7), (6,8). We can see that once we define the starting point couple and the ending point couple, the best warping path will be found. In the next section, we will introduce a comparison between two sets of point sequences.

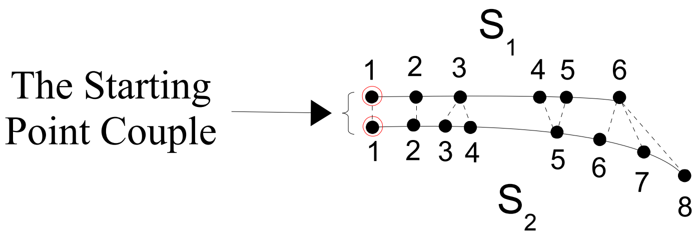
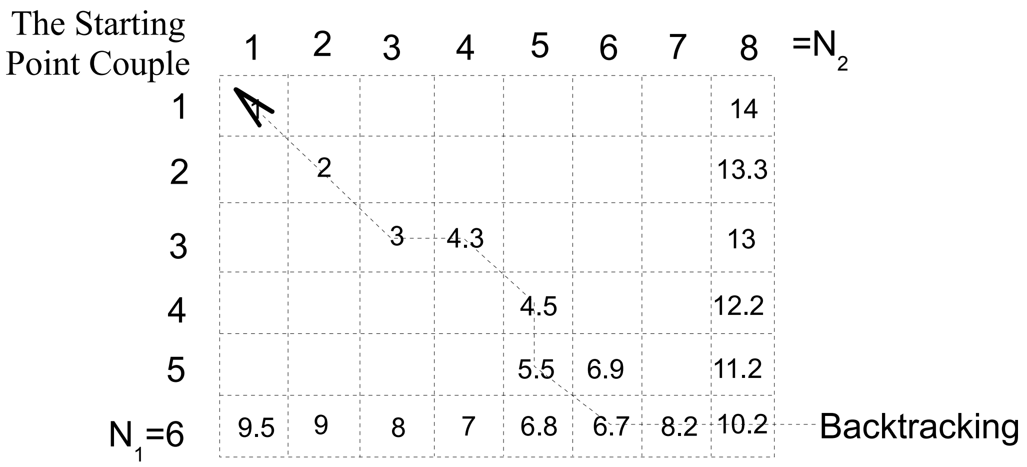


Figure 3.7: Two point sequences (two single-stroke symbols)



	1	2	3	4	5	6	7	8	=N_2
1	1							14	
2		2						13.3	
3			3	4.3				13	
4					4.5			12.2	
5					5.5	6.9		11.2	
N_1=6	9.5	9	8	7	6.8	6.7	8.2	10.2	

Figure 3.8: The cumulative distance matrix $D(i, j; h)$ of Eq. (3.6) illustration and the best warping path.

3.5 Matching between Two Multi-Stroke Symbols

At the beginning of this chapter, we have analyzed the complexity of the comparison between two multi-stroke symbols. In this thesis, we will discuss three different methods for measuring the distance between two multi-stroke symbols. The first one is when all the strokes in a symbol are concatenated by a natural handwritten order. Thus, the multi-stroke comparison problem is converted to the single-stroke comparison problem. We proposed a second method, named DTW A* (DTW A star), which searches the best warping path between two stroke sets [56].

The third distance, *modified Hausdorff distance* (MHD) which usually measure the distance between two raster graphics, will be used for matching two handwritten graphical symbols. In the next three sub-sections, these three approaches are discussed respectively.

3.5.1 Concatenating Several Strokes

The first method is to simply concatenate several strokes in the symbol by a natural handwriting order. A simple example of concatenation is shown in Fig. 3.6, and the symbol "4" is converted into an single-stroke symbol. The information of pen-up is designed for penalizing the different number of strokes. After concatenating strokes, we can use DTW to compute the distance between two single-stroke symbols using the features in Section 3.3. We call this method as classical DTW.

3.5.2 DTW A Star

People write a multi-stroke symbol with different specified handwriting orders. The handwriting order and direction depend on many factors, e.g. education, habitation, personal habits, etc. The stroke order of symbol is not invariant to different people handwriting habits. As analyzed in Tab. 3.1, the combination number of stroke orders and stroke directions is too large. It is too time-consuming to try all the concatenations of a multi-stroke symbol. In this section, we propose a new distance comparing two multi-stroke symbols by keeping the continuity constraint.

For explaining our distance, we use as dynamic programming a point-to-point distance matrix. Given two multi-stroke symbols as shown in Fig. 3.9, rows and columns of this matrix represent the two symbols respectively. The strokes of one symbol are placed in one side (rows or columns). The respective position of the strokes in the two sequences is irrelevant and has not to respect the temporal order.

The main idea consists in iteratively constructing a small warping path until all the points are used. Once we choose a starting point couple, four possible directions of warping path are possible. Each direction represents a point-to-point distance matrix for matching two strokes or two sub-parts from two strokes. In each iteration, we search a warping path, which minimizes warping cost and which finishes at least one stroke (the classical DTW shown in Fig. 3.8). For finding the best warping path, four cumulative distance matrices (Fig. 3.9) are explored for four directions respectively.

For example, given two symbols, one contains two strokes while the other contains one stroke. The two strokes of the first symbol are placed in rows and the stroke of the second symbol is placed in columns (one for each point) as shown in Fig. 3.9. Once we defined a starting point (the blue rectangle in Fig. 3.9), there are four possible matching directions (four possible warping paths).

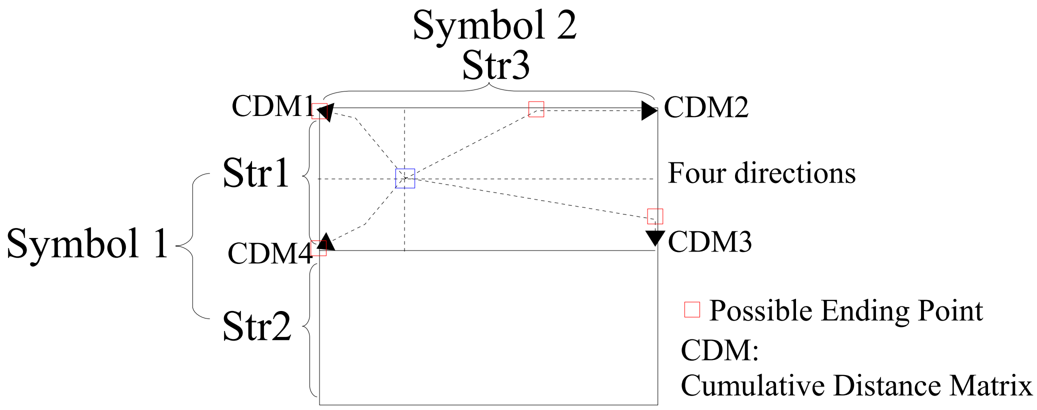


Figure 3.9: Defining a starting point couple (the blue rectangle) and finding a warping path between a 2-stroke symbol (symbol 1) and a single-stroke symbol (symbol 2) in four directions.

In each cumulative matrix, we can apply the classical DTW algorithm as mentioned in Section 3.4.1 to find the minimum cost warping path. We allow, however, DTW to not stop at the diagonal opposed point (the ending point) in the cumulative distance matrix, but along the borders of the matrix (red squares).

In fact in Fig. 3.8 we can find that the warping path stopping at P(9)=(6,8) is not the best as the distance increases after P(7). We can cut this warping path by choosing the minimum distance among the points of the cumulative matrix edges:

$$D(1, N_2), D(2, N_2), ..., D(N_1, N_2)$$

and

$$D(N_1, 1), D(N_1, 2), ..., D(N_1, N_2).$$

In reality, we first calculate the whole cumulative distance matrix until the end of both two strokes. Then the warping path will stop at finishing at least one of two strokes. Thus a new ending point couple will be obtained. For example in Fig. 3.8, we choose the sub-path:(1,1),(2,2),(3,3),(3,4),(4,5),(5,5),(6,6).

With this strategy, starting point couples are chosen for associating the two subsequences with respect to the continuity constraint in each step. In each step, we repeat to choose a starting point couple from non-used points from the two strokes. The searching procedure will be finished until all the points are associated in the warping path. Our objective is to find the warping path that minimizes the associated cost in Eq. (3.4). The distance of DTW A* also is normalized by the number of couples in Eq. (3.5).

Fig. 3.10 shows the best warping path for associating two sets of point sequences. This solution contains four DTW sub-warping paths, which are obtained from step 1 to step 4. The matching directions are not necessary the same. We will search a set of sub-warping paths which minimizes the associated cost (the sum of point-to-point distances).

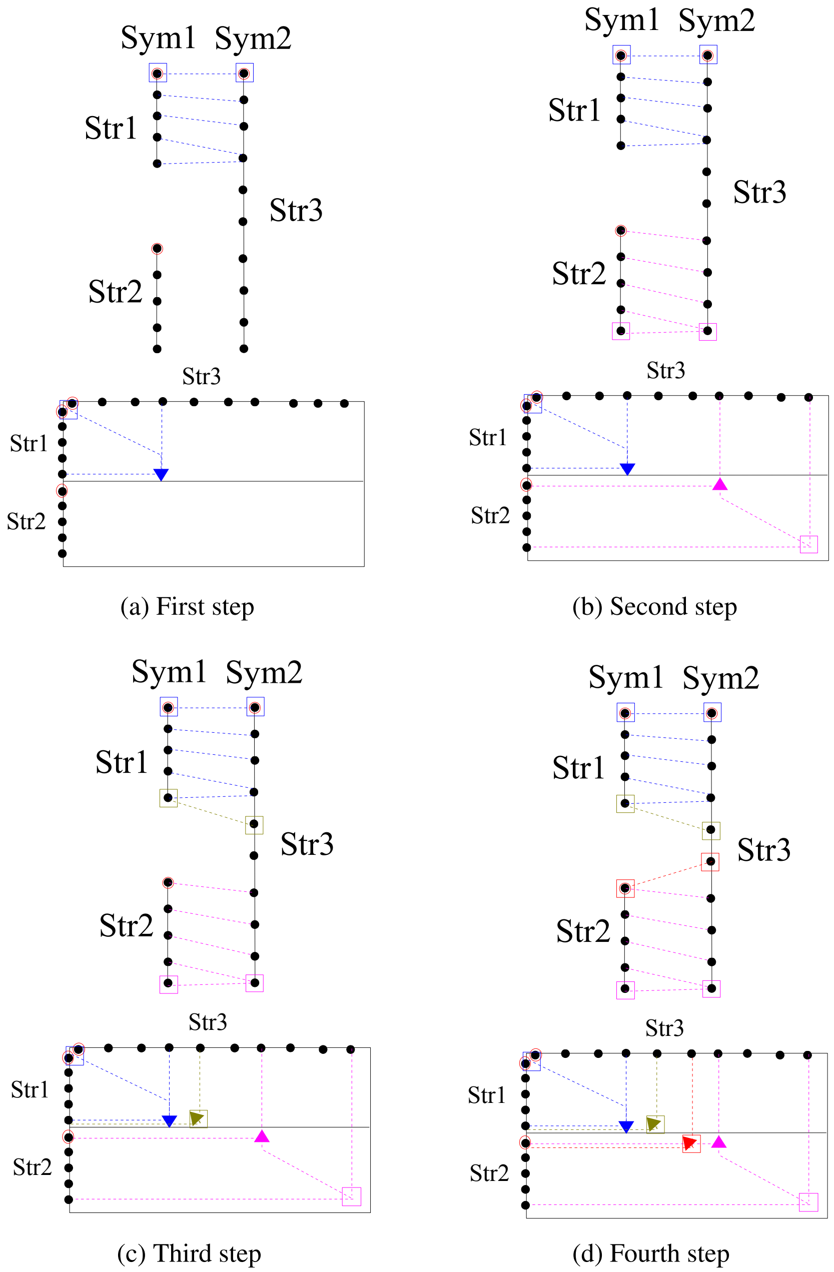
However, there is a large number of possibilities. For searching the best warping path, we will use an A* algorithm [57] which accelerates the search as discussed in the next section.

A* Algorithm

In this section, we will use an A* algorithm (A star) [57] to limit some futureless explorations. The A* algorithm iteratively searches the best path in a graph (a tree in our case) from the starting node (empty associated point) to the ending node (all associated points). However, not all the possible trees are generated because of a heuristic function in the A* algorithm. In each step, only the best hypothesis is explored for the next step.

Fig. 3.11 shows the problem complexity. In fact, there is a large number of possibilities for choosing the starting point couples. From these point couples, there are many potential ending point couples. Since the warping path will cut a stroke into several pieces, the number of possible combinations becomes larger than that in Tab. 3.1.

The A* algorithm uses a distance-plus-cost heuristic function $f(x) = g(x) + h(x)$ of each step x. The cost $g(x)$ gives the cost of best warping path from the starting step to the current step x, and the heuristic cost $h(x)$ estimates the minimum distance to the ending step. Ref. [57] describes the A* algorithm in detail. In this



(a) First step

(b) Second step

(c) Third step

(d) Fourth step

Figure 3.10: A solution of warping path between two symbols (graphic and matrix views)

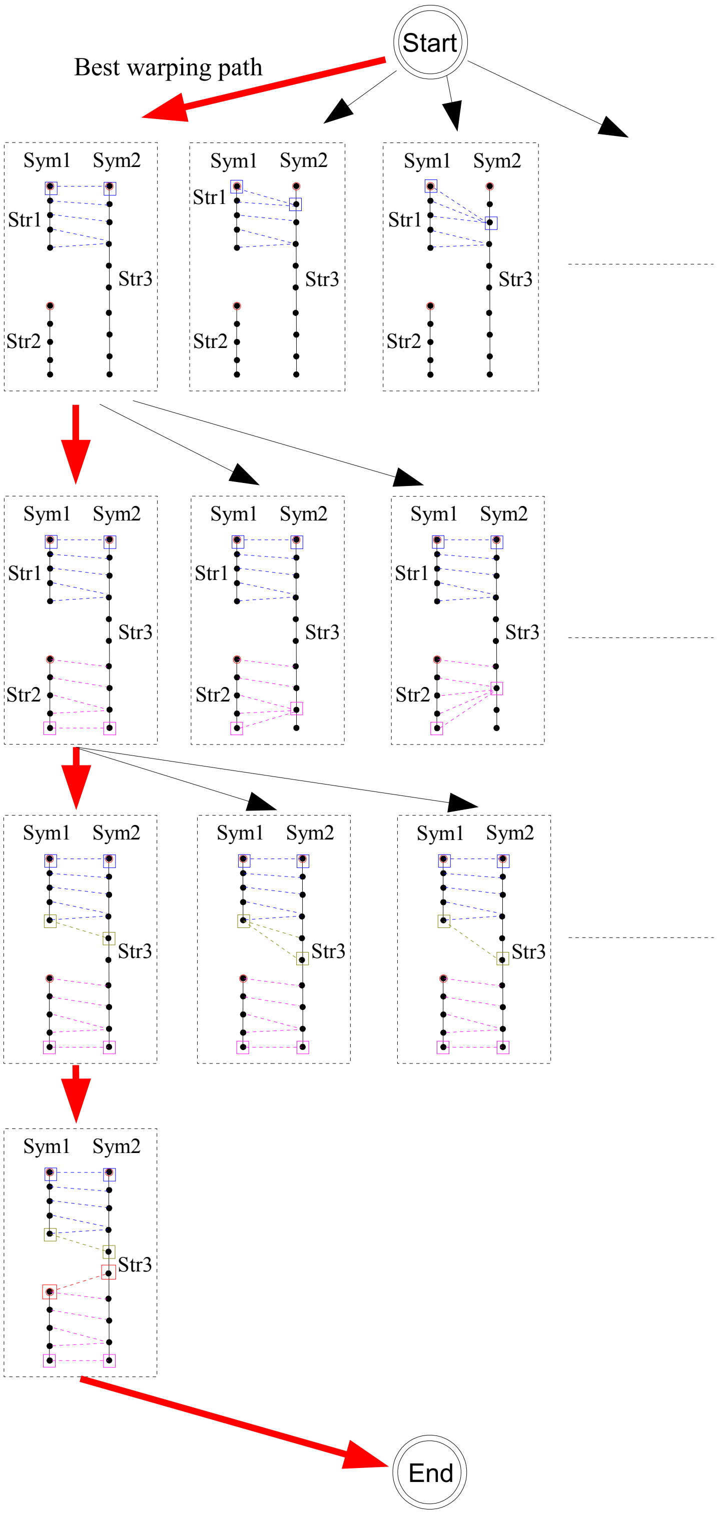


Figure 3.11: An illustration of searching complexity for the best warping path

section, we define the two functions, $g(x)$ and $h(x)$ for our problem. Considering the heuristic distance $h(.)$, it should be as larger as possible but equal to or less than the optimal distance (no estimation) for going to the ending step, which means that $h(.)$ is admissible.

We first define each step x by a warping path $P_x(h) = (i_x(h), j_x(h))$, $1 \leqslant h \leqslant H_x$ between the two symbols $Sym1 = (p_1(1), ..., p_1(N_1))$ and $Sym2 = (p_2(1), ..., p_2(N_2))$. The warping path is a sequence of associated index pairs. Its cost is defined by the sum of pair costs:

$$g(x) = \sum_{1}^{H_x} dist(p_1(i(h)), p_2(j(h))). \tag{3.7}$$

We define a set of non-used points $NUPt(Sym, x)$ for a symbol Sym in a step x. The heuristic cost $h(.)$ therefore can be defined by :

$$h(x) = \frac{1}{2}(h_{sub}(x, Sym1, Sym2) + h_{sub}(x, Sym2, Sym1)), \tag{3.8}$$

where

$$\begin{aligned} h_{sub}(x, SymA, SymB) &= \sum_{p_1(i) \in NUPt(SymA,x)} dist(p_1(i), ppv(p_1, SymB)) \\ ppv(p_1, SymB) &= \underset{p_2(j) \in NUPt(SymB,x)}{\arg\min} dist(p_1, p_2(j)). \end{aligned} \tag{3.9}$$

This heuristic distance $h(.)$ is admissible because we always choose the minimum distance between the two sets of non-used pair points during associating the point pairs.

Even using the A* algorithm to accelerate the searching, the number of combinations is still large. In order to furthermore reduce the number of combinations, we try to limit the number of choosing starting point couples rather than using all the non-used point couples. This strategy will be developed in the next section.

Choosing Starting Point Couples

Generating next steps from the step x, we have to choose a non-used starting point couple, which is used for starting up two sequences with a matching in four directions in maximum. For each direction, a new step will be obtained. Even the A* algorithm can reduce the searching complexity, there are still many possibilities using all the non-used points for a next step. In this section, we propose a strategy for limiting the starting point couple generation.

We define the non-used *segments* in the step x for each stroke in a symbol Sym by $Segs(Sym, x)$. The boundary points of these segments are defined by $FSeg(Sym, x)$. A set of new starting point couples $\{(p_i, p_j)\}$ between two symbols, $Sym1$ and $Sym2$, are produced from $FSeg(Sym1, x)$ to the closest points in $Segs(Sym, x)$, and vice versa :

$$\begin{aligned}
&\{(p_i, p_j)\} = \\
&\{\forall p_i \in FSeg(Sym1, x), \forall seg \in Segs(Sym2, x), (p_i, ppv(p_i, seg))\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \cup \\
&\{\forall p_j \in FSeg(Sym2, x), \forall seg \in Segs(Sym1, x), (ppv(p_j, seg), p_j)\}
\end{aligned} \tag{3.10}$$

Fig. 3.12 shows the possible starting point couples of the first step in Fig. 3.10 which is as considered as the step x. In this case, it exists three starting couples, they are (P1, P6) with only one direction, (P1, P8) with two possible directions and (P5, P11) with only one possible direction. In the general case, up to four directions have to be considered. All the possibilities will be explored by the A* algorithm for searching the best warping path.

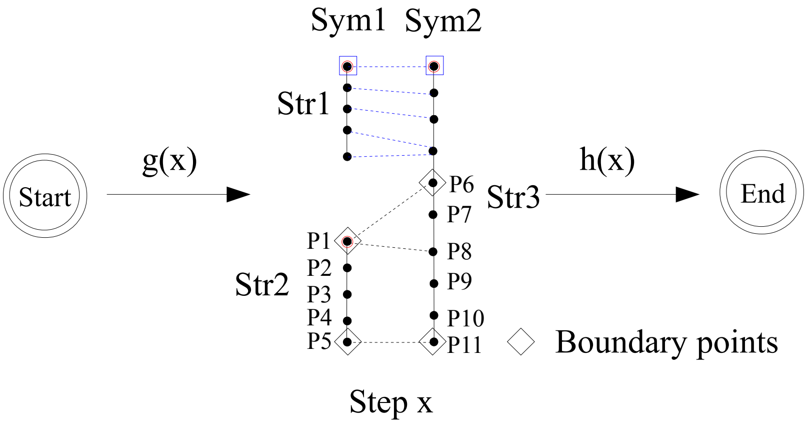


Figure 3.12: Three starting point couples of the first step in Fig. 3.10

Note that the starting point selection (and also for the ending points) leads to two properties of DTW A* :

– Quality of the final solution: if some possibilities are too limited, we cannot arrive at the best solution,

– Running speed: limiting the possibilities and the branches explored from the current step can make the system faster.

The proposed method in this section is named DTW A*. Even we have optimized a lot the A* algorithm in term of time, it is still time-consuming and memory-consuming (storing a large number of hypotheses). More details will be given in the experimental Section 3.7. Beam searching is a possible solution. It is still difficult to use it in practice. In the next section, we will discuss a faster distance, a modified Hausdorff distance, which is similar to the heuristic function of Eq. (3.8).

3.5.3 Modified Hausdorff Distance

The algorithm DTW A* is too slow and too much memory cost in practice. In this section, we will introduce a *modified Hausdorff distance* (MHD) [35, 48, 58] that is faster with a low memory cost between two multi-stroke symbols.

We assume two multi-stroke symbols, $Sym1 = \{p_1(1), ..., p_1(N_1)\}$ and $Sym2 = \{p_2(1), ..., p_2(N_2)\}$. MHD is defined by:

$$\begin{aligned} MHD_{sym}(Sym_1, Sym_2) = \; & \tfrac{1}{N_1+N_2}(subhauf(Sym_1, Sym_2) \\ & +subhauf(Sym_2, Sym_1)) \end{aligned} \tag{3.11}$$

where

$$subhauf(Sym_1, Sym_2) = \sum_{pt_i \in Sym_1} \min_{pt_j \in Sym_2} (dist(pt_i, pt_j)). \tag{3.12}$$

In this thesis, MHD is slightly different from the MHD definition given in [48]. We choose an average distance rather than a maximum distance between two point sets to prevent the effect of outliers. In off-line data (images), Hausdorff distance is used for computing the distance using only x and y coordinates of pixels. In on-line data, we can easily fuse the local direction features and the curvature feature in Section 3.3 for the point-to-point distance $dist(pt_i, pt_j)$. By doing this, MHD can distinguish "□" and "O", which have similar x and y coordinates, but different directions and curvatures for each point.

Using the mentioned distances, we can implement the hierarchical clustering to quantify symbols. In order to evaluate these distances, we will introduce two existing datasets, single-line mathematical expressions and flowcharts in the next section.

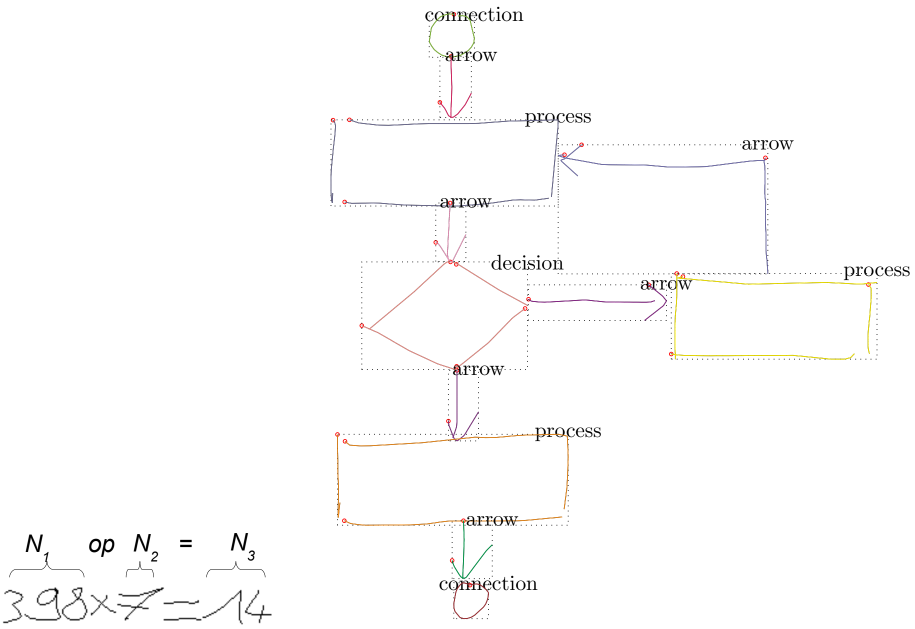
3.6 Existing On-line Graphical Language Datasets

The first simple database is a synthetic handwriting database named *Calc* (Calculate) [59] of realistic handwritten expressions synthesized from isolated symbols. The expressions in *Calc* are produced according to the grammar N_1 *op* $N_2 = N_3$ where N_1, N_2 and N_3 are numbers composed of 1, 2 or 3 real isolated handwritten digits. The distribution of the number of digits for $N_{i=\{1,2,3\}}$ is 70% of 1 digit, 20% of 2 digits and 10% of 3 digits randomly. Furthermore, *op* represents one of the operators $\{+, -, \times, \div\}$. Fig. 3.13a shows an example picked from *Calc* with N_1, N_2, N_3 and *op* containing 3 digits, 1 digit, 2 digits and "×" respectively. Fifteen classes exist in total.

The second handwriting database is a realistic handwritten flowchart database named *FC* database [60]. We use only the six different graphical symbols that represent the basic operations (data, terminator, process, decision, connection, arrows) without any handwritten text, as displayed in Figure 3.13b. It contains six classes.

Tab. 3.2 shows statistical information on the two databases. Each of them is composed of a training part and a test part. Although *Calc* has more symbols than that of *FC*, but *FC* has a larger number of strokes. Furthermore, Fig. 3.14 shows a symbol distribution on different stroke numbers in each symbol. Most of symbols in *Calc* (54.9%) are single-stroke symbols, and 40.2% of symbols in total are two-stroke symbols. By contraries *FC* obviously contains more multi-stroke symbols in proportion. In addition to a high proportion of multi-stroke symbols, *FC* use a more general two-dimension language, flowcharts, rather than single-line mathematical expressions in *Calc*. *FC* therefore is more challenging for unsupervised multi-stroke symbol learning.

In the next section, because the DTW A* algorithm is very slow, only qualitative experiments and a symbol classification task with a limited number of classes will be tested. Classical DTW and MHD will be compared using clustering quality assessment *Purity* ang *NMI* on these two datasets.



(a) (b)

Figure 3.13: Two different handwritten graphical languages: (a) a synthetic expression from *Calc* composed of real isolated symbols, (b) an example of flowchart in *FC* database.

		Symbol Number	Stroke Number	Strokes/Symbol	Class Number	Writer Number
Calc	Training	5472	8185	1.50	15	180
	Test	3035	4547	1.50	15	100
FC	Training	3641	8827	2.42	6	31
	Test	2494	6059	2.43	6	15

Table 3.2: Symbol number and class number on two databases

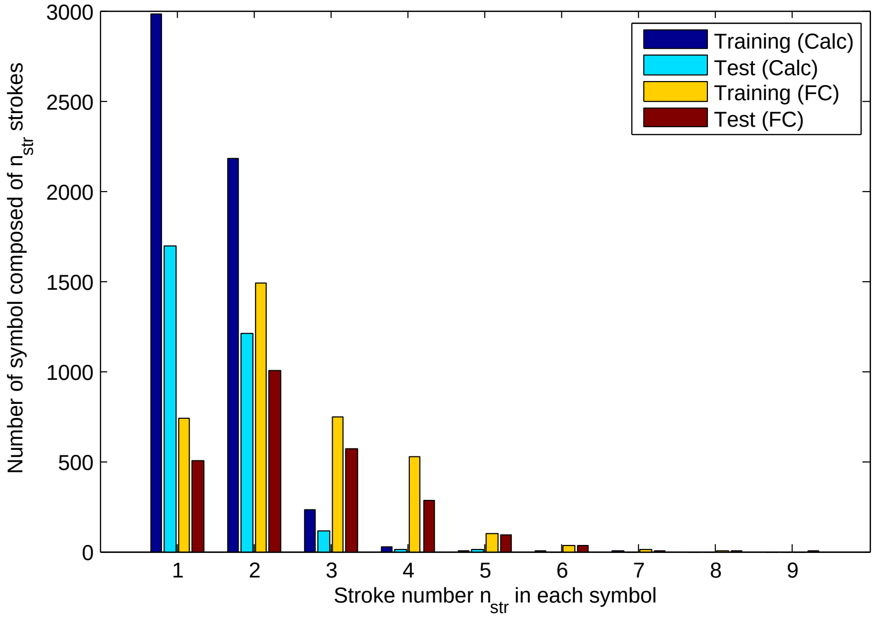


Figure 3.14: Symbol distribution in terms of stroke number in each symbol

3.7 Experiments

In this section, we first qualitatively study the proposed DTW A*, and then the two distances (classical DTW and MHD) are compared using *NMI* and *purity* during clustering on the two different datasets, *Calc* and *FC*.

3.7.1 Qualitative Study of DTW A*

Since the DTW A* algorithm is time-consuming and memory-consuming even when the starting couples of points are limited. In this section, we study a qualitative assessment for a point-to-point matching between two multi-stroke symbols. More precisely, we have selected some representative cases where the computation of DTW A* is feasible and we compare it with the classical DTW. A deeper optimization of the metric in the context of a KNN classifier shows that the metric DTW A* can reach equivalent results as Hausdroff or DTW in a tractable time [61].

Matching Multi-Stroke Symbols

Before matching, all the symbols are normalized into a reference bounding box $\{x \in [-1, 1], y \in [-1, 1]\}$ by keeping the ratio, and re-sampled to a fixed number of 20 points. In order to simply observe the behavior of DTW A*, in this experiment, only two coordinates (x, y) are used for calculating the Euclidean distance between two points; the other features will be used in the later classification application. Fig. 3.15 compares results obtained by the classical DTW (Section 3.5.1) and by our proposed method DTW A* (Section 3.5.2) between two similar symbol shapes, but written with different orders and different directions.

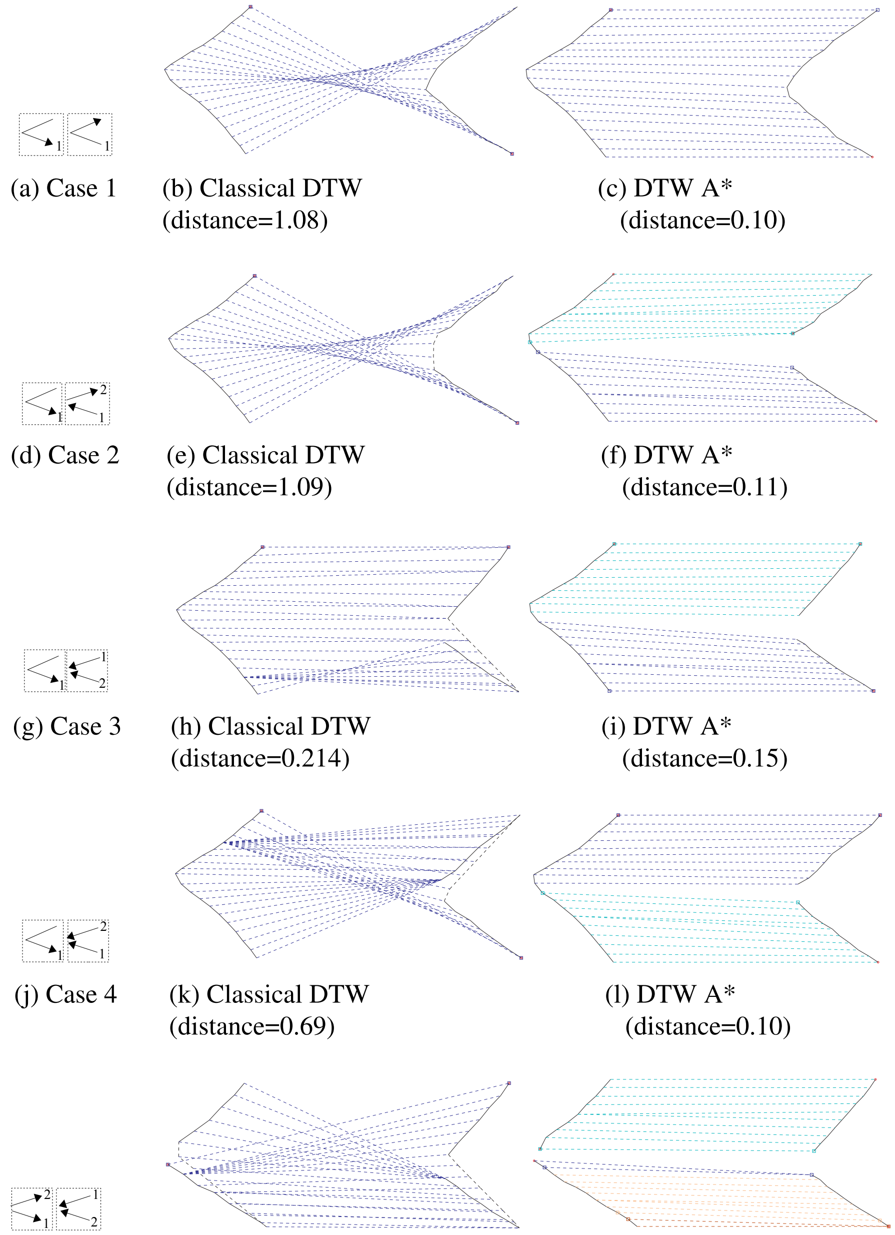
Fig. 3.15a presents an example with two strokes written by two different directions. The distance computed by DTW A*, displayed in Fig. 3.15c, is much smaller than that obtained by the classical DTW, in Fig. 3.15b. The examples presented from the case 2, (Fig. 3.15d), to the case 4 (Fig. 3.15j) compare a symbol "$<$" composed of one stroke, and the same symbol "$<$" written with two strokes in the opposed direction. Our algorithm searches the best warping path among possible different directions and different orders. The last case, Fig. 3.15m, shows a capacity of the system to match two multi-stroke symbols with different written directions and orders. These examples illustrate that DTW A* is independent of written directions and orders of strokes composing the symbols.

A more complex example as shown in Fig. 3.16 compares two allographs of x. Our algorithm found the best solution in 5 steps which are 5 sub-warping paths. The first two steps show the point-to-point matching of the top-left branch of x. The bottom-right branch is matched in the third step, etc. Our algorithm can cut the strokes into sub-graphemes which minimize the distance DTW between segments from two symbols.

Classifying Multi-Stroke Symbols

With the previous experiments we shown that the DTW A* distance was interesting for giving a small distance for similar static patterns but written in different ways. Complementary, it is useful to examine its discriminative power, i.e. its capacity of giving larger distances for pattern of other classes than for those of the same class.

Here again, some selected cases are proposed in Tab. 3.3. In Tab. 3.3, all the examples are re-sampled into 30 points since this number of points are already applied in some classification applications [39, 60]. We can see that the shapes '4' et '8' are correctly found even they have different strokes. The calculating time and the number of hypotheses are shown in Tab. 3.3 and depend on a complexity of



(a) Case 1 (b) Classical DTW (distance=1.08) (c) DTW A* (distance=0.10)

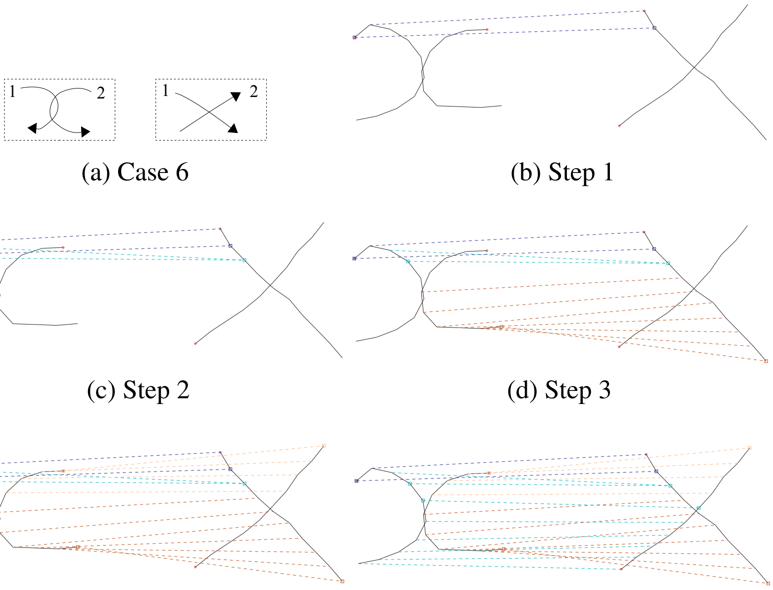
(d) Case 2 (e) Classical DTW (distance=1.09) (f) DTW A* (distance=0.11)

(g) Case 3 (h) Classical DTW (distance=0.214) (i) DTW A* (distance=0.15)

(j) Case 4 (k) Classical DTW (distance=0.69) (l) DTW A* (distance=0.10)

(m) Case 5 (n) Classical DTW (distance=0.55) (o) DTW A* (distance=0.10)

Figure 3.15: Tests on matching two multi-stroke symbols (Classical DTW vs DTW A*)



(a) Case 6

(b) Step 1

(c) Step 2

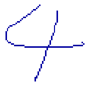
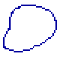
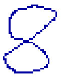
(d) Step 3

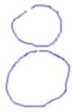
(e) Step 4

(f) Step 5

Figure 3.16: The best solution between two x

shape and specially on the number of resampling points.

	 (2 str)	 (1 str)	(2 str)	 (1 str)

 (2 str)	dist=0.29
	time=127 sec
	77 304 hyp
		dist=0.37
		time<1sec
		3 749 hyp
			dist=0.23
			time=787sec
			238 193 hyp
				dist=0 17
				time<1sec
				218 hyp

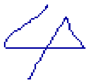
 (1 str)	dist=0.15
	time<1 sec
	112 hyp
		dist=0.44
		time<1 sec
		699 hyp
			dist=0.27
			time=176 sec
			88 820 hyp
				dist=0.37
				time<1sec
				16 hyp

Table 3.3: Classification between symbols (dist=distance, sec=second, str=stroke et hyp=hypothese)

Tab. 3.3 reveals that although DTW A* is good at matching two sequence sets, it is time-consuming and memory-consuming (a large number of hypotheses). Chen's master thesis [61] optimizes this method, and runs a KNN classifier on the *FC* dataset to compare the three distances DTW A*, classical DTW and MHD. The proposed 12-features have been applied in this application. Fig. 3.4 shows the recognition rates. DTW A* is only slightly better (97.47%) than classical DTW (96,79%), and more surprisingly MHD is also very efficient (97.31%). One explanation is that for flowchart the sequence information is very irrelevant, and that it is better not to rely on. To study the sensitivity with respect to the training size set, a 5-fold cross validation is also proposed. We can notice that DTW A* is quite stable (96.90%) although only one fifth of the training samples were available. Conversely, performances of classical DTW drops to 91.93%, in that case missing samples of the training set are not compensated by the flexibility of the matching process.

	DTW A*	DTW	MHD
Normal KNN	97.47%	96.79%	97.31%
Cross-Validation	**96.90%**	**91.93%**	**95.65%**
Decrease	**0.57%**	**4.86%**	**1.66%**

Table 3.4: KNN classification and cross-validation on the dataset *FC* [61].

In the next section, we will assess the clustering quality when comparing classical DTW with multi-stroke concatenation and MHD.

3.7.2 Comparing Multi-Stroke Symbol Distances Using Clustering Assessment

In this section, we will compare two distances, classical DTW with multi-stroke concatenation and MHD on the two training parts of the two datasets, *Calc* and *FC*, mentioned in Section 3.6. The comparison is based on MHI (Eq. (2.23)) in terms of different numbers of clusters (n_p) using the hierarchical clustering. The link metric of *Average* [62] between clusters in the hierarchical clustering is used in this evaluation.

Fig. 3.17 shows the purity using classical DTW and MHD on the training part of *Calc*. At the beginning of learning ($n_p = \{50, 100\}$), the purity of MHD is higher than that of classical DTW. After that, with a larger number of clusters, the purity of MHD is slightly lower (around 1%). Fig. 3.18 shows a similar case using an evaluation of MHD. NMI of MHD on 50 clusters is higher, and then is slightly lower. Clustering with 100 clusters shows a higher *Purity* but a lower NMI.

On the more challenging dataset *FC*, Fig. 3.19 first displays *Purity* on the training part. MHD largely outperforms classical DTW by more than 10% after 100

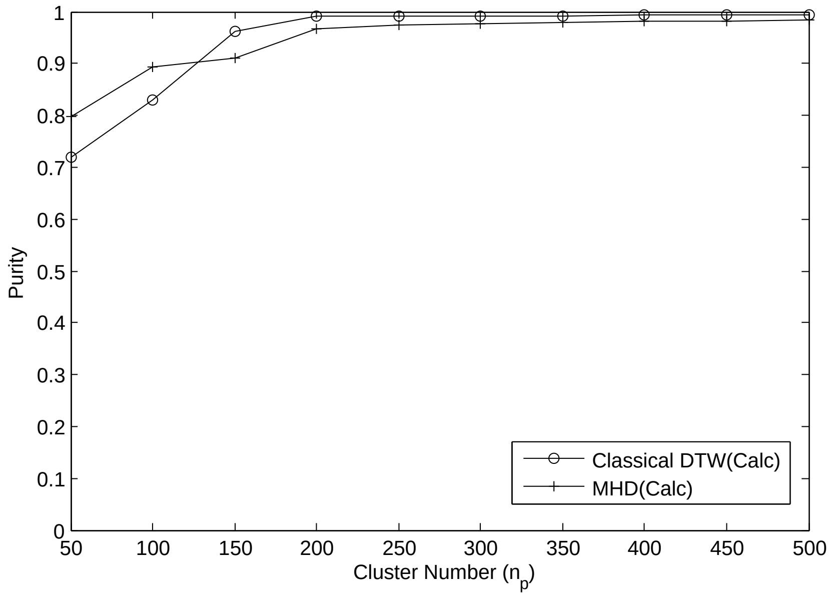


Figure 3.17: Evaluating *Purity* using classical DTW and MHD on the training part of *Calc*

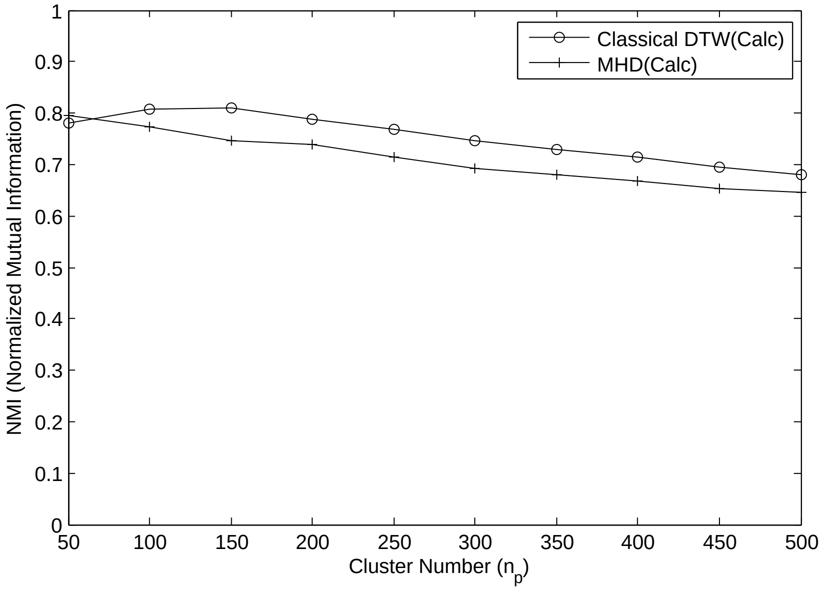


Figure 3.18: Evaluating NMI using classical DTW and MHD on the training part of *Calc*

clusters. A similar situation is illustrated in Fig. 3.20.

On the training part of the simple dataset *Calc*, 54.9% of symbols contain one stroke. On the training part of *FC*, there are 79.7% of more-than-one-stroke symbols. The stroke order in *Calc* is more stable than that of *FC*. With a larger number of clusters, it is possible to get a higher *Purity* on *Calc* using classical DTW. But on the more challenging dataset *FC*, with a complex stroke order problem, classical DTW is defeated. To tackle the complex stroke order problem, we prefer MHD to compare the distance between two multi-stroke symbols in this thesis.

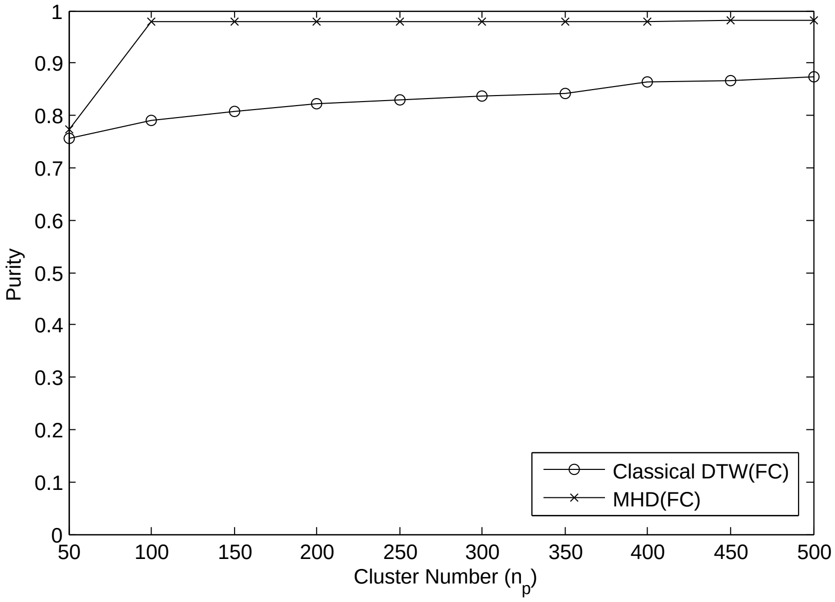


Figure 3.19: Evaluating *Purity* using classical DTW and MHD on the training part of *FC*

3.8 Conclusion

In this chapter, we have discussed how to quantify multi-stroke symbols using a clustering technique. Among different clustering techniques, we have chosen the hierarchical clustering since it allows an easy adaptation to the number of desired clusters. In order to implement the hierarchical clustering, we have studied three distances between multi-stroke symbols.

First of all, we proposed 12 local features of each point for these distances. These features are independent of written direction. Using these features, we have described a classical distance DTW between two single-stroke symbols, and then a

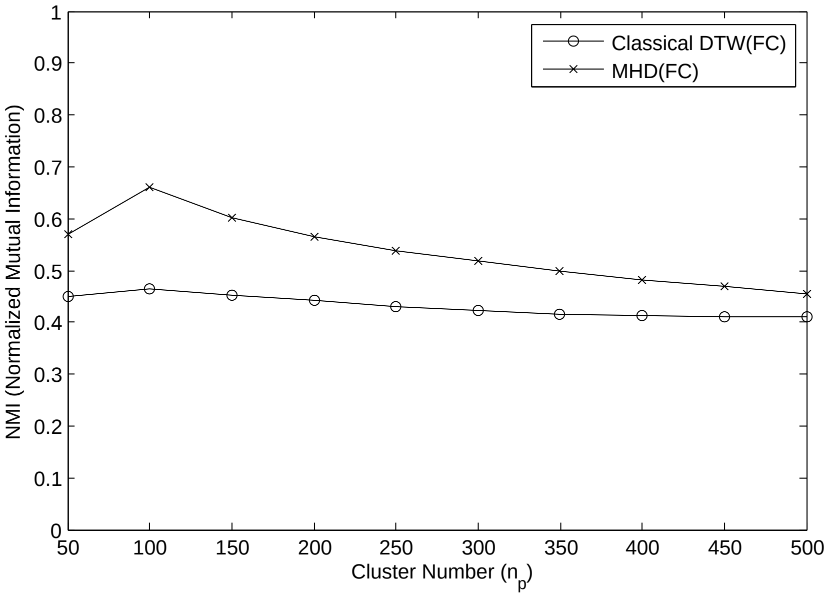


Figure 3.20: Evaluating NMI using classical DTW and MHD on the training part of *FC*

complex stroke-order problem in multi-stroke symbol comparison have been proposed.

An easy solution is when multi-stroke symbols are concatenated into single-stroke symbols by a natural order. The problem of multi-stroke symbol comparison is reduced to the problem of single-stroke symbol comparison. We therefore can use the algorithm DTW for comparing two symbols (classical DTW).

Rather than the natural stroke order concatenation, we proposed a distance DTW A* for searching the minimum sum of point-to-point associated costs between two multi-stroke symbols by a continuity constraint of DTW. To reduce the large number of possible stroke order concatenations, the A* algorithm is used for accelerating the search. Limiting the starting point couples during the matching reduce searching complexity. In the experiment part, we qualitatively study DTW A* by a visual point-to-point matching between two multi-stroke symbols, and then a simple classification application has been proposed to illustrate the discriminant ability of this distance.

Despite the proposed optimizations, DTW A* is too slow to use. We have described another distance MHD for comparing two multi-stroke symbols. In order to compare performance between classical DTW and MHD, we have discussed two clustering criteria, *Purity* and *NMI*. In the experiments, the distance MHD obviously outperforms the distance classical DTW on the more challenging dataset *FC*.

In the next chapter, we will analyze how to extract multi-stroke symbols from relational sequences, a distance between two single-stroke symbols. Since the basic units are single strokes and the evaluated dataset is only the *Calc*dataset, the classical DTW algorithm will be applied.

4

Discovering Graphical Symbols Using the MDL Principle On Relational Sequences

This chapter deals with the problem of symbol knowledge extraction from a mass of handwritten documents. We assume that some unknown symbols are used to compose a handwritten message, and from a dataset of handwritten samples, we would like to recover the symbol set used in the corresponding language. We applied our approach in on-line handwriting, and select a domain of numerical expressions, mixing digits and operators, to test the ability to retrieve the corresponding symbol classes. The proposed method is based on three steps: a quantization of the stroke space, a description of the layout of strokes with a relational graph, and the extraction of an optimal lexicon using the Minimum Description Length (MDL) algorithm.

4.1 Introduction

The knowledge of symbols which compose a given language is something essential to recognize, and then to interpret a handwritten message based on this language. For this reason, most of the existing recognition systems, if not all, need the definition of the character or symbol set, and require a training dataset. The training dataset defines the ground-truth at the symbol level so that a machine learning algorithm can be trained on this task to recognize symbols from handwritten information. Many recognition systems take advantage from the creation of large, realistic corpora of ground-truthed input. Such datasets are valuable for the training, evaluation, and testing stages of the recognition systems. They also allow for comparison between state-of-the-art recognizers. However, collecting all the ink samples and labeling them at the stroke level is a very long and tedious task. Hence, it would be very interesting to be able to assist this process, so that most of the tedious work can be done automatically. Only a high-level supervision need to be defined to conclude the labeling process.

In this respect, we propose to extract automatically the relevant patterns which will define the lexical units of a language. This process is carried out from the redundancy in appearance of basic regular shapes and regular layout of these shapes in a large collection of handwritten scripts.

For the targeted application, which will be defined in more details further and which is related to online numerical expressions, we consider that the strokes are the basic units. Then, these units are composed with some specific composition rules, to produce a symbol of the language, this symbol being an instance of a lexical unit.

The problem is to identify from a large collection of handwritten scripts, all the lexical units based on the observation of the strokes. Some of them corresponding directly to a symbol, others are only a part of a symbol. Eventually, the same kind of stroke according to the context will be either a single symbol or a piece of a more complex symbol.

Let us illustrate the concept with a simplified example. Assume that we observe only two kinds of strokes: horizontal strokes and vertical strokes {'$-$', '$|$'}. In a first model, consider also, that the only composition rule is the left to *Right* rule (R). Then, it is possible to produce this kind of string : "$| - | | | - | | | - | | |$

” . Based on all the available strings in the training dataset, we would like to be able to define a lexicon, i.e. a list of lexical units, which will allow to describe in an optimal way the entire corpus. With this example, two possible lexicons would be L = {“| − |”, “|”} or L = {“| − | |”, “|”}. A similar problem is studied in unsupervised language acquisition [11]; the lexicon is extracted from texts, considered a sequence of characters.

Suppose now, that we add two new composition rules: the $Below$ rule (B) and the $Intersection$ rule (I). Then, with these two-dimensional spatial relations in addition to sequences of strokes, we are able to compose more complex messages, such as “$| + | = ||$”. In this case, the search space for the combination of strokes which forms possible symbols is much more complex since it is no longer a linear one.

We give an overview of the proposed system in Section 4.2, then the extraction of the graphemes and of their spatial relationships are presented in Section 4.3. We describe the algorithm which is used to build the lexicon in Section 4.4, before presenting the experimental results in Section 4.5.

4.2 Overview

This section introduces an overview of the extraction of the lexicon, which will be described deeper in next sections. Given a handwriting database, Fig. 4.1 describes the proposed scheme to extract the lexicon of symbols. This scheme contains three principal steps, quantization of strokes, building relational graphs between strokes and converting them into relational sequences, and lexicon extraction.

First, we code each stroke using a finite set of graphemes, called *codebook*. This is the quantization step, which will rely on the algorithms presented in Chapter 3. The second step, the extraction of spatial relations, analyzes spatial relations between the strokes. For instance, “$=$” is composed of the two same graphemes “$-$” with the spatial relation below “B” which corresponds to the following subsequence $(-, B, -)$.

These graphemes and spatial relations are organized in a relational graph. From the relational graph, the sequences containing graphemes and spatial relations are extracted, they are next processed by the third step, which computes the lexicon.

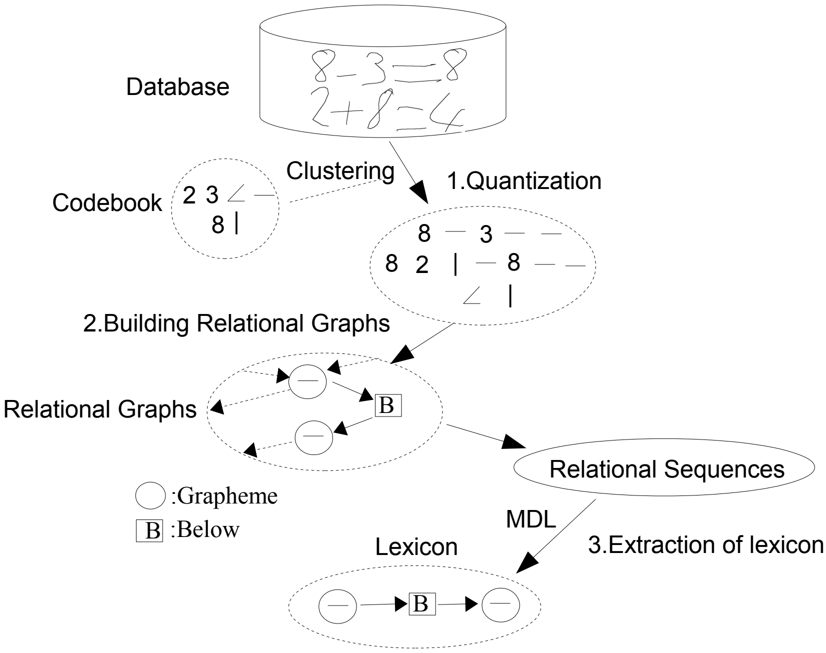


Figure 4.1: Lexicon extraction overview

The main idea of the developed algorithm is to use the frequency of sub-sequence of graphemes/relations to detect symbols. If the subsequence $(-, B, -)$ is very frequent, we could consider $(-, B, -)$ as a lexical unit. In fact, a lexical unit usually represents a symbol.

The focus in this chapter is mainly on the extraction of the lexicon from the relational sequences, which are produced from relational graphs. We start by presenting briefly the construction of the relational graph.

4.3 Extraction of Graphemes and Relational Graph Construction

For the quantization of strokes, we firstly extract the graphemes. Since the DTW distance as shown in Section 3.7.2 slightly outperforms the MHD distance on the *Calc* dataset, the DTW distance will be used to calculate the shape similarity between two strokes. Clustering techniques are used for the generation of the codebook. Many different algorithms are available for this task. As described in the previous chapter, we select an agglomerative hierarchical clustering based on the Lance-William formula [42] for its computation efficiency. Once the graphemes (prototypes) are selected with the hierarchical clustering on the training part, all the strokes on the test part and on the training part are tagged with the virtual label of the closest grapheme using the DTW distance.

The second step extracts the spatial relations between the strokes. To start with a limited complexity example, we will predefine three generic spatial relations, right (R), below (B) and intersection (I) and test the method on the *Calc* dataset. Fig. 4.2 illustrates an example of the predefined *Right* relation of the stroke "2" in an expression "$2 + 8 = 4$". The right projection of "2" contains "$+8 = 4$".

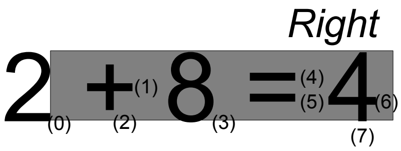


Figure 4.2: From a reference stroke "2", *Right* spatial relation is defined using the projection on the right side.

Because of the orientation of these three relations, we choose the top-left stroke as the first stroke to start building a Directed Acyclic Graph (DAG) in this chapter. To build the DAG, we have considered from each node (stroke) the outcome of at most two possible edges when relation B and/or R are satisfied and only one edge when relation I (with a higher priority) is encountered. In other words, I is exclusive with R and B. The edges are oriented towards the nearest strokes for the considered relation.

We don't predefine *left* and *above* spatial relations. Thus, no loop in the graph will be created. It is a breadth-first-like graph building. In this way, we obtain a DAG. All the nodes in the DAG can be traveled by several possible paths (sequences). Thus, given a handwriting database containing expressions $\{e_i\}$, it is transformed into a set of sequences of graphemes/relations $\{sq_j\}$. This transformation bridges the gap between the graph and descriptions as sequences.

An example of the DAG construction from an expression is presented in Fig. 4.3. The graphemes are marked by the indices of strokes in equation to avoid any ambiguity since several strokes share the same grapheme. In this example, two paths (sequences) are necessary to traveled through all the nodes in the graph : ($2_{(0)}$, R, ..., $-_{(4)}$, R, $\angle_{(6)}$, I, $|_{(7)}$) and ($2_{(0)}$, R, ..., $-_{(4)}$, B, $-_{(5)}$, R, $\angle_{(6)}$, I, $|_{(7)}$). In the next section, we explain how to extract the lexicon from these sequences.

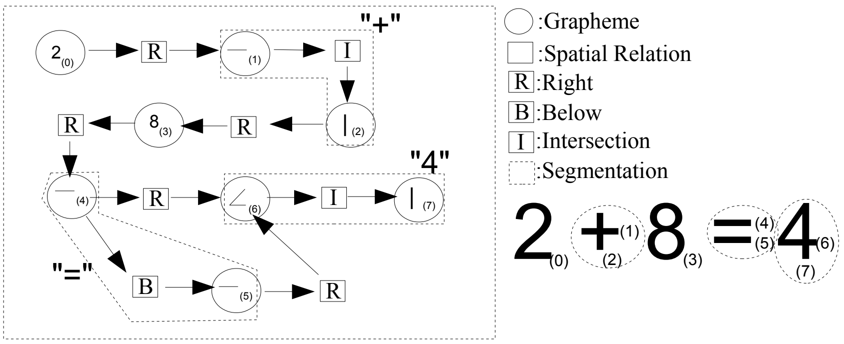


Figure 4.3: Example of the relational graph of the expression “$2 + 8 = 4$”

4.4 Extraction and Utilization of the Lexicon

We use the iterative algorithm proposed in [20] to build the lexicon from the sequence of graphemes/relations. The principal idea of this algorithm is to minimize the description length of sequences by iteratively trying to add and delete a word, in terms of Rissanen’s Minimum Description Length (MDL) principle [18]. The MDL principle means that the best lexicon minimizes the description length of the lexicon and that of the observation; in our case the observation is the sequence of graphemes/relations, this is a big difference with Marcken’s work [20] where electronic texts are processed.

A simple example was shown in Section 2.1.1 to understand this process. In the end, we get an optimal lexicon L on the training handwriting database containing the sequences of graphemes/relations $\{sq_j\}$.

4.4.1 Segmentation Using Optimal Lexicon

Here we explain how to obtain a segmentation of a new expression using the computed optimal lexicon L. A new expression e is transformed into one or more sequences $\{sq_k\}$ of graphemes/relations from the relational graph, $e \rightarrow \{sq_k\}$. We get the segmentation from the sequences $\{sq_k\}$ by the Viterbi representation with the optimal lexicon L. In fact, the segmentation from $\{sq_k\}$ contains the spatial relations, I, R, and B. These spatial relations are used to define the lexicon at the symbol level. For instance, the sequence $(-, B, -)$ which defines the symbol “$=$” is different from the sequence $(-, I, -)$ which corresponds to the “$+$” symbol. The segmentation of the complete expression is the result of merging segmentation from each path.

To illustrate this segmentation step, let us consider a lexicon with the following 15 elements:

$$\begin{aligned}&\{(R),(B),(I),(2),(-),(|),(8),(\angle),\\&(2,R),(-,I,|),(R,8,R,-,R),(\angle,I,|),\\&(R,8,R),(-,B,-),(\angle,I,|)\}.\end{aligned}$$

The two paths extracted from the graph given in Fig. 4.3 can be coded using this lexicon according to:

$$\{(2_{(0)},R),(-_{(1)},I,|_{(2)}),(R,8_{(3)},R,-_{(4)},R),(\angle_{(6)},I,|_{(7)})\}$$

and

$$\{(2_{(0)},R),(-_{(1)},I,|_{(2)}),(R,8_{(3)},R),(-_{(4)},B,-_{(5)}),(R),(\angle_{(6)},I,|_{(7)}).$$

Then, it is possible to remove the spatial relations from these sequences, they do not hold any information regarding the segmentation, so that to keep only the graphemes and their grouping. Thus, the following two sets of segments are produced:

$$\{\{2_{(0)}\},\{-_{(1)},|_{(2)}\},\{8_{(3)},-_{(4)}\},\{\angle_{(6)},|_{(7)}\}\}$$

and

$$\{\{2_{(0)}\},\{-_{(1)},|_{(2)}\},\{8_{(3)}\},\{-_{(4)},-_{(5)}\},\{\angle_{(6)},|_{(7)}\}.$$

The union of these two sets is the segmentation of the equation, { $\{2_{(0)}\}$, $\{-_{(1)},|_{(2)}\}$, $\{8_{(3)},-_{(4)}\}$, $\{8_{(3)}\}$, $\{-_{(4)},-_{(5)}\}$, $\{\angle_{(6)},|_{(7)}\}$ }. We define this union of segmentations as $s(e,L)$ given a lexicon L.

However, there may be two members in $s(e,L)$ which are intersected, but not self-included which means conflict. For example, the two sets of $\{8_{(3)},-_{(4)}\}$ and $\{-_{(4)},-_{(5)}\}$ are in conflict due to the presence of the stroke $-_{(4)}$ in both sets: $CB(\{8_{(3)},-_{(4)}\},\{-_{(4)},-_{(5)}\})$. We call the conflict as CB since the brackets are crossing. We define the function $C(.)$ as the number of occurrences for a sequence in the training dataset. We solve this conflict by tracing back to $C((8_{(3)}, R, -_{(4)}))$ and $C((-_{(4)}, B, -_{(5)}))$ in lexicon and by keeping the bigger $C(.)$ in sequences of graphemes/relations; the other set is deleted. Probably we will keep $\{-_{(4)}, -_{(5)}\}$ since $C((-_{(4)}, B, -_{(5)})) > C((8_{(3)}, R, -_{(4)}))$. Therefore, we get the segmentation for the equation in Fig. 4.3, { $\{2_{(0)}\}$, $\{-_{(1)}, |_{(2)}\}$, $\{8_{(3)}\}$, $\{-_{(4)}, -_{(5)}\}$, $\{\angle_{(6)}, |_{(7)}\}$ }.

We keep the single strokes in the symbol segmentation since some symbols have a hierarchical meaning. For instance, the symbol "±" are composed of "+" and "−". In the future work, it may be useful. We build the segmentation as a hierarchical structure from the hierarchical lexicon L. Finally, we get the segmentation defined by $S(e, L) = \{\{2_{(0)}\}$, $\{-_{(1)}\}$, $\{|_{(2)}\}$, $\{8_{(3)}\}$, $\{-_{(4)}\}$, $\{-_{(5)}\}$, $\{\angle_{(6)}\}$, $\{|_{(7)}\}$, $\{-_{(1)}, |_{(2)}\}$, $\{-_{(4)}, -_{(5)}\}$, $\{\angle_{(6)}, |_{(7)}\}\}$. This hierarchical structure provides us grammar information, i.e. $\{\{\angle_{(6)}, |_{(7)}\}\} \to \{\{\angle_{(6)}\}, \{|_{(7)}\}\}$.

In a word, given a handwriting database $\{e_i\}$, we transformed $\{e_i\}$ to sequences of graphemes/relations $\{sq_j\}$ and then extracted an optimal lexicon L from these sequences. Considering a new expression e, we obtained the segmentation $S(e, L)$ in terms of the lexicon L. At the end we clean the conflict in segmentation and made the segmentation as the hierarchical structure $S(e, L)$.

4.4.2 Segmentation Measures

Our objective is to verify if our extracted segmentation corresponds to a human made segmentation $S(e, G)$ (ground-truths). We evaluate the performance of segmentation with lexicon L by four measures from [11]. The first is the recall rate:

$$R_{\text{Recall}} = \frac{|S(e, G) \cap S(e, L)|}{|S(e, G)|}, \tag{4.1}$$

where $|.|$ is the cardinality of a set. The recall rate evaluates the percentage of right segmentation which are found in ground-truths. On the contrary the second measure R_{CB} calculates the percentage of crossing brackets in $S(e, G)$ compared with the $S(e, L)$. R_{CB} is defined by:

$$R_{CB} = \frac{|\{A | A \in S(e, G), \exists B \in S(e, L), CB(A, B)\}|}{|S(e, G)|}. \tag{4.2}$$

R_{CB} reveals the errors of the segmentation of L compared with ground-truths.

The third measure is defined by $R_{Lost} = 1 - R_{Recall} - R_{CB}$. R_{Lost} means the percentage of segments (symbols) in $S(e, G)$ which are not found. Note that these three measures are justified because of the hierarchical structure of the resulting segmentation. In addition, we define a fourth measure based on the segmentation found by the Viterbi representation using the longest words:

$$R_{Top} = \frac{|\{A|A \in S(e,G), \exists B \in Top(S(e,L)), B = A\}|}{|S(e,G)|} \tag{4.3}$$

where $Top(S) = \{D|D \in S, \forall E \in S, E \neq D, D \not\subset E\}$ which extracts a set of the longest possible segments without inclusion. Thus, R_{Top} evaluates the performance of Viterbi representation.

To explain the proposed measures, we use, as illustrated in Fig. 4.4, a lexicon L_m giving the hierarchical structure segmentation $S(e, L_m) = \{\ \{|_{(1)}\}, \{-_{(2)}\}, \{|_{(3)}\}, \{-_{(4)}\}, \{2_{(5)}\}, \{-_{(6)}\}, \{-_{(7)}\}, \{\angle_{(8)}\}, \{|_{(9)}\}, \{-_{(2)}, |_{(3)}\}, \{-_{(2)}, |_{(3)}, -_{(4)}\}, \{2_{(5)}, -_{(6)}\}\}$. The ground-truth is defined by $S(e, G) = \{\{|_{(1)}\}, \{-_{(2)}, |_{(3)}, -_{(4)}\}, \{2_{(5)}\}, \{-_{(6)}, -_{(7)}\}, \{\angle_{(8)}, |_{(9)}\}\}$. In this case, the recall rate is that $R_{\text{Recall}} = 3/5 = 0.6$; since three symbols of the ground-truths $\{|_{(1)}\}$, $\{-_{(2)}, |_{(3)}, -_{(4)}\}$ and $\{2_{(5)}\}$ are found. The crossing bracket rate R_{CB} is $1/5 = 0.2$ as the ground-truth $\{-_{(6)}, -_{(7)}\}$ is crossing with the segmentation $\{2_{(5)}, -_{(6)}\}$. R_{Lost} is 0.2 since the ground-truth $\{\angle_{(8)}, |_{(9)}\}$ is lost. R_{Top} is $2/5 = 0.4$ because $\{|_{(1)}\}$ and $\{-_{(2)}, |_{(3)}, -_{(4)}\}$ are found by the longest of hierarchical segmentation.

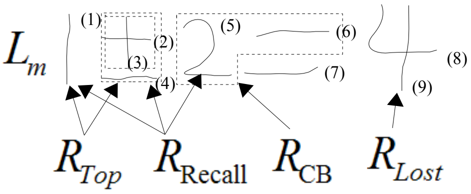


Figure 4.4: Hierarchical segmentation of a lexicon evaluated by four measures

Measures at Multi-Stroke Symbol Level

A dataset may contain many single-stroke symbols. Since we are using a hierarchical symbol segmentation. All the single-stroke symbols are found in the recall rate. So the R_{Recall} advantages the dataset with lot of single stroke symbols. That is why we introduce a Multi-stroke Recall rate :

$$R_{\text{MRecall}} = \frac{|S(e, G_M) \cap S(e, L)|}{|S(e, G)|} \tag{4.4}$$

where $S(e, G_M)$ represents ground-truth segments which contain more than two strokes, $S(e, G_M) = \{g | g \in S(e, G), |g| \geq 2\}$. $R_{\text{MRecall}} = 1/3 = 0.333$ in Fig. 4.4 where only $\pm$ is found among three symbols $\pm$, $=$, and 4.

4.5 Experiment Results and Discussion

On the training part of *Calc* presented in Section 3.6, we firstly extract all the prototypes of strokes (graphemes). In the proposed approach, the main parameter is the number of prototypes in the grapheme clustering and the number of iteration (i.e. symbols) in the MDL algorithm. Fig. 3.18 and Fig. 3.17 show that the DTW distance works better than the MHD distance using 150 prototypes. To illustrate the full chain, we first set the number of prototype to 150 and study the effect of the number of iterations. Then we search for the optimal number of prototypes. Finally the optimal configuration is tested on the test dataset.

Using 150 prototypes, we try iteratively to add and delete a word with the lexicon in terms of the algorithm in [20]. The lexicon starts with 153 words. In our case, a word is in fact at the beginning a prototype (150) or a spatial relation (3). Fig. 4.5 shows the accuracy of segmentation for the four measures, R_{Recall}, R_{Top}, R_{CB} and R_{Lost} during the unsupervised learning of the lexicon on the training part of the database. The best accuracy R_{Top} of 65% is reported in the 20th iteration and then it decreases slowly. The convergence of the algorithm is obtained at iteration number 498. No more decrease in the description length is possible, and the number of words is then 504. According to R_{CB} and the overlap of R_{Top} and R_{Recall}, our system does not make any error before the 20th iteration, but it leaves 35% of symbols lost in terms of R_{Lost}. After 20th iteration, R_{Recall} keeps increasing until the end of iteration, but R_{Top} decreases gradually. It means that the new added word does not represent symbols, but frequent sequences of symbols or of parts of symbol. At the end of iteration R_{Recall}, R_{CB}, R_{Lost} and R_{Top} are 77%, 10%, 13% and 61% respectively.

To find the optimal number of prototypes of strokes, Fig. 4.6 shows the rates for different numbers of prototypes at the end of the lexicon extraction; no more word

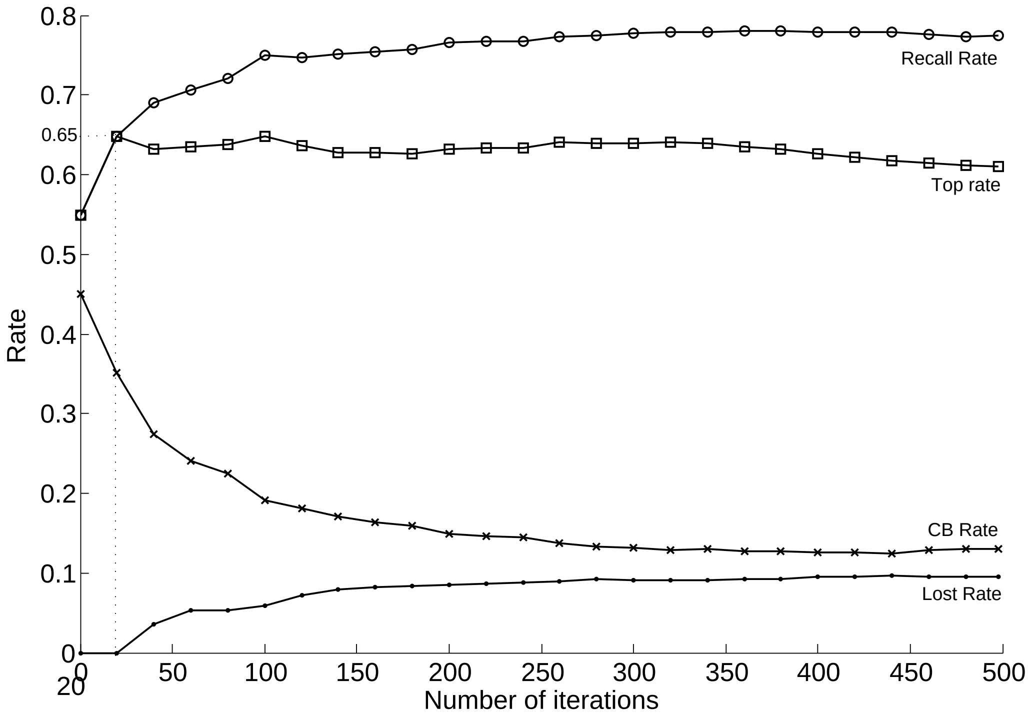


Figure 4.5: Accuracy of segmentation

can be added and deleted. The best R_{Recall} of 78% ($R_{\text{MRecall}} = 51.2\%$) is reported using 120 prototypes ($R_{CB} = 10\%$, $R_{Lost} = 12\%$, $R_{Top} = 62\%$). R_{CB} always decreases since more and more different prototypes of strokes are found. R_{Top} remains fluctuating roughly around 60% after 75 prototypes. The best compromise in term of the number of prototypes is 120 because of the high R_{Recall} and R_{Top} and the low R_{CB} and R_{Lost}.

Next, we test the learned lexicon using 120 prototypes of strokes on the test part of *Calc*. We obtain the following results: R_{Recall} of 74% ($R_{\text{MRecall}} = 41.2\%$), R_{CB} of 10%, R_{Lost} of 16%, R_{Top} of 63%. R_{Top} is comparable to the one computed on the learning part. These results show the robustness of our lexicon. In the field of unsupervised learning on texts, the similar problem of segmentation is studied a lot. Thus, we compare the recall rate in handwritten expressions with that in texts. The alphabet (a, b, c,..., z) size is fixed to 26 while our grapheme number is unknown. Learning a lexicon in texts is more stable. Using the same learning method of description length in [20], R_{Recall} of 90.5% and R_{CB} of 1.7% are reported respectively on an English corpus, Brown corpus [21]. Although our R_{CB} of 10% is much higher than that in texts, but we get a fair R_{Recall} of 74%.

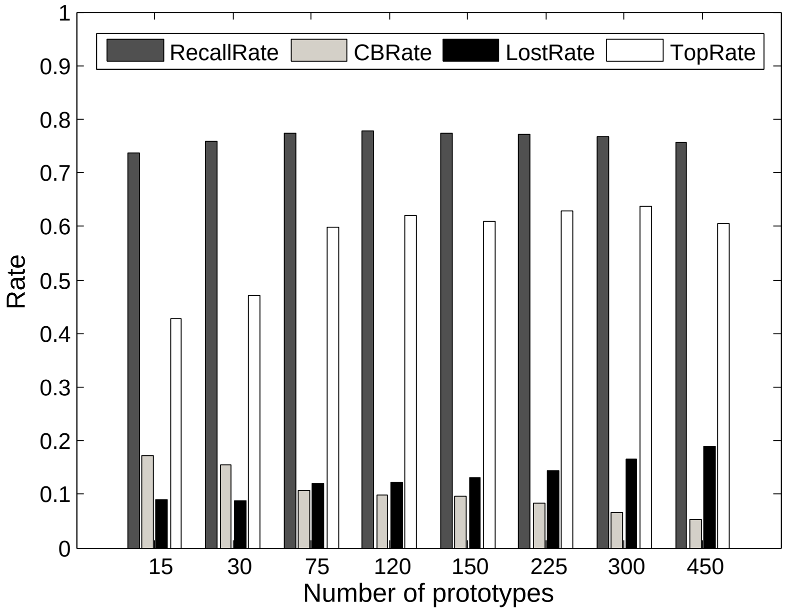


Figure 4.6: Rates for different numbers of prototypes on the training part

4.6 Conclusion

In this chapter we presented an unsupervised learning method of lexicon on simple handwritten mathematical expressions. Firstly, the graphemes are extracted by agglomerative hierarchical clustering. Secondly, the graphs of spatial relation are generated inspired by SRT, and these graphs are transformed into sequences. We extracted the lexicon from these sequences by reducing the description length.

At the end, we got a recall rate of 74% ($R_{\text{MRecall}} = 41.2\%$) on the test part of our database *Calc*. The grammar of *Calc* is very simple and only one-line expressions exist. Therefore, if we increase the complexity of handwriting database, we may need some new spatial relations. Then, the relational graph becomes more complicated and we will need some graph mining techniques as for example in [13]. This is the issue which will be addressed in the next chapter with the work on the flowchart language.

5

Discovering Graphical Symbols Using the MDL Principle On Relational Graphs

In the previous chapter, we have embedded the spatial relations into a string, i.e. a sequence of symbols, to be able to derive the presence of composed graphical symbols using the MDL principle. However, the spatial relations were limited to some predefined spatial relations and the structure is limited to linear sequences. In this chapter, we will extend the predefined spatial relations to unsupervised spatial relations, and extend the relational sequences to relational graphs for being able to process more general graphical languages. Unsupervised spatial relations are produced by a clustering technique. In order to implement this clustering, we extract spatial relation features at three levels [25]: topological relations, orientation relations, and distance relations. The spatial relations are embedded into a feature space of a fixed-length dimension, and then we apply the *k-means* clustering algorithm to generate spatial relation prototypes. Therefore, more typical spatial relations of the considered language will be used to build the relational graph. Using the MDL principle directly on the relational graphs, we can extract relevant sub-graphs as the graphical symbols.

5.1 Introduction

Handwritten scripts derive from a graphical language. In other words, a language, which uses a set of rules, generates some handwritten scripts. A language could be Context Free Grammar (CFG) generator [63], but certainly the human natural language is more complex than CFG [64]. On textual corpora, which can be considered as a subform of simple graphical languages, unsupervised learning of CFG has been discussed and considered as a non-trivial task [17, 65, 66].

In the previous chapter, we proposed to convert the graphical language into sequences, so that the MDL principle on texts could be applied. We have applied this method on the single-line mathematical expressions, but not on the more more complex flowcharts yet. In this chapter, we propose to extend this kind of approach on real graphical languages where not only left to right layouts have to be considered.

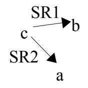
In this case, the search space for the combination of units which makes up possible lexical units is much more complex since it is no longer a linear one as in texts. We can describe these two-dimensional spatial relations with a graph. The problem is to search the repetitive sub-graphs in the graphs. Note that searching a repetitive graph in graphs is different from searching a repetitive sub-graph in graphs. We can embed the graphs into a feature space [67] so that the graphs can be indexed. The graph searching can be very fast. Sub-graph searching requires that all the sub-graphs should be derived from graphs beforehand. However, the sub-graph generation would be time-consuming and memory-consuming. Imagining a graph containing 10 nodes, there are $C_{10}^{1}+C_{10}^{2}+C_{10}^{3}+...+C_{10}^{9}+C_{10}^{10}=1023$ sub-graphs where $C_n^m = \frac{n(n-1)...(n-m+1)}{m(m-1)...2\cdot 1}$ [68], and C_{10}^{i} means how many ways we choose i nodes from 10 nodes. Thus, a sub-graph mining technique is required to extract the repetitive patterns in the graphs. Such a task is performed with the SUBDUE (SUBstructure Discovery Using Examples) system using a beam searching [23]. It is a sub-graph based knowledge discovery system which extracts substructures in the graphs using the MDL principle.

This chapter proposes a solution to model the two-dimensional graphical language with a graph. We first present a system overview in Section 5.2, and then the relation graph construction of graphical language will be presented in Section 5.3. At the end, we will discover the repetitive sub-graphs as the multi-stroke symbols using the MDL principle.

5.2 System Overview

In this section, we give an overview of the proposed method for extracting graphical symbols (lexicon) from a handwriting corpus as shown in Fig. 5.1. We use four principal steps: i) building the relational graphs using neighbor strokes, ii) quantization of strokes into grapheme prototypes as presented in Chapter 3, iii) quantization of spatial relations, and iv) extraction of the lexicon composed of graphemes and spatial relations which will be presented in this chapter. Compared with the previous chapter, the spatial relations are unsupervised and the MDL principle is applied in graphs without converting them into relational sequences.

As shown in Fig. 5.1, given a new graphical document, we first build the relational graphs using the n_{cstr} closest strokes without labeling nodes and edges. In order to give labels for the edges and the nodes, the stroke quantization and the spatial relation quantization will be done as shown in Fig. 5.1. Thus, we will obtain a quantified relational graph. The MDL principle will be applied directly on the relational graphs to search sub-graphs as the graphical symbols.

As an example in Figure 5.1, we assume a handwriting database containing a flowchart "□ → □ → □". Each stroke is marked by the index (.) to avoid ambiguity. We first build the relational graph using the closest neighbor strokes. To quantify the nodes, we will use an unsupervised clustering algorithm to obtain the following set of graphemes {a'−', b'¬', c'|', d'\', e'/'} defining the codebook. Then, we code each stroke (node) using this codebook. Afterward, we quantify the spatial relations (edges), which will be discussed later. We assume that two spatial relations $SR1$ and $SR2$ have been found. As a simple example, the frequent sub-graphs () could be discovered from the relational graph using the MDL principle. We can consider this sub-graph as a graphical symbol.

We have presented the stroke quantization in Chapter 3. In this chapter, we will use the hierarchical clustering based on the MHD distance since the MHD distance

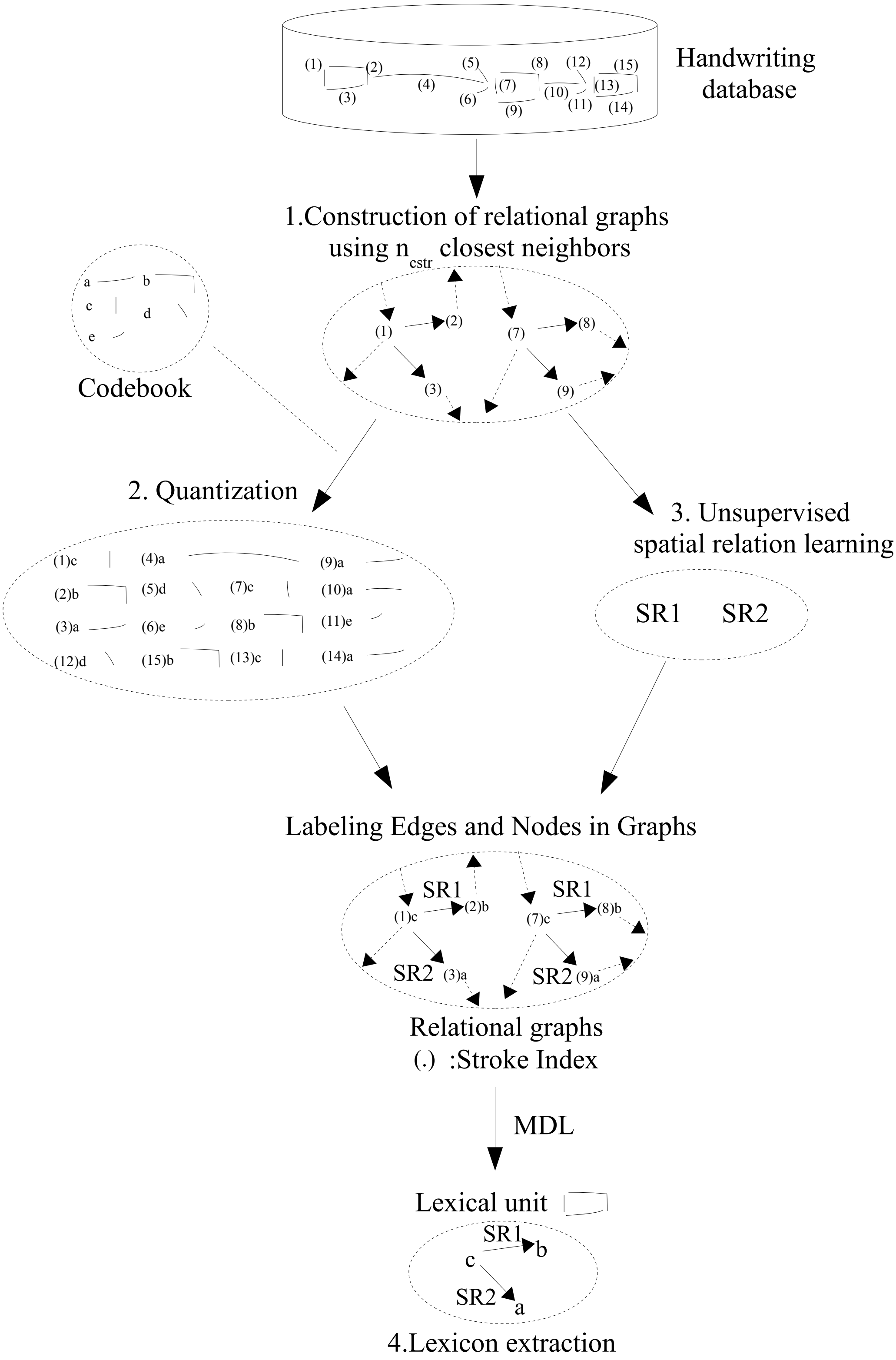


Figure 5.1: Overview for unsupervised graphical symbol learning

outperforms the DTW distance on the *FC* dataset as studied in Chapter 3. The *FC* dataset will be one of evaluated datasets. In this chapter, we mainly focus on the problem of building relational graphs between strokes, and then extracting multi-stroke symbols (a lexicon) from these graphs. In the next section, we will introduce the relational graph building, which is different from the method as presented in Section 4.3.

5.3 Relational Graph Construction

In this section, we will build the relational graphs between strokes, and then edges in the relational graphs will be quantified. In the next subsection, we will normalize spatial composition to build the relational graphs.

5.3.1 Spatial Composition Normalization

First of all, given some graphical evidences (a set of strokes $\{str_i\}$), a strokes pre-processing is required. Since the strokes may be collected by different input devices or written by different individuals, a same layout may be found at different scales. We define a graphical sentence as a layout organized by a set of strokes where this set contains only full graphical symbols, not a part of symbol. Each graphical sentence is independent of the other graphical sentences in terms of the spatial relations, but all the graphical sentences use the same graphical symbol set. For example, a set of graphical sentences can be a set of pages that are homogeneous, but independent. Fig. 5.2 shows two graphical sentences produced by a handwritten flowchart language. Two graphical sentences are independent in terms of the spatial relations. All the strokes in a graphical sentence have to be normalized by local average diagonal size of stroke bounding boxes. We analyze the spatial relations on this normalized layout. Considering the shape of strokes, we use the hierarchical clustering in Chapter 3 to regroup the strokes (single-stroke symbols) into a finite set of n_p graphemes, named *codebook*, and the strokes are labeled with the nearest grapheme [33]. This step is the quantization of strokes (single-stroke symbols). In the generation of codebook, we do not consider the size of strokes, but only the shape of strokes is considered. In the next sections, we first construct the relational graph between strokes, and then quantify spatial relations (edges in the relational graph) to discover multi-stroke graphical symbols.

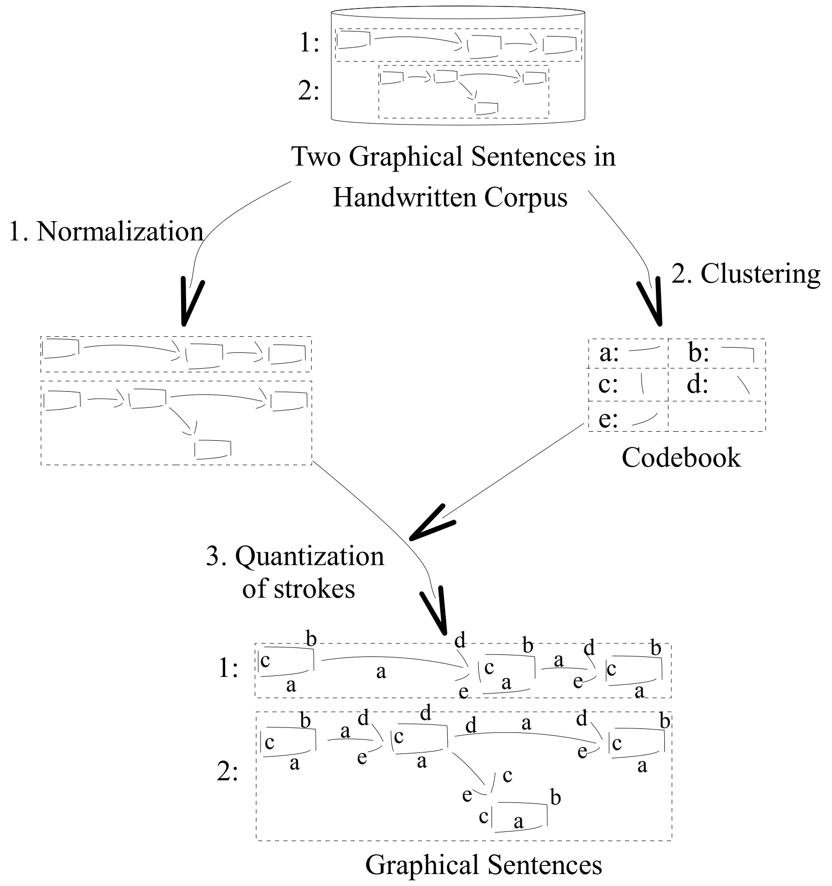


Figure 5.2: Stroke Pre-processing

5.3.2 Constructing a Relational Graph using Closest Neighbors

In this section, we will build a relation graph between strokes for a graphical sentence, and then the spatial relations (edges) can be quantified.

Given a graphical sentence, we describe the layout of strokes with a relational graph. We consider nodes as the strokes and edges as the spatial relations, as shown in Fig. 5.3. A spatial relation is defined from a reference stroke to an argument stroke. In other words, an edge is directed. This allows for instance to distinguish between two different layouts, such as "$-\ >$" and "$>\ -$". Concerning the complexity, suppose we have n_{str} different strokes, to create a complete directed graph for all the nodes (strokes), the number of directed edges is

$$2 \cdot C^2_{n_{str}} = n_{str}(n_{str} - 1). \tag{5.1}$$

In that case, the search space would be far more too complex to search patterns in the complete directed graph. Therefore, the number of out-directed edges from a reference stroke should be limited to n_{cstr} closest strokes where $n_{cstr} \leqslant n_{str} - 1$ since we, human, have a limited perceived visual angle [69]; we prefer some symbols composed of the closest strokes. Consequently, strokes which are far away from the reference stroke are not necessary to be linked with the reference stroke. The reduced number of directed edges is:

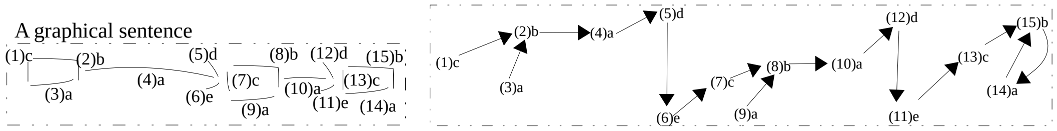
$$n_{str} \cdot n_{cstr}. \tag{5.2}$$

However, if n_{cstr} is too small, we could lose some symbols. A spatial distance $dist_{sp}$ for the closest strokes is defined by the Euclidean distance between two closest points in the two sequences of points (two strokes) respectively. Formally, suppose that we have two strokes, $str_x = (..., p_x(i), ...)$ and $str_y = (..., p_y(j), ...)$ where $p_x(i)$ and $p_y(j)$ are the points of two strokes, we define the distance between two strokes as:

$$dist_{sp}(str_x, str_y) = \min_{p_x(i) \in str_x, p_y(j) \in str_y} dist(p_x(i), p_y(j), 'x, y') \tag{5.3}$$

where $dist(p_x(i), p_y(j), 'x, y')$ is the Euclidean distance between two points using only x and y features in Section 3.3. Considering a reference stroke str_{ref}, we can find the closest stroke, $CStr(str_{ref}) = \underset{str_p \in \{str_i\}}{\arg\min}\, dist_{sp}(str_{ref}, str_p)$ where $CStr(str_{ref})$ is not necessary equal to $CStr(CStr(str_{ref}))$. At the end, we can extract the spatial relation couples from the edges of relational graph for clustering to find the spatial relation prototypes.

For instance, we consider the stroke (1) as the reference stroke in Fig. 5.3. We can see that the reference stroke (1) has intuitively some obscure symbol relationships with the nearby strokes (2) and (3). As an example, we choose the $n_{cstr} = 1$ closest stroke to create the relational graph. The relation graph shows that $(CStr(Stroke(1)) = Stroke(2)) \neq (CStr(Stroke(2)) = Stroke(4))$. If we had created a complete directed graph between all the 14 strokes, the number of edges would be using Eq. (5.1) $14 \cdot (14 - 1) = 182$. Since we choose only the $n_{cstr} = 1$ closest stroke from a reference stroke, only 15 edges exist in the relational graph. The number of edges have been significantly reduced. We have 15 spatial relation couples at the end for clustering.

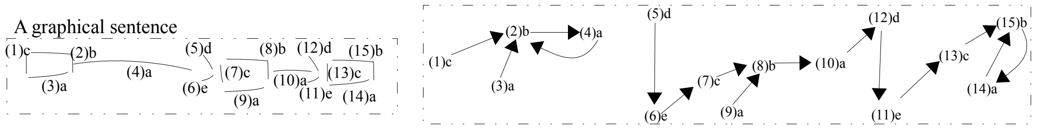


(a) A graphical sentence

(b) Corresponding relational graph

Figure 5.3: Creation of the relational graph (b) for a graphical sentence (a)

In another graphical sentence in Fig. 5.4a with a small difference that the strokes (5) and (6) are moved to right a little. If we still use the $n_{cstr} = 1$ closest stroke for producing the relational graph, the arrow $\{(4), (5), (6)\}$ will be separated into two parts as described in Fig. 5.4b. It means that this arrow will be lost in discovering symbol procedure. We can raise n_{cstr} for resolving this problem.



(a) Another graphical sentence flowchart

(b) Corresponding relational graph

Figure 5.4: The stroke (4) is far away from the strokes (5) and (6) in a graphical sentence (a). An arrow $\{(4), (5), (6)\}$ is separated into two parts in the relational graph (b)

In the next section, we will continue to use the flowchart in Fig. 5.3, and embed the spatial relations in fixed-dimension feature space for clustering.

5.3.3 Extracting Features for Each Spatial Relation Couple

We have created a relational graph for a graphical sentence and many spatial relation couples are extracted. In this section, we first extract features of a spatial relation couple between two strokes.

The spatial relation can be represented in three levels: distance relations, orientation relations, and topological relations [25]. In our case, the distance relation describes how far an argument stroke is from a reference stroke. We use $dist_{sp}(str_{ref}, str_{arg})$ as the *distance* relation (1 feature). The orientation relations illustrate some directional relations. A fuzzy model of directional relations between two strokes can be applied [29]. Using this fuzzy model, we choose eight fuzzy reference directional relations, *left*, *right*, *above*, *below*, *above-left*, *above-right*, *below-left*, and *below-right* which define 8 features in the range [0,1]. Concerning the topological relations, many different relations may be considered [37]. We use only a topological relation of *intersection* (1 binary feature). Since sizes of grapheme are ignored in the step of quantization of strokes, we add a *relative size* from a reference stroke to an argument stroke (1 feature). We define the diagonal length of the bounding box of a stroke str_i as $Dig(str_i)$. The relative size from a reference stroke to an argument stroke is $RS(str_{ref}, str_{arg}) = Dig(str_{arg})/Dig(str_{ref})$. Tab. 5.1 summarizes the value range of each feature. Four groups of features are extracted: Relative Size (S), Distance (D), Fuzzy Directions (F8), and Intersection (I).

	Relative Size (RS)	Distance(D)	F8	I
	$RS(str_{ref}, str_{arg})$	$dist_{str}(str_{ref}, str_{arg})$	Fuzzy Directions	Intersection
Range	$(0, +\infty)$	$[0, +\infty)$	$[0, 1]$	0 or 1

Table 5.1: Value range of each feature in spatial relation

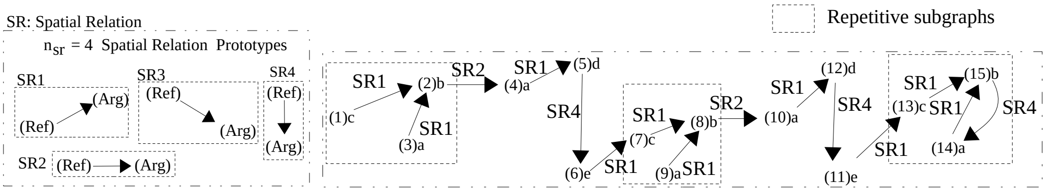
Considering the different value ranges of each feature, we use a double sigmoid function [70] to normalize $RS(str_{ref}, str_{arg})$ and $dist_{str}(str_{ref}, str_{arg})$ into $[0, 1]$. The double sigmoid function reduces the effect of the extreme large and small values (outliers). Thus, the 11 features are balanced in terms of their dynamics. Finally, we get a spatial relation vector of 11 dimensions to model a spatial relation couple between two strokes. In addition, those spatial relation features are also compatible with two multi-stroke symbols. The distance between two spatial relation vectors is simply defined as the Euclidean distance.

In fact, there are usually many relational graphs in a training part. In the next section, we will use the these spatial relation vectors extracted from relational graphs for clustering, and then we will get the spatial relation prototypes.

5.3.4 Quantifying Spatial Relation Couples

We have obtained many spatial relation vectors. In this section, we simply use *k-means* clustering algorithm to generate n_{sr} spatial relation prototypes. Therefore, edges (spatial relations) in relational graphs can be labeled with the closest spatial relation prototype. This labeling step is the quantization of spatial relations. After the quantization of edges, we got labeled-edge relational graphs between strokes from graphical sentences.

Fig. 5.5 shows an example where $n_{sr} = 4$ spatial relation prototypes (SR1 to SR4) are used. We quantify the spatial relations of the relational graph in Fig. 5.3b. As a targeted application, we will discover symbols which are repetitive sub-graphs on the relational graph in the next section.



(a) Spatial relation prototypes (b) Quantified relational graph and repetitive sub-graphs

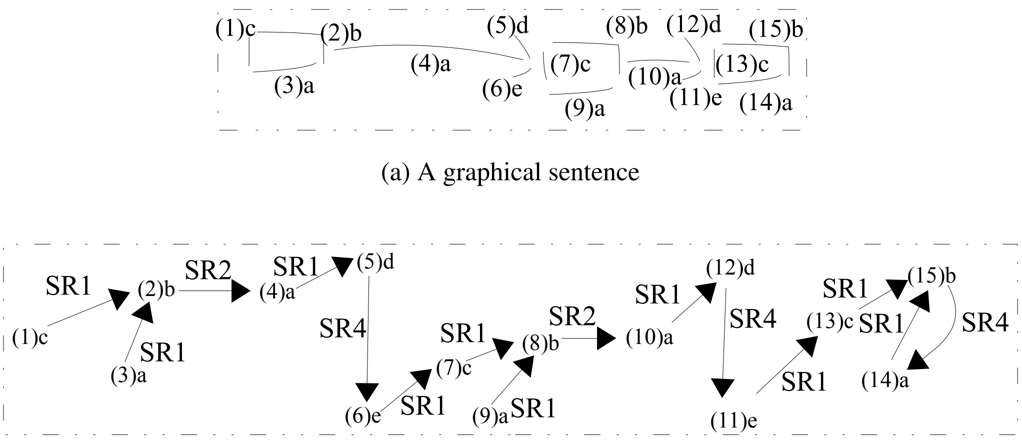
Figure 5.5: Quantization of spatial relations and an example of repetitive sub-graphs

5.4 Lexicon Extraction Using the Minimum Description Length Principle on Relational Graphs

In the previous section, we get the relational graph for a graphical document. This section presents an algorithm from [13] using the Minimum Description Length (MDL) principle [18] to extract repetitive substructures (sub-graphs) in graphs, which will be considered in our context as the lexical units. A simple example is presented in Section 2.1.2. The system SUBDUE (SUBstructure Discovery Using Examples) [23] iteratively extracts the best lexical unit (substructure) using the MDL principle. We will apply this system to extract lexical units, which could be hierarchical structure [24].

For explaining the iterative procedure and the hierarchical structure, we continue to use the example from the previous section as shown in Fig. 5.6. We have obtained a relational graph with the learned spatial relations for a graphical sentence, and then we try to extract a lexicon in this flowchart which contains a repetitive hierarchical structure "→ □".

There are three instances of "□" which is the most frequent sub-graphs in this



(a) A graphical sentence

(b) Corresponding learned relational graph

Figure 5.6: A corresponding relational graph (b) for a graphical sentence (a)

flowchart. Probably we could discover "□" as the first lexical unit LU_1 ("□") in the first iteration. In the second iteration, another frequent lexical unit LU_2 ("→") will be extracted. LU_i designates a discovered lexical unit in i^{th} iteration, as shown in Fig. 5.7. Obviously, this flowchart has to be a part of a training dataset which allows to compute frequency of possible lexical units. If LU_2 ("→") is more frequent in the training dataset, LU_2 will be extracted in the first iteration according to the MDL principle.

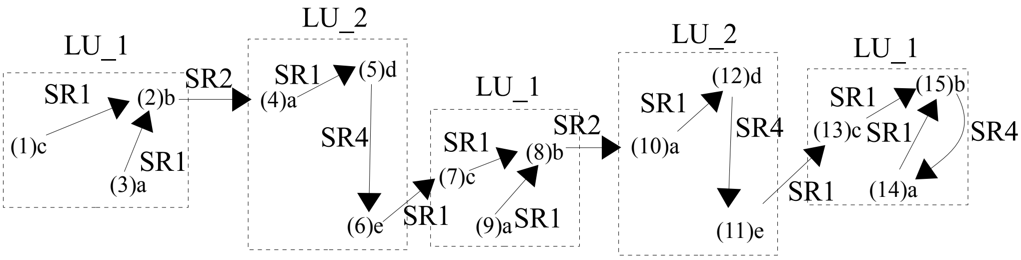


Figure 5.7: Two discovered lexical units LU_1 (3 instances of "□") and LU_2 (2 instances of "→") in the first and the second iteration respectively

In each iteration, we replace all the instances of a lexical unit with its name in the relational graph. In the third iteration, we can get a new relational graph as shown in Fig. 5.8. Keeping the lexical unit discovering procedure, we could probably get a hierarchical lexical unit LU_3, which contains LU_1 and LU_2. When no more lexical unit can reduce the description length, we get a lexicon which is a list of lexical units L =(LU_1, LU_2, LU_3,...) in terms of $DL(G, u)$.

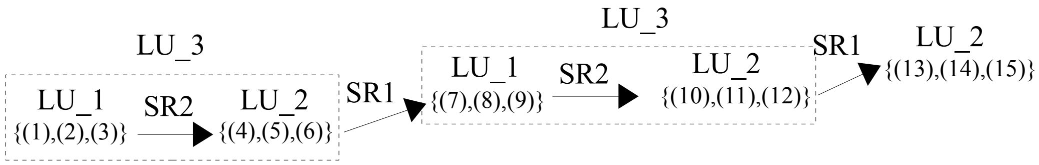


Figure 5.8: A hierarchical lexical unit LU_3 composed of LU_1 and LU_2 is extracted

At the end, we extract a sorted list of lexical units from the relational graphs using MDL principle. We call this list of lexical units as a lexicon. In the next section, we evaluate the extracted lexicon using a task of hierarchical segmentation of new graphical evidences from the same graphical language. To evaluate this hierarchical segmentation, we will used the already defined metric M_{Recall} from Section 4.4.2. The experiment results will be presented in the next section.

5.5 Experiments

There are many parameters have been defined in our discovery system. We found a problem that is how to optimize the parameters (noted as a *configuration*) in order to get a higher recall rate. We will use the greedy optimization algorithm [71] to optimize the configuration. In our experiment, the greedy optimization algorithm is described as:

1. Initialize all the parameters in a configuration.
2. One epoch:
   (a) Choose a parameter p in the configuration.
   (b) Find the value p with the maximum recall rate, and keep it in the configuration.
   (c) Until all the parameters are tested.
3. Check if the epoch number arrives the maximum number or if the best recall rate remains stable. If yes, the algorithm will stop. Otherwise it will go to the step 2.

We use the word "epoch" instead of "iteration" to avoid an ambiguity of SUBDUE iterative learning. The configuration contains four main parameters:

1. n_p denotes the prototype number during hierarchical clustering. In this experiment, we call the generated prototypes as graphemes.
2. SRF (Spatial Relation Feature) denotes which spatial relations have been used.
   (a) Distance (Dist)
   (b) Relative Surface (RS)
   (c) Eight Fuzzy Relations (F8)
   (d) Intersection (I)
3. n_{sr} denotes the spatial relation prototype number during the *k-means* clustering.
4. n_{cstr} denotes how many edges from a reference stroke to n_{cstr} closest argument strokes.

Since each epoch is time-consuming, we have finished two epochs. Because there are too many figures from experiments, the figures from 1^{st} epoch have been skipped. We only show the experimental figures from 2^{nd} epoch. In the 1^{st} epoch, we got an optimized configuration as shown below:

- $n_p = 50$,
- $SRF =$"$Dist|RS|F8|I$",
- $n_{sr} = 10$,
- $n_{cstr} = 3$.

We use this configuration for the two datasets, *Calc* and *FC*. We will evaluate an optimized configuration on the training part, and then the optimized configuration will be assessed on the test part.

5.5.1 Parameter Optimization on the *Calc* Corpus

In this section, we will optimize the configuration obtained in the 1^{st} epoch on the training part of *Calc* corpus. The configuration is shown in the previous section. We try a range of codebook size from 5 to 500. Fig. 5.9 shows recall rates at the multi-stroke symbol level M_{Recall} during the discovery procedure. As mentioned in Section 5.4, SUBDUE iteratively discovers lexical units (sub-graphs) from graphs. The x-axis denotes the SUBDUE iteration number. Each curve represents an iterative learning using a fixed codebook size (see figure legend). M_{Recall} always increases since SUBDUE discovers more and more graphical symbols. Among these curves, green curves outperform the others. It means that the best codebook size is locate between 40 and 90. Comparing the best $M_{Recall} = 51.2\%$ attained in Section 4.5 using the MDL principle on relational sequences, the proposed method on relational graphs got obviously higher recall rates at a range of 70%.

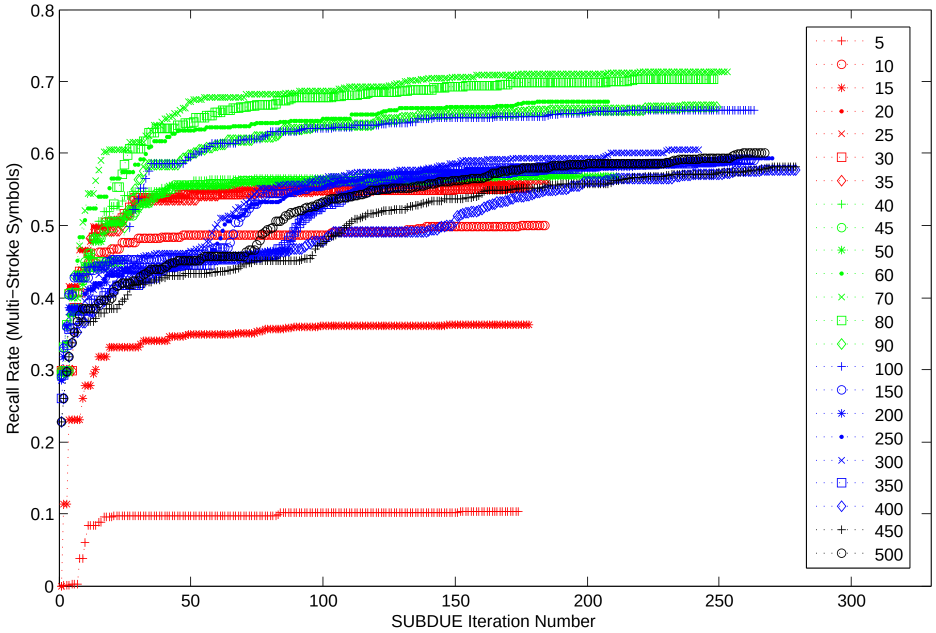


Figure 5.9: SUBDUE iterative discovery procedure in the 2^{nd} epoch (*Calc*, Codebook Size Selection n_p)

Final recall rates are illustrated in Fig. 5.10 when no more lexical units are found by SUBDUE. The best recall rate is 71.3% using the number of 70 graphemes (codebook size). This recall rate is much more better than a recall rate of 52.1% mentioned in Section 4. It got a higher M_{Recall} by 19.2%; 470 more multi-stroke symbols are found. In this segmentation task, constructing relational graphs succeed to find more multi-stroke symbols.

During hierarchical clustering as studied in Section 2.3.2, we can obtain the dendrogram that describes how to merge two clusters. Hierarchical clustering is to merge two clusters with the minimum distance among all the cluster pairs. Fig. 5.11 shows this linkage distance that is used to reach the wanted prototype number. We can see in this figure that for the optimal $n_p = 70$ the linkage distance is 0.576. In this experiment, we use MHD for the similarity between two graphical symbols. If MHD between two graphical symbols is larger than 0.576, the two graphical

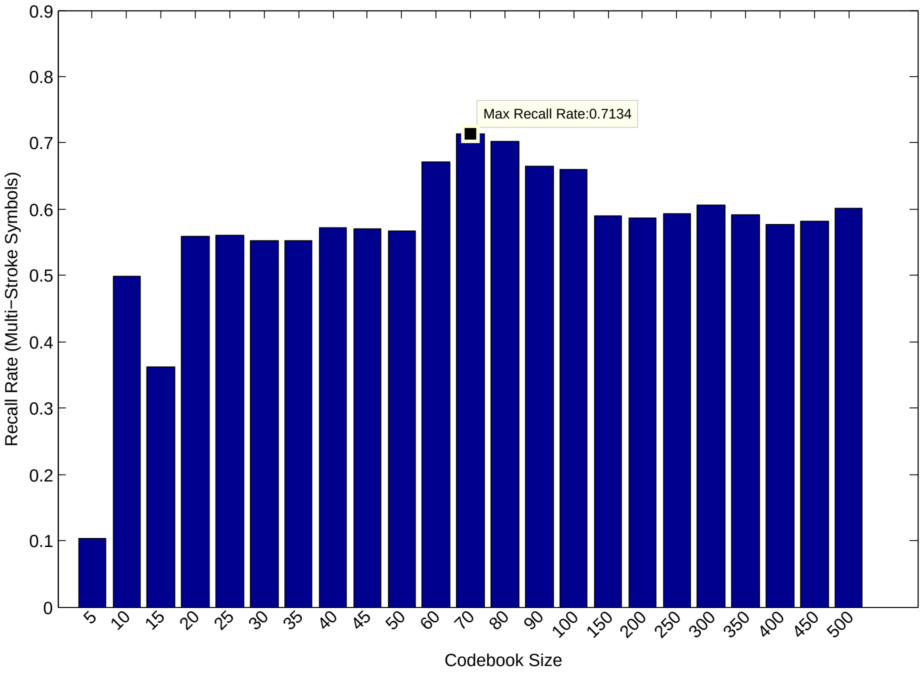


Figure 5.10: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (*Calc*, Codebook Size Selection n_p)

symbols won't be considered as the same symbol.

Using 70 graphemes to discover graphical symbols, we continue the optimization by looking for the best set of spatial relation features. Fifteen spatial relation feature sets (all the possible combinations) are chosen for discovering graphical symbols. Fig. 5.12 shows that a combination of *F8* and *I*, which obtains a recall rate of 78.9%, outperforms the others. We improve again the recall rate by 7.4%. It means that the unsupervised learned spatial relation is good for discovering the graphical symbols.

As mentioned in Section 5.3.4, we use *k-means* to generate the spatial relation prototypes. The number of spatial relation prototypes is an important parameter. Fig. 5.13 shows multi-stroke symbol recall rates according to different spatial relation feature prototype numbers. A maximum recall rate of 77.2% is obtained using the same 10 spatial relation feature prototypes. Since random initialization in *k-means*, recall rate may change with the same configuration. The recall rate is slightly lower than that in Fig. 5.12 with the same configuration. The recall rate is reduced by 1.5% but it is still stable.

Keeping the 10 spatial relation feature prototypes, we test different n_{cstr} closest strokes from a reference stroke during relational graph construction. Fig. 5.14 illus-

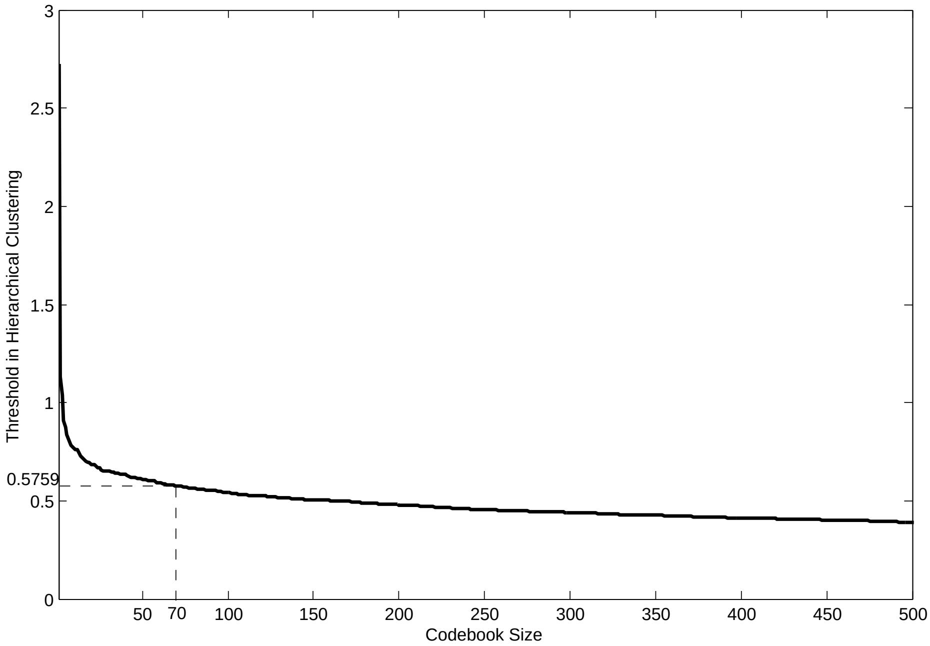


Figure 5.11: Linkage distance during hierarchical clustering in the 2^{nd} epoch (*Calc*, Codebook Size Selection n_p)

trates recall rates according to different n_{cstr} closest strokes. We find the maximum value recall rate when $n_{cstr} = 4$. It means that most of symbols can be found by 4 nearest strokes on this dataset. It achieves at a recall rate of 78%. This recall rate is almost same with the spatial relation prototype number test as illustrated in Fig. 5.13.

The whole system seems reach a stable state because the best recall rate does not change much in the two previous figures (Fig. 5.13 and Fig. 5.14). At the end of 2^{nd} epoch, we get an optimal configuration specified for the training of *Calc* dataset.

– $n_p = 70$ (Corresponding Threshold: 0.576)

– $SRF =$“$F8|I$”

– $n_{sr} = 10$

– $n_{cstr} = 4$

Using the optimal configuration, we obtained the same recall rate of 78% at the multi-stroke symbol level on the test part of *Calc* dataset. On this simple dataset, we can keep the same recall rate. That means our proposed discovery method is stable. Our previous work [26] reported the recall rate of 62.3% (84.2% including all the symbols) on the test part. We got a much higher recall rate by 15.7% at the multi-stroke symbol level in this thesis. In the next section, we will run the same

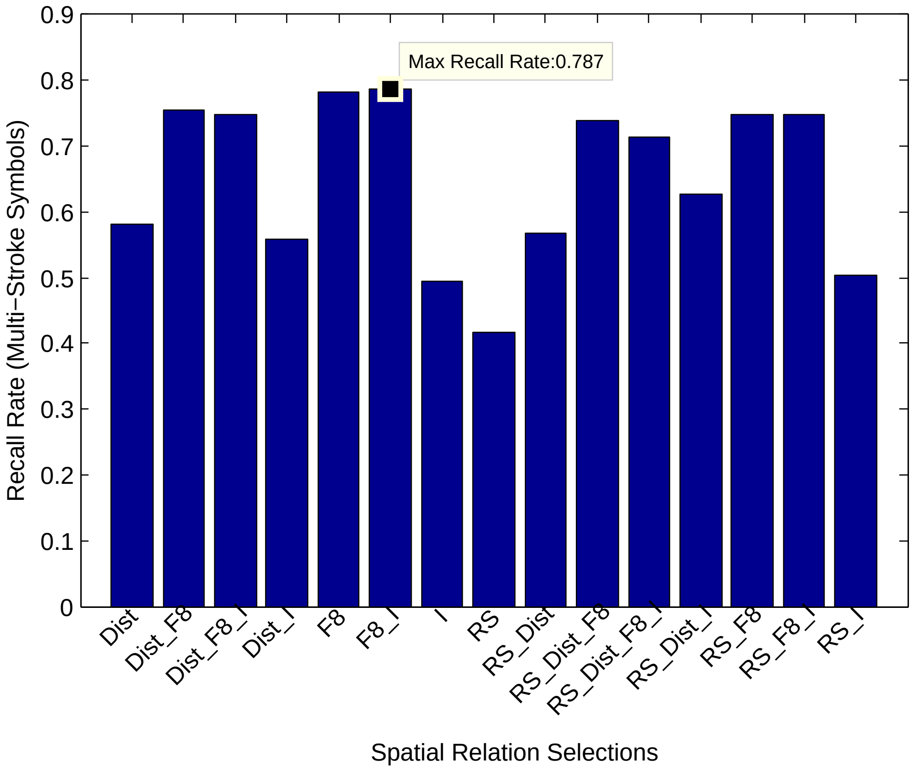


Figure 5.12: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (*Calc*, Spatial Relation Feature Selection)

experiment protocol on the more challenging *FC* corpus.

5.5.2 Parameter Optimization on the *FC* Corpus

In this section, we run our symbol discovery system on the *FC* dataset. As displayed in Fig. 3.14 (see Section 3.6), the *FC* dataset contains much more proportion of multi-stroke symbols. This high proportion of multi-stroke symbols leads to a higher complexity on the *FC* dataset. The same experiment protocol is carried out on this dataset with the initial configuration as defined at the beginning of Section 5.5.

The codebook size n_p is first tested in a range of $[5, 500]$, and the learning curves are shown in Fig. 5.15. The red curves and the green curves attain better recall rates. Fig. 5.16 reports a maximum recall rate of 53.1% with the same final best codebook size of 70. We can see that the *FC* dataset contains more strokes per symbol on average as shown in Tab. 3.2. Thus, it is more difficult to discover multi-stroke symbols. The maximum recall rate will be lower in theory.

Comparing the recall rates on the *Calc* dataset (see Fig. 5.15 vs Fig. 5.9), it

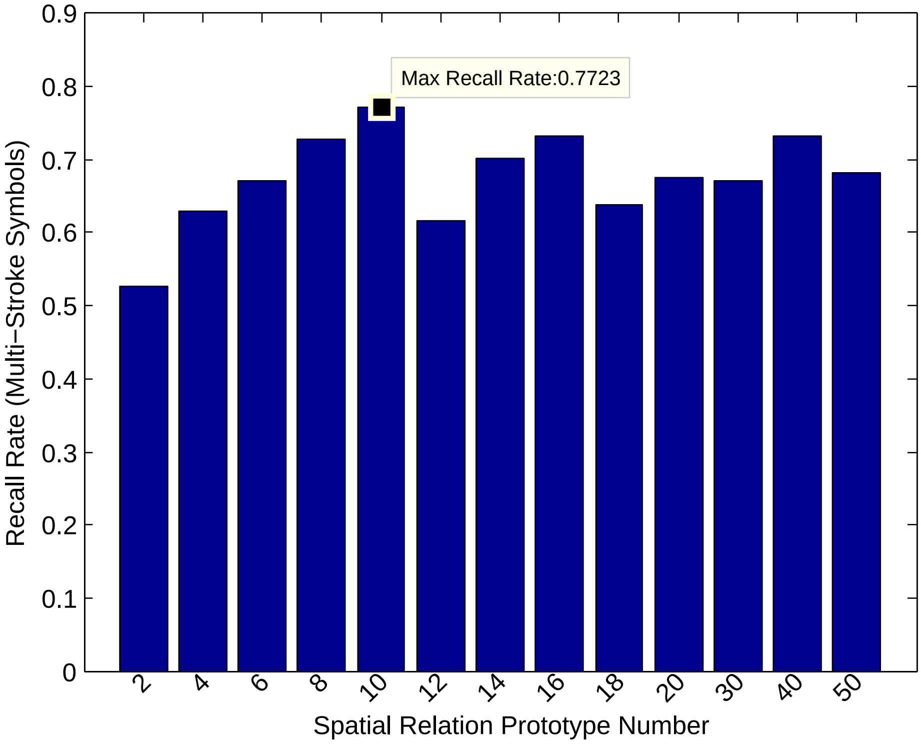


Figure 5.13: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (*Calc*, Spatial Relation Feature Prototype Number n_{sr})

seems that a smaller codebook size (larger MHD distance) was preferable on *FC* dataset. However, Fig. 5.17 reports a smaller MHD distance of 0.533 than 0.576 shown in Fig. 5.11. This smaller MHD distance is used to control the codebook size in the hierarchical dendrogram. It means that we need a more sensitive distance to group graphical symbols on a more complex dataset. A more distinguished distance function is preferable on the *FC* dataset.

Using 70 graphemes for the codebook, we try to select the best set of spatial relations. The recall rates are shown in Fig. 5.18. $F8$ achieves the maximum recall rate of 55.2%. Comparing the best feature set $\{F8\ , I\}$ of *Calc* shown in Fig. 5.12, *Calc* need the "Intersection" (I) spatial relation. In fact, many symbols on *Calc* can be detected by connected strokes, e.g. "$+$", "4", etc. However, on the *FC* dataset, the "intersection" spatial relation is not so important. Moreover, the directional information $F8$ is more important, for instance the symbol arrow "$- >$".

Keeping selected features, we test different spatial relation prototype numbers in a range $[2, 300]$. Using 30 prototypes, a maximum recall rate of 55.5% is illustrated in Fig. 5.19. The number of spatial relation prototypes is larger than that on the *Calc* dataset. In fact, the spatial relations are more complex on the *FC* dataset.

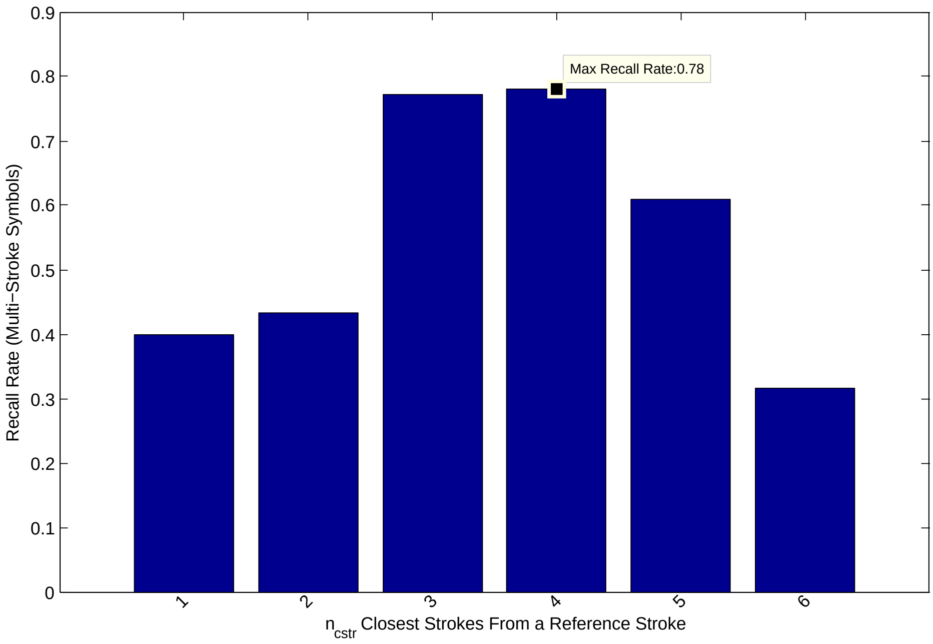


Figure 5.14: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (*Calc*, Closest Stroke Number n_{cstr})

We test different numbers of closest strokes n_{cstr} during the relational graph construction. We attain a maximum recall rate of 55.5% (1103 multi-stroke symbols) when $n_{cstr} = 3$ as shown in Fig. 5.20. Most of symbols are composed of 3 neighbor strokes. This parameter is quiet close to that on the *Calc*.

At end of the 2^{nd} epoch, we get an optimal configuration that is specified for the *FC* dataset. We can see the recall rates from the last two experiments (Fig. 5.19 and Fig. 5.20) become stable. The configuration is illustrated as below:

- $n_p = 70$ (Corresponding Threshold: 0.533)
- $SRF =$"$F8$"
- $n_{sr} = 30$
- $n_{cstr} = 3$

Using the optimized configuration in the previous section, we obtained a recall rate of 45.1% at the multi-stroke symbol level on the test part of *FC* dataset. On the more complex *FC* dataset, the recall rate is much lower than that on the *Calc* dataset. However, comparing the recall rate of 40.8% (52.8% including all the symbols) on the test part in our previous work [38], the recall rate of 45.1% is much higher in this thesis.

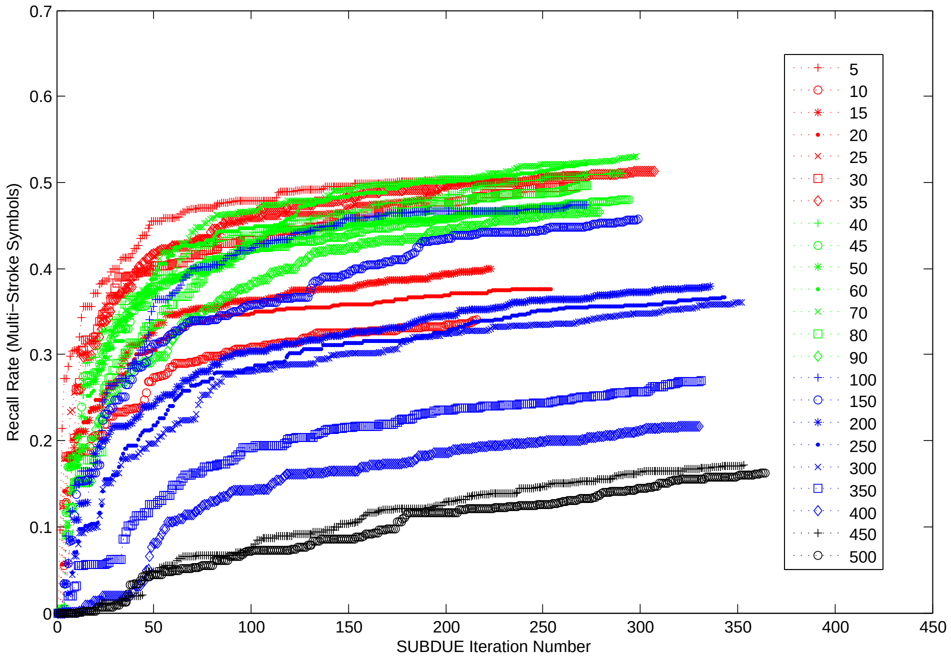


Figure 5.15: SUBDUE iterative discovery procedure in the 2^{nd} epoch (*FC*, Codebook Size Selection n_p)

5.6 Conclusion

In this chapter, we have proposed a multi-stroke symbol extraction approach on relational graphs between strokes using the MDL principle. An edge in the relational graphs represents a spatial relationship between two strokes. The proposed approach contains four main steps: (i) construction of a relational graph based on the closest strokes, (ii) quantifying single strokes into graphemes as presented in Chapter 3, (iii) quantifying spatial relations for labeling the edges of the graph, and (iv) discovering multi-stroke symbols (lexical units) on relational graphs using the MDL principle. We build the relational graph between strokes using the n_{cstr} closest strokes. Quantifying strokes consist in finding graphemes using hierarchical clustering and label them using the closest prototype. In order to quantify the edges in relational graphs using clustering, we extract spatial relation features at three levels: distance relation, orientation relation, and distance relation. Rather than a simple predefined spatial relation set, we generate spatial relation prototypes using clustering, which can be adapted for any graphical language. At the end, we will extract sub-graphs in relational graphs as multi-stroke symbols using the MDL principle. In the meantime, a symbol segmentation will be obtained. Evaluating

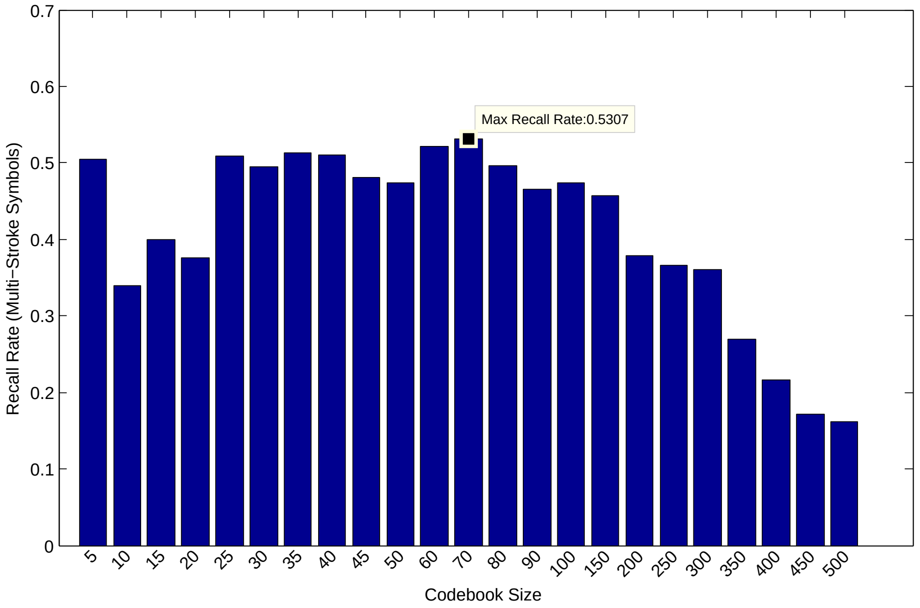


Figure 5.16: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (*FC*, Codebook Size Selection n_p)

by the segmentation measures proposed in the previous chapter, the MDL principle on the relational graphs works much better. Two optimal configurations are found on two datasets respectively, *Calc* and *FC*. Testing on the *Calc* dataset, we achieve recall rates at the multi-stroke symbol level, 78% on both the training part and the test part. On the more challenging *FC* dataset, recall rates of 55.5% and 45.1% are reported respectively on training part and test part. Those interesting results shows that we can get a correct hierarchical symbol segmentation to some extent. In the next two chapters, we will introduce an application based on this method.

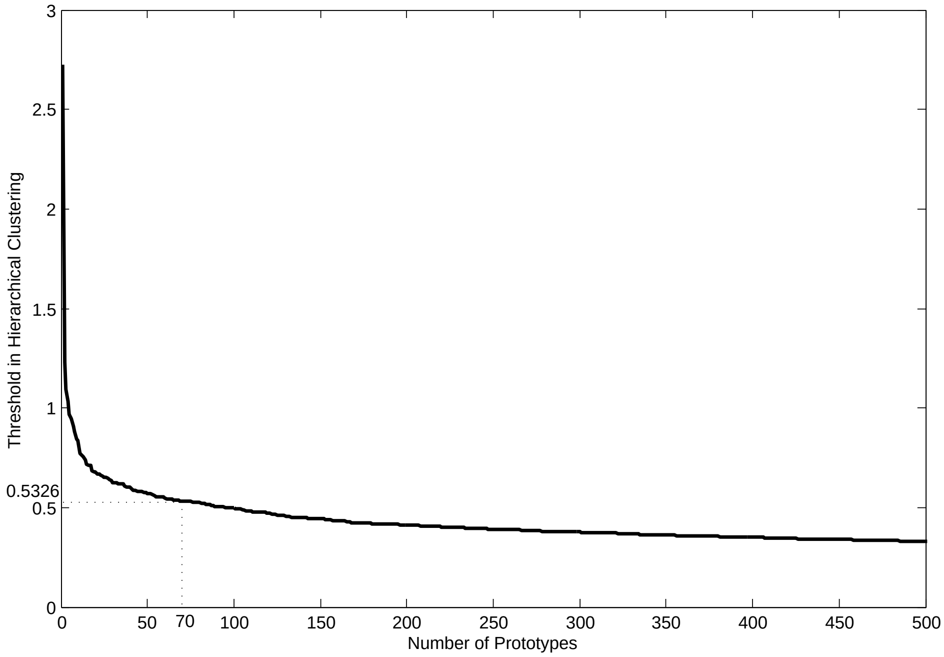


Figure 5.17: Linkage distance during hierarchical clustering in the 2^{nd} epoch (FC, Codebook Size Selection n_p)

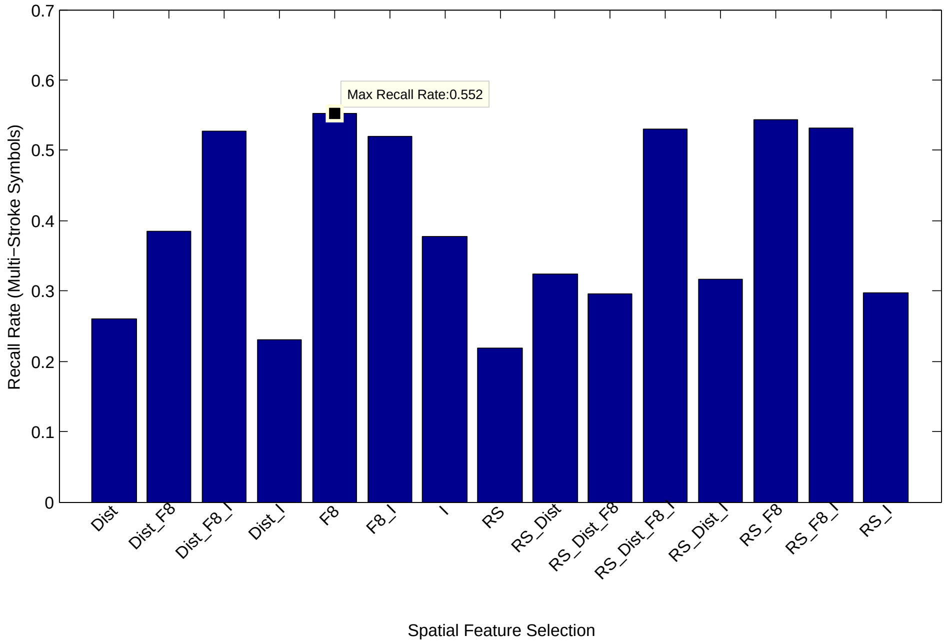


Figure 5.18: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (FC, Spatial Relation Feature Selection)

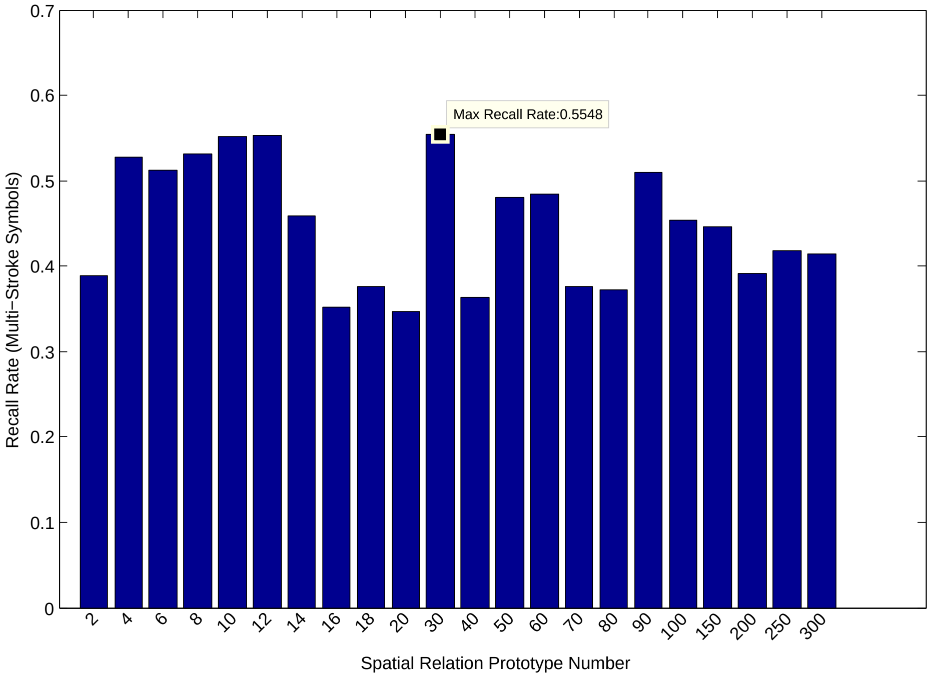


Figure 5.19: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (FC, Spatial Relation Feature Prototype Number n_{sr})

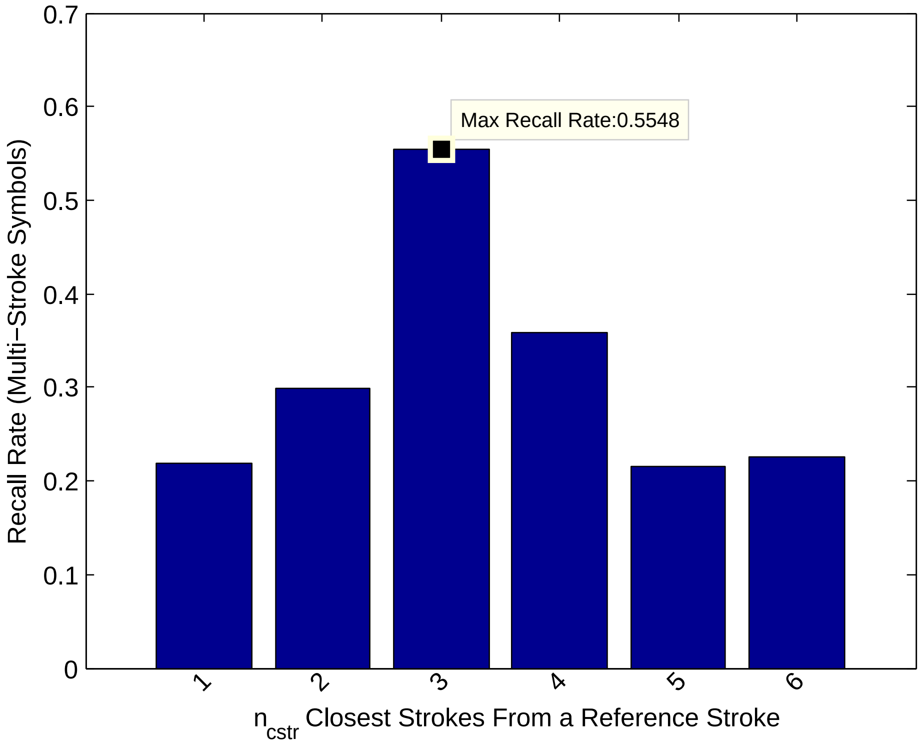


Figure 5.20: Recall rate at multi-stroke symbol level when a discovery procedure is finished in the 2^{nd} epoch (FC, Closest Stroke Number n_{cstr})

6

Reducing Symbol Labeling Workload using a Multi-Stroke Symbol Codebook with a Ready-Made Segmentation

The training of most of the existing recognition systems requires availability of large datasets labeled at the symbol level. However, producing ground-truth datasets is a tedious work. Two repetitive tasks have to be chained. One is to select a subset of strokes that belong to the same symbol, the next step is to assign a label to this stroke group. In this chapter, we discuss a framework to reduce the human workload for labeling at the symbol level a large set of documents based on any graphical language. This chapter towards a the first step in fully unsupervised process. Indeed here we use an existing segmentation and focus on the clustering algorithm and its evaluation. A hierarchical clustering is used to produce a codebook with one or several strokes per symbol. Then the codebook is used for a mapping on the raw handwritten data. Evaluation is proposed on the two different datasets, *Calc* and *FC* as shown in Section 3.6.

6.1 Introduction

Many existing recognition systems [1] need a training dataset which defines the ground-truth at the symbol level. To build the training dataset, we have to collect all the ink samples and label them at the symbol level, whereas it is a very long and tedious task. Hence, we propose to reduce this workload, so that most of the tedious works can be done automatically, by this way only a high-level supervision needs to be done to conclude the labeling process.

We can divide such a process into two steps, (a) segmenting handwritten scripts into symbols using the unsupervised symbol extraction method as shown in Chapter 5, and (b) grouping them into a codebook in which a user can label symbols in order to reduce human effort. This chapter is limited to the second step: the codebook generation, annotation and its assessment. An offline handwriting annotation system [14] proposes a similar idea to label a large number of well segmented isolated characters: clustering them into several clusters of characters, and labeling the clusters in order to reduce human effort.

Let us show an example to introduce the problem. Fig. 6.1 considers an example of a graphical language. For clarity, all the strokes are indexed "$(.)$". Fig. 6.2a displays the correct segmentation into symbols. The dashed rectangles represent the proposed segments in this figure. In the ideal case, each segment contains exactly one graphical symbol. A symbol shape usually represents a kind of symbol (the same label). Hence, according to their shapes, we group the segments in clusters. The corresponding clusters are shown in Fig. 6.2b. Choosing a pattern representative of each cluster yields a visual codebook used by a human to be labeled as "4", "+", "=", and "8" respectively (see Fig. 6.2c). Strokes in the pattern representative are marked by an index "$(*.)$". In this chapter, we choose, as the pattern representative, the segment which minimizes the sum of distances to the other segments in the same cluster. Hence, we can label the handwritten scripts at the codebook level (the high-level supervision) from the perfect symbol segmentation.

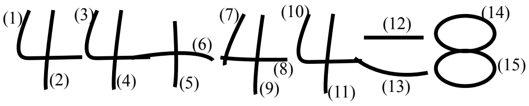
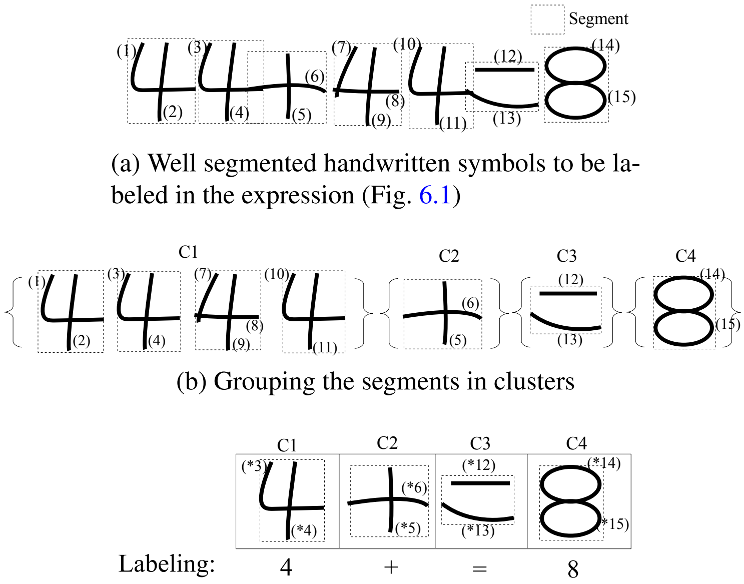


Figure 6.1: A raw handwritten expression



(a) Well segmented handwritten symbols to be labeled in the expression (Fig. 6.1)

(b) Grouping the segments in clusters

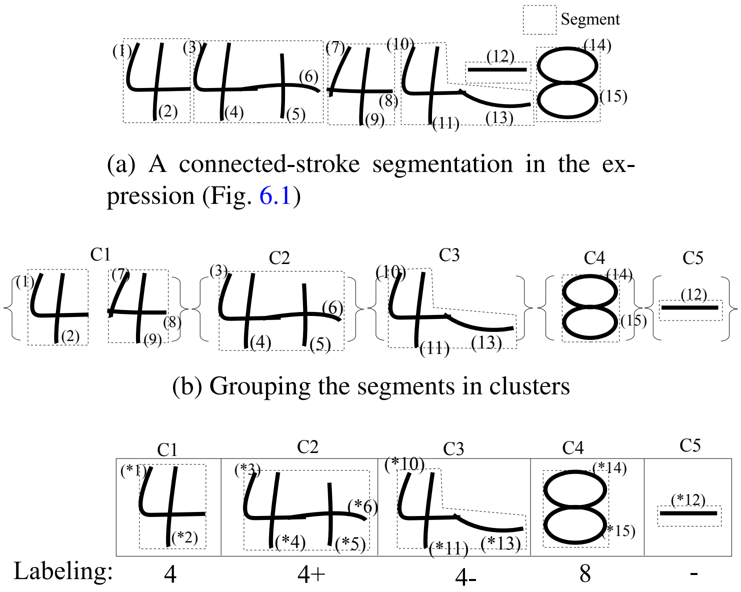
(c) Visual codebook for the user labeling

Figure 6.2: Reducing the human labeling workload in on-line handwritten graphical language in the perfect case.

However, generating the perfect segmentation (each segment being precisely composed of a symbol) is far from being trivial as discussed in Chapter 5. For instance, if we assume that the segmentation is based on an unsupervised learning scheme to extract frequent patterns, then some segments that contain a symbol plus sub-parts of another symbol, or even several symbols (multi-symbols) will be produced.

Similarly, if the segmentation is based on the connected strokes as displayed in the example of Fig. 6.3a, the same problem of multi-symbol segment will be present. In that case, the cluster *C3* contains the digit “4” and a sub-part of “=”, while the cluster *C2* contains two symbols, “4” and “+”. A user can separate the symbols, and then label them in the visual codebook (see Fig. 6.3c). The cluster *C3* can be labeled as “4-”. If we cannot recognize the sub-part of symbol “-”, the user will leave it unlabeled. In addition, the multi-symbol mapping problem will be studied, e.g. the cluster *C2* mapping.

After mapping the pattern representatives to the raw handwritten scripts with the codebook, some mistakes will be present. For example, we can find that the label “ – ” (minus) is wrong and we have to correct it. Thus, we also propose a criterion that measures how much work has been reduced. This criterion assesses the workload at the stroke level since in a manual labeling process, the ink is manipulated at



(a) A connected-stroke segmentation in the expression (Fig. 6.1)

(b) Grouping the segments in clusters

(c) Visual codebook for the user labeling

Figure 6.3: A connected-stroke segmentation and its labeling

the stroke level.

In this chapter, we introduce the proposed strategy for reducing workload on symbol labeling in Section 6.2. The codebook generation, its mapping and assessment are presented in Sections 6.3, 6.4, and 6.5 respectively. Experiment results and a conclusion will be given in Section 6.6 and in Section 6.7.

6.2 Overview

First of all, we introduce an overview of our annotation system in Fig. 6.5. The system is divided into three main steps: generating the segmentation (segments), clustering the segments and producing the codebook (different segment shapes), and codebook mapping from the user labeled codebook to the raw data.

In the first step, we need to generate a segmentation in order to apply our mapping procedure. Three different segmentations are used in this chapter. The first segmentation is user defined and corresponds to the ground-truth, i.e. the perfect segmentation. To study the ability of our algorithm to deal with multi-symbol segments, we produced an under-segmentation by merging the top-n frequent bigrams at the symbol level. A segment in a cluster can contain several symbols. It is a kind of side effects produced by our proposed method. Thus, this segmentation can evaluate the ability of multi-symbol mapping. This segmentation can be produced easily with the *Calc* dataset (presented in Section 3.6) where symbols can be ordered from left to right. For instance, top-1 frequent bigram as shown in Fig. 6.1 is “44”, and Fig. 6.4 shows the segmentation by merging “44” at the symbol level.

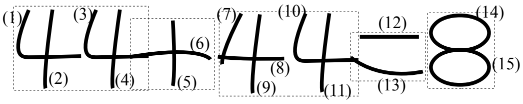


Figure 6.4: Merging the top-1 frequent bigram in Fig. 6.1

The two previous segmentations are original from the ground-truth segmentation, which is not available in real applications. To consider a real segmentation algorithm, the third segmentation is considered. It relies on the connected strokes to define a segment (like in Fig. 6.3a). Using these three segmentations, we will generate three different codebooks in the next section and then use them for the labeling stage.

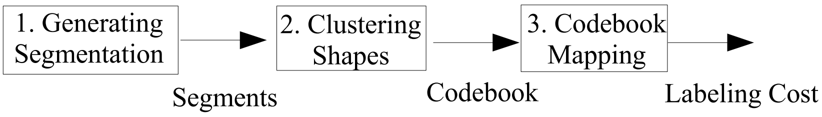


Figure 6.5: Three main steps on the annotation system

6.3 Codebook Generation using Hierarchical Clustering

In this section, we generate the codebook from the ready-made segmentation using the hierarchical clustering. Each segment may contain several strokes. In addition, because of the nature of on-line handwriting, two instances of the same symbol can be drawn with a different number of strokes, a different stroke order and different stroke orientations. To overcome this problem, we propose to use the Modified Hausdorff Distance (MHD) [35, 48] as presented in Section 3.5.3.

Thus we consider each segment as a set of points, $seg = \{pt\}$. For being size independent, all the segments should be normalized into a reference bounding box $\{x \in [-1, 1], y \in [-1, 1]\}$ by keeping the ratio. In addition to raw data (x, y), we used the eight orientations, and the local curvature (cosine) to have the 11-feature local description of a point as described in Section 3.3. The MHD distance will be used in this chapter for the two datasets, *Calc* and *FC*, because the MHD distance largely outperforms the DTW distance on the *FC* dataset as presented in Section 3.7.2.

A clustering technique is needed for producing the codebook, which is then brought into play for computing the membership of each segment. As mentioned in the Section 3.2, we have chosen the hierarchical clustering since its convenience of tuning the number of prototypes from dendrogram. The membership of each segment is then generated: all the segments are grouped into n_p clusters.

We select, as the pattern representative, the sample seg_c which minimizes the sum of MHD distances to the other samples of the same cluster C:

$$seg_c = \arg\min_{seg_p \in C}\Big(\sum_{seg_q \in C} MHD_{seg}(seg_p, seg_q)\Big). \tag{6.1}$$

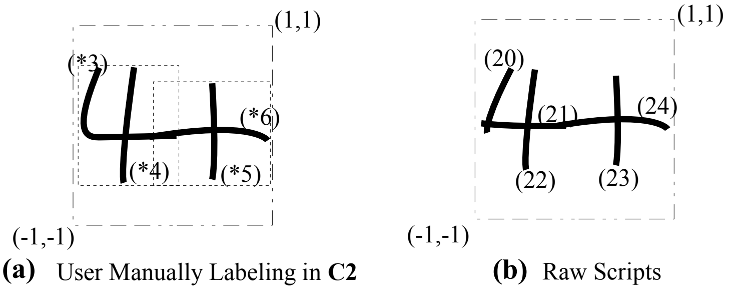
The pattern representatives will be organized as the visual codebook, an example is displayed in Fig. 6.2c. For example, a segment “4” is selected for the pattern representative among four segments “4”. We need the mapping process from the visual codebook to the raw scripts. In the next section, the codebook mapping problem will be discussed.

6.4 Codebook Mapping from a Visual Codebook to Raw Scripts

In the previous section, the codebook composed of multi-stroke segments has been obtained. A representative sample has been selected from each cluster to generate the visual codebook. A user labels therefore these chosen segments stroke by stroke in the visual codebook. In this section, we discuss how to label raw scripts with the labeled visual codebook.

In the visual codebook, segments in a cluster are not always from the same single symbol, e.g. the segment “4+” as shown in Fig. 6.3c represents more than one symbol. If we meet unknown symbols (sub-parts of symbol), we will leave them unlabeled. This task of segmentation and partial labeling of the representatives is quite simple. A mapping algorithm has been developed to complete the labeling of all unlabeled strokes in the original cluster. The mapping procedure involves normalizing a segment into a bounding box, and then searching for all unlabeled strokes with the closest labeled stroke using the MHD distance. After this mapping process, the symbols are segmented and labeled.

With the example of the cluster C2 and the visual codebook given in Fig. 6.3c, we assume a new instance of two symbols "4+" as displayed in Fig. 6.6b for better explanation. This instance contains one more stroke (5 instead of 4), and belongs to the cluster C2. A user has first to manually label the two symbols contained in the representative of the C2 cluster, i.e. the symbol "4" for the two strokes (*3, *4) and the symbol "+" for the two strokes (*5, *6), as displayed in Tab. 6.1a. In our system, each stroke is associated with a symbol index and its label. The symbol index denotes the symbol to which the stroke belongs (symbol segment).



(a) User Manually Labeling in **C2**

(b) Raw Scripts

Figure 6.6: The user manually labels the cluster C2 (a), and then the system finds a mapping for raw scripts (b).

Then we have to automatically label the remaining strokes (20 to 24) belonging to C2. This is done by a mapping procedure to find the best match between the unlabeled strokes and the labeled ones. Considering the two segments $\{(*3),(*4),(*5),(*6)\}$ and $\{(20),(21),(22),(23),(24)\}$, Tab. 6.1 shows the mapping procedure which normalizes the segments and looks for the corresponding labeled stroke. The numbers of strokes between two mapping segments are not necessarily equal. The mapping pairs $\{\{(20)\rightarrow(*3)\},\{(21)\rightarrow(*3)\},\{(22)\rightarrow(*4)\},\{(23)\rightarrow(*5)\},\{(24)\rightarrow(*6)\}\}$ are achieved. The symbol "4" $\{\{(20)\rightarrow(*3)\},\{(21)\rightarrow(*3)\},\{(22)\rightarrow(*4)\}\}$ and the symbol "+" $\{\{(23)\rightarrow(*5)\},\{(24)\rightarrow(*6)\}\}$ are segmented and labeled.

We have presented the whole annotation process. Since there may be some

Str	Sym	Label
(*3)	1	4
(*4)	1	4
(*5)	2	+
(*6)	2	+

Str: stroke index
Sym: symbol index

(a) The representative labeled by the user

Str	Sym	Label
(20)	1	4
(21)	1	4
(22)	1	4
(23)	2	+
(24)	2	+

(b) A raw segment in the cluster

Table 6.1: Each stroke in raw segment (b) is given the label contained in its closest stroke of labeled representative (a).

errors produced from the symbol segmentation (e.g. the connected-stroke segmentation), we will introduce a labeling cost to evaluate how much annotation work has been reduced in the next section.

6.5 Labeling Cost

In the previous section, the visual codebook was manually labeled. We then execute the mapping procedure to label all the other segments. As the segmentation and clustering are not perfect, some errors have to be manually corrected by the user. These corrections increase the cost of symbol labeling. Since the user labels the segments and raw handwritten scripts in a dataset stroke by stroke, we define the labeling cost C_{label} at the stroke level by:

$$C_{label} = \frac{N_c + N_{db} - N_{correct}}{N_{db}}, \tag{6.2}$$

where N_c is the number of strokes in the proposed codebook, N_{db} is the number of strokes in the dataset, and $N_{correct}$ is the number of strokes which are correctly labeled in the original dataset. Thus, $N_{db} - N_{correct}$ is the number of strokes for which the label has to be corrected or filled in the original dataset. N_c and N_{db} can be easily obtained by counting how many strokes are in the codebook's representatives and in the dataset respectively. We compute $N_{correct}$ according to the number of strokes which correspond to correctly segmented and correctly labeled symbols. If $C_{label} < 1$, the system reduces the human effort for labeling. The lower labeling cost is preferable. In fact, we can consider C_{label} as the percentage of strokes in dataset which still need a manual operation. For instance, after labeling the visual codebook and mapping as shown in Fig. 6.3, the labeling cost is $C_{label} = \frac{15+12-13}{15} = 0.933$, which is not an interesting rate because of the small size of the example (15 strokes).

In the next section, our proposed method will be tested on the two different datasets as presented in Section 3.6, the single-line mathematical expressions (*Calc*) and the flowchart dataset (*FC*).

6.6 Evaluation

In this section, we evaluate different codebook sizes (prototype numbers) during the hierarchical clustering on the two datasets (*Calc*) and (*FC*) as introduced in Section 3.6, and with different segmentation methods. As an illustration, we also display a subset of the visual codebook.

6.6.1 Evaluation of Codebook Size:

As proposed in Section 2.3.2 [62], six different metrics can be used to control the hierarchical clustering. As the initialization, we use the *Average* metric to calculate the codebook. The comparison between the metrics will be discussed later.

Fig. 6.7 shows the labeling costs on the two datasets with two segmentations: the ground-truth segmentation and the connected-stroke segmentation. Using the ground-truth segmentation, the labeling cost is very low on both datasets respectively: 6.4% with 150 prototypes on the *Calc* dataset training part and 4.3% with 100 prototypes on the *FC* dataset training part. It shows that in the ideal case we can reduce most of the human workload. The *FC* dataset contains only 6 classes. Among the 6 classes, the arrows have a high shape variation. The *Calc* dataset is composed of more classes of symbols, but more stable for each symbol. That is why the *Calc* dataset needs less number of prototypes using the ground-truth segmentation.

Using the connected-stroke segmentation, the labeling cost on the training part of *FC* dataset reports a high value, 97.5% with 40 prototypes. It means that most of graphical symbols on the *FC* dataset are not connected-stroke component. On the training part of *Calc* dataset, the labeling cost is much lower, 46.9% with 250 prototypes, since the most of graphical symbols, digits, are connected-stroke component. Hence, the segmentation quality is vital for the labeling cost.

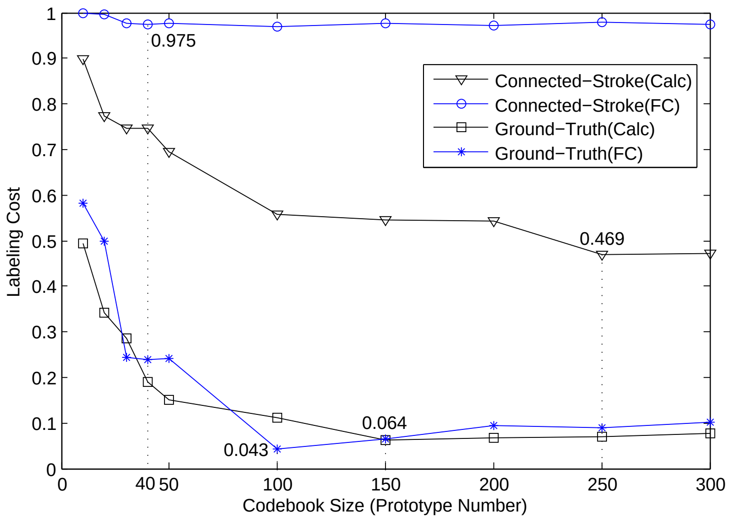


Figure 6.7: Labeling cost with different codebook sizes on the training parts of two datasets with the ground-truth segmentation and the connected-stroke segmentation

6.6.2 Evaluation on Hierarchical Clustering Metrics:

As the initialization configuration, we use the *Average* metric in the hierarchical clustering. In this section, we compare the six hierarchical clustering metrics on the two datasets respectively using their respective best codebook size: (1) *Single*, (2) *Average*, (3) *Complete*, (4) *Centroid*, (5) *Median*, and (6) *Ward* (minimum variance) as shown in Section 2.3.2. Fig. 6.8 shows the labeling cost for the six metrics using the ground-truth segmentation. Clearly, the *Average* metric reports the lowest labeling cost on the training parts of both datasets. The *Single* metric can be easily infected by outliers since it chooses the closest sample to compute the distance between two clusters. Hence, this metric got the highest labeling cost. The simple *Average* metric outperforms the others since the larger MHD distance in average means that larger dis-similarity in average between two clusters. As shown in Section 2.3.2, the others not only depends on not the MHD distance, but they also depends on other criterion, e.g. the *Ward* metric depends on the increment of variance.

6.6.3 Evaluation on Merging Top-N Frequent Bigrams:

In this section, we want to test the performance of multi-symbol codebook mapping in the system. A synthesis segmentation for this mapping will be created on the *Calc* dataset. The mathematical expressions are arranged from left to right. Us-

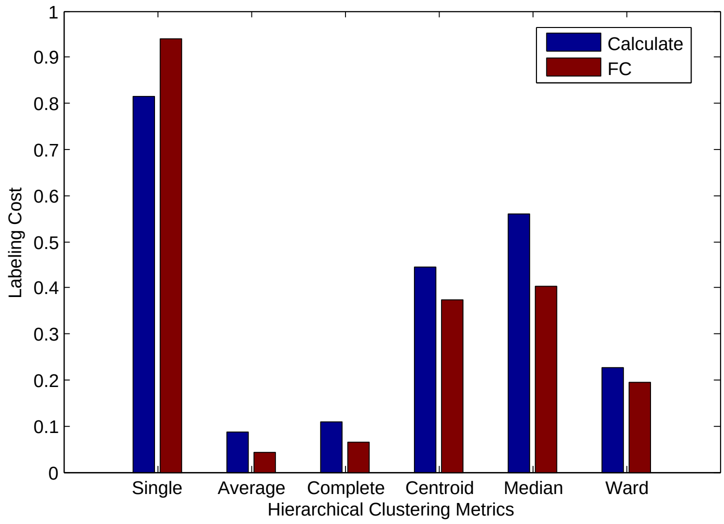


Figure 6.8: Evaluating the hierarchical clustering metrics on the training parts

ing the ground-truth segmentation, we can calculate the bigram distribution. The top-n (t_n) frequent bigrams are merged as new multi-symbol segments to test multi-symbol mapping in the codebook.

Fig. 6.9 shows the labeling cost on the training part of *Calc* dataset during the merging of the top-n (t_n) frequent bigrams produced from 0 to 50 with a step of 10. As shown in Fig. 6.9, two mapping methods are used. The first is the proposed multi-symbol mapping of this chapter. The second is the single-symbol mapping; each cluster are associated with only one label. The zero in x-axis means that the ground-truth segmentation is used. It shows that the multi-symbol mapping obviously works better, and is more stable. If two symbols are merged in one during the segmentation step, this error can be “saved” or “corrected” by the multi-symbol mapping.

6.6.4 Evaluation on Test Parts:

In the previous experiments, we use the training parts, actually used as validation sets, from the two datasets to choose the best parameter setting: 150 prototypes on the *Calc* dataset and the 100 prototypes on the *FC* dataset with the *Average* metric. Using these parameters and the connected-stroke segmentation, we obtain fair labeling costs of 50.4% (2292 strokes) and 97.2% (5889 strokes) on the test parts of the two datasets respectively. Nevertheless, using the ground-truth segmentation, labeling costs of 13.1% (596 strokes) and 13.5% (818 strokes) are achieved respec-

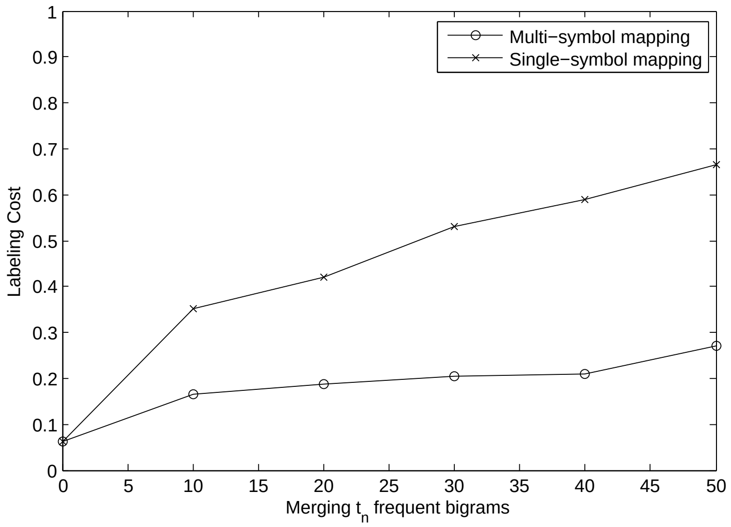


Figure 6.9: Labeling cost on merging the top-n (t_n) frequent bigrams on the training part of *Calc* dataset

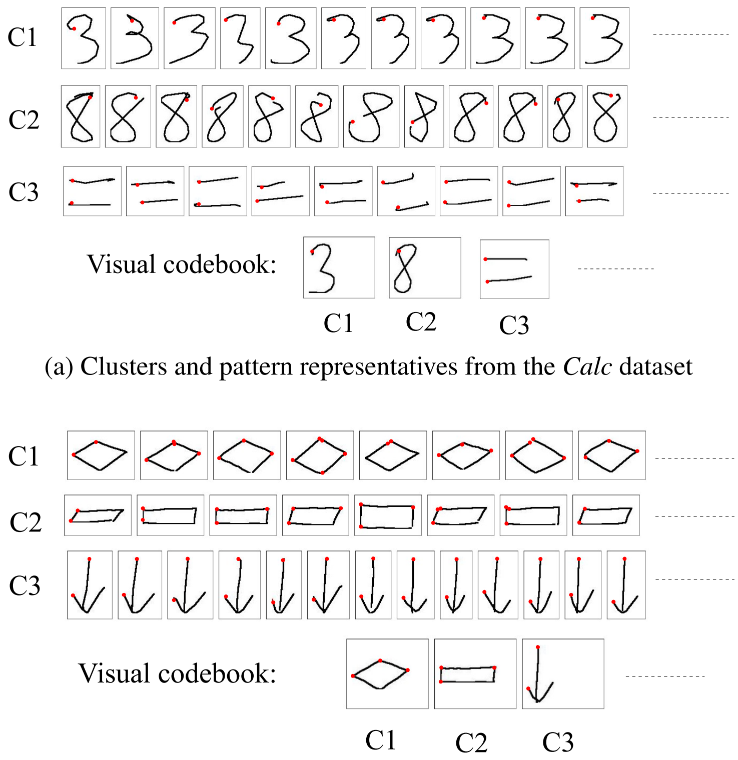
tively. These values show that the method is quite effective and that a lot of the annotation task can be saved.

6.6.5 Visual Codebook:

An illustration of the results of the clustering based on the ground-truth segmentation is displayed in Fig. 6.10. In these selected examples, we can see that the segment shapes are well grouped in the clusters. In each segment, a red point represents the starting point of a stroke. As displayed in Fig. 6.10a, several digits “8” with different writing orientations and different pen-down positions are actually grouped in the same cluster. It means that the proposed features can distinguish the symbol shapes very efficiently and correctly.

6.7 Conclusion

In this chapter, we proposed a framework for reducing the annotation workload using the codebook mapping for online graphical languages. Starting with a ready-made segmentation, the segments are grouped into the codebook using the hierarchical clustering. The visual codebook is generated for the user labeling. To evaluate the system performance, we define the labeling cost to evaluate how much labeling work has to be done by the user. On the test part of two datasets, *Calc*



(a) Clusters and pattern representatives from the *Calc* dataset

(b) Clusters and pattern representatives from the *FC* dataset

Figure 6.10: Clusters and pattern representatives

dataset and *FC* dataset, the low labeling costs of 13.1% and 13.5% are reported respectively using the ground-truth segmentation. Much of work has been reduced thanks to a good segmentation.

However, generating a good quality of segmentation is difficult by an unsupervised method. As shown in the experiment part, the labeling cost of connected-stroke symbol segmentation, which is a real segmentation method in state-of-the-art works, is still very high. Our approach as presented in Chapter 5 based on the MDL principle is a possible option for generating the unsupervised segmentation. In the next chapter, we will apply this unsupervised segmentation method to reduce the symbol labeling cost in this case.

7

Reducing Symbol Labeling Workload using a Multi-Stroke Symbol Codebook with an Unsupervised Segmentation

In the previous chapter, we have presented a method to reduce the symbol labeling cost using a mapping from the codebook to the raw scripts. Therefore, the ground-truth dataset can be more easily built for a recognition system. However, if when using the connected stroke segmentation, we can largely reduce the labeling cost on the *Calc* dataset, it was not the case not on the *FC* dataset. The key problem is how to produce a more precise symbol segmentation with an unsupervised way. In this chapter, we propose an iterative unsupervised handwritten graphical symbols learning framework which can be used for assisting such a labeling task. This framework mainly uses the symbol segmentation method as presented in Chapter 5. Initializing each stroke as a segment, we construct a relational graph between the segments where nodes are the segments and edges are spatial relations between them. To extract relevant patterns, quantization of segments and quantization of spatial relations are implemented. Discovering graphical symbols is then the problem of finding sub-graphs according to the Minimum Description Length (MDL) principle.

The discovered graphical symbols will become new segments for the next iteration. In each iteration, the quantization of segments yields the codebook in which the user can label graphical symbols. To evaluate this method, the labeling cost as defined in the previous chapter has been used. This method will be applied on the *Calc* dataset and the more challenging *FC* dataset.

7.1 Introduction

Graphical symbols, which are the lexical units of graphical languages, are composed of a spatial layout of single or several strokes. Usually everybody share some conventions about symbol shapes. The conventions allow individuals to read graphical messages comprising similar symbols. The ground-truth dataset, which contains those symbols, is vital for the current recognition system training. However, it is a tedious and time-consuming task to collect all the ink samples and label them at the symbol level.

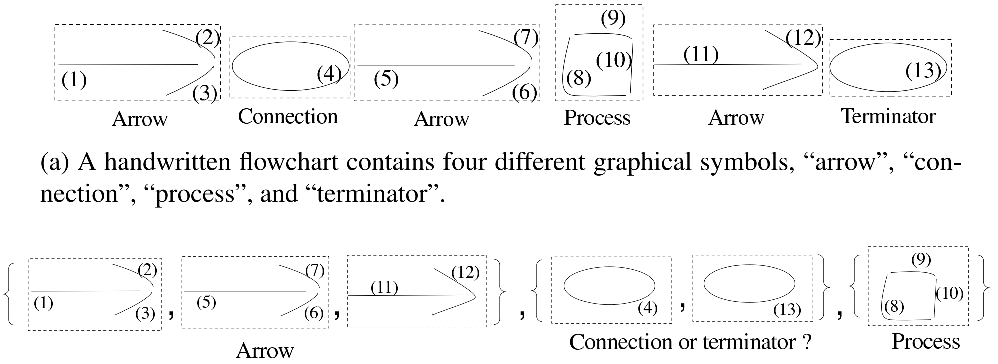
In the previous chapter, we have proposed a codebook mapping method which is a high-level supervision. This method can assist symbol labeling process with a correct symbol segmentation. Therefore, most of the tedious works can be done automatically. We remind that the basic units in on-line handwriting are strokes. A symbol is made of a single stroke or several strokes within confines of specific spatial composition. The key point is to identify the symbols from a large collection of handwritten strokes in spatial layouts. Let us illustrate some simple examples to understand the problem.

Imagine a document with only two different stroke shapes, e.g. “$-$” and “$>$”. Without any context, “$-$” and “$>$” might be regarded as two different symbols “minus” and “greater than” respectively. Each stroke corresponds directly to a single symbol. If the two strokes are placed together like “$\rightarrow$”, they will becomes another symbol “arrow”. The stroke is only a part of symbol. Eventually, the same kind of stroke according to the context will be either a single symbol or a piece of a more complex symbol. Searching the different shapes of strokes, termed as *graphemes*, has been studied in Chapter 3. In this case, we have two graphemes “$-$” and “$>$” for the two strokes respectively.

Let us put the two strokes together: it exists many composition rules named *spatial relations*. Two different symbols, “$\succ\!-$” and “$\rightarrow$”, can be constructed. The only difference between them is that the grapheme “$-$” is arranged on the right side in “$\succ\!-$” while on the left side in “$\rightarrow$”. These relations (left and right) are easily defined manually, but not for a complex graphical language, e.g. the *FC* dataset. Chapter 2.2 has studied the spatial relation learning. The spatial relation is defined from a reference stroke to an argument stroke at the three levels: distance, orientation, and topology.

Considering a more complicated example, Fig. 7.1a shows four different symbols, “arrow”, “connection”, “process”, and “terminator”. But the ground-truth is unknown in advance. To avoid an ambiguity that some strokes share the same grapheme, the stroke is referenced by their index $(.)$. Which set of strokes (a segment) represents a symbol? Why the combination of the strokes {(1),(2),(3)} is a valid symbol (actually “arrow”)? An intuitive answer is that the spatial composition is “frequent”; it exists two similar patterns in the layout, {(1),(2),(3)} and {(5),(6),(7)}, comprising same graphemes and same spatial relations respectively. Nevertheless, the equally frequent combination of less strokes {(1),(2)} does not mean a symbol. Moreover, the third arrow {(11),(12)} only contains two strokes, but its shape is similar with the previous two arrows. Graphical symbols with the same label (ground-truth) can contain different number of strokes and different graphemes. Hence, the problem is to search some repetitive patterns in a layout yielding to the *graphical symbols*. Chapter 5 presented the symbol searching method using the Minimum Description Length (MDL) principle. Therefore, a segmentation will be generated at the symbol level.

From the produced segmentation, we group segments in a small finite set of symbol hypothesis called a codebook with a higher semantic level. The codebook requires less annotation operations like in Fig. 7.1b. Only 3 segments have to be labeled instead of 6 symbols including 13 strokes in Fig. 7.1a. But all similar segments in a cluster of the codebook do not contain the same ground-truth: different symbols can be mixed in one cluster. For instance, the stroke (4) of symbol “connection” and the stroke (13) of symbol “terminator” are merged in the same cluster because of two similar shapes. Chapter 6 introduced a multi-symbol codebook mapping which is tolerant to some symbol segmentation errors produced by the MDL principle.



(a) A handwritten flowchart contains four different graphical symbols, "arrow", "connection", "process", and "terminator".

(b) A generated code book from the handwritten flowchart in Fig. 7.1a includes three sets of segments.

Figure 7.1: Reducing the human effort on labeling symbols

In this chapter, the proposed learning framework is revealed in Section 7.2. In this framework, we extract the codebook composed of multi-stroke symbols in which a user can label. The framework iteratively executes the symbol extraction method proposed in Chapter 5. The iterative learning process is explained using examples. Section 7.4 shows the system performance evaluated by the labeling cost as defined in Section 6.5.

7.2 Unsupervised Multi-stroke Symbol Codebook Learning framework

Our proposed automatic multi-stroke symbol extraction system is illustrated in Fig. 7.2. The on-line handwriting is imported in system, and the codebook is then exported for the annotation. Six main steps have to be taken into account in the system, which is an iterative learning. In the iterative learning framework, we consider the segment as the basic unit which may contain a complex multi-stroke structure or a simple one-stroke grapheme. The initial segmentation is set up with each single stroke. Using the symbol discovery method proposed in Chapter 5, we build firstly the relational graph with the segment as nodes and the spatial relations as edges. After the quantization of segments and of spatial relations, we make use of the SUBDUE system to discover the new symbols (sub-graphs). The segments in a new symbol will be merged into a new segment for the next iteration.

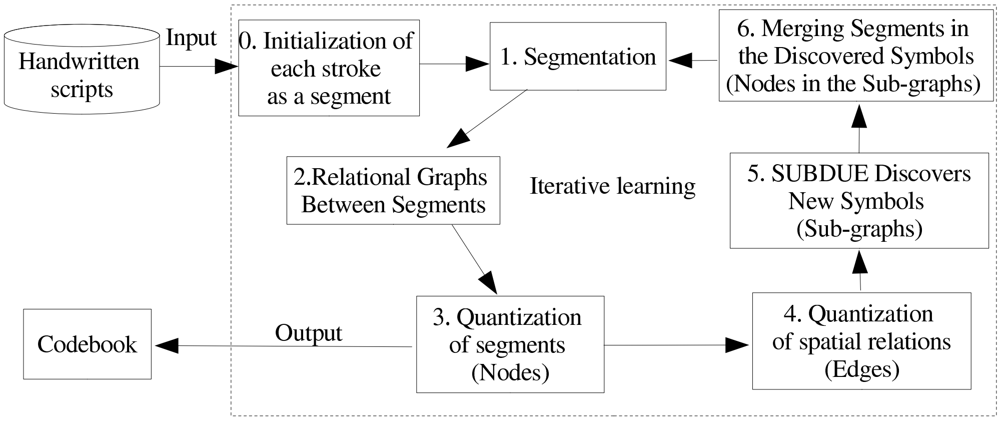

Handwritten scripts
Input
0. Initialization of each stroke as a segment
1. Segmentation
6. Merging Segments in the Discovered Symbols (Nodes in the Sub-graphs)
2.Relational Graphs Between Segments
Iterative learning
5. SUBDUE Discovers New Symbols (Sub-graphs)
Codebook
Output
3. Quantization of segments (Nodes)
4. Quantization of spatial relations (Edges)

Figure 7.2: Automatic multi-stroke symbol extraction system

7.2.1 Relational Graph Construction Between Segments

For the initialization step as shown in Fig. 7.2, we consider each stroke as a segment. In later steps, new segmentations will be generated from discovered symbols. The new segmentations will be discussed later. After obtaining the segmentation, we construct the relational graph between segments as presented in Section 5.3. The node is defined as the segment and the edge is defined as the spatial relation. The spatial relation is considered as a relationship from a reference segment to an argument segment. In other words, the relational graph is directed. We define n_{seg} as the number of segments. The number of out-directed edges from a reference segment should be limited to n_{cstr} closest segments where $n_{cstr} <= n_{seg} - 1$. This limitation can avoid a complete graph building. Furthermore, a limited perceived visual angle [69] exists in our eye; the symbols composed of the closest segments are preferable. However, if n_{cstr} is too small, we could lose some stroke compositions, and so some symbols.

As an example, Fig. 7.3a illustrates the generated relational graph ($n_{cstr} = 2$) in the first iteration for the flowchart in Fig. 7.1a. The relational graph contains 26 edges and 13 nodes. In the first iteration, each segment is composed of a single stroke. To search the patterns in the graph, we have to quantify the segments (nodes) and the spatial relations (edges). In the next section, we remind how to quantify the segments using the hierarchical clustering.

7.2.2 Quantization of Segments (Nodes)

In the previous section, the relational graph between the segments have been constructed. In this section, the segments will be quantified so that we can generate a codebook in which we can annotate the clusters.

We first choose a distance to calculate the dis-similarity between two segments for the clustering. As shown in Chapter 3.7.2, the MHD distance is the best dissimilarity function for the hierarchical clustering on the more complex *FC* dataset. Therefore, we propose to use the MHD distance as defined in Section 3.5.3. For the shape matching, the MHD distance between two segments is defined in Eq. (3.11).

In off-line data (images), the MHD distance uses usually only x and y coordinates of pixels (two features). In on-line data, we have fused local direction features and one curvature feature as described in Section 3.3 by computing the Euclidean distance $dist(pt_i, pt_j)$ between two feature vectors. The MHD distance therefore can distinguish "□" and "O", which have similar x and y coordinates, but different directions and curvatures for each point.

As studied in Chapter 3, we proposed to use the hierarchical clustering to regroup the segments. During the hierarchical clustering, the dendrogram will be produced. The dendrogram is a binary tree to describe how to merge two clusters in the hierarchical clustering. The number of clusters n_p can be determined by a distance *threshold* in the dendrogram. If the MHD distance between two clusters is less than the threshold, the two clusters will be merged into a new cluster.

The codebook size will be changed since more and more discovered symbols will be added into the codebook. In this chapter, we will find the best threshold to control the number of clusters. Therefore, in each iteration of system, the codebook size will be changed in terms of the threshold. Using the threshold, we assume that all the segments will be grouped into n_p clusters in an iteration.

We define the center sample seg_c who minimizes the sum of MHD distances to the other samples in the cluster C using Eq. (6.1). The center samples will be organized as a visual codebook. According to the iterative algorithm in Fig. 7.2, this quantization step can also be the last step before the output.

For instance, Fig. 7.3b displays that the segments are partitioned into $n_p = 7$

clusters in the first iteration from the segments in Fig. 7.3a. In reality, it exists a great number of samples in a cluster. We choose the center sample in each cluster, so that a visual codebook in Fig. 7.3c is generated. The user can therefore label these samples in the codebook at the higher level. This procedure is the quantization of segments. Afterward, we quantify the spatial relations in the relational graph.

7.2.3 Quantization of Spatial Relations (Edges) Between Segments

As studied in Section 7.2.1, we have created a relational graph between the segments. In the previous example, the number of edges in the relational graph is 26 as mentioned in Section 7.2.1 which means that it exists 26 spatial relation couples between two segments. In this section, we quantify the spatial relation couples into n_{sr} categories. We first extract the features of spatial relations. Section 5.3.3 proposed 11 features aiming at describing the three level spatial relations. The *K-means* clustering algorithm is applied to generate n_{sr} spatial relation prototypes, and then all the edges in relational graph are grouped into n_{sr} categories.

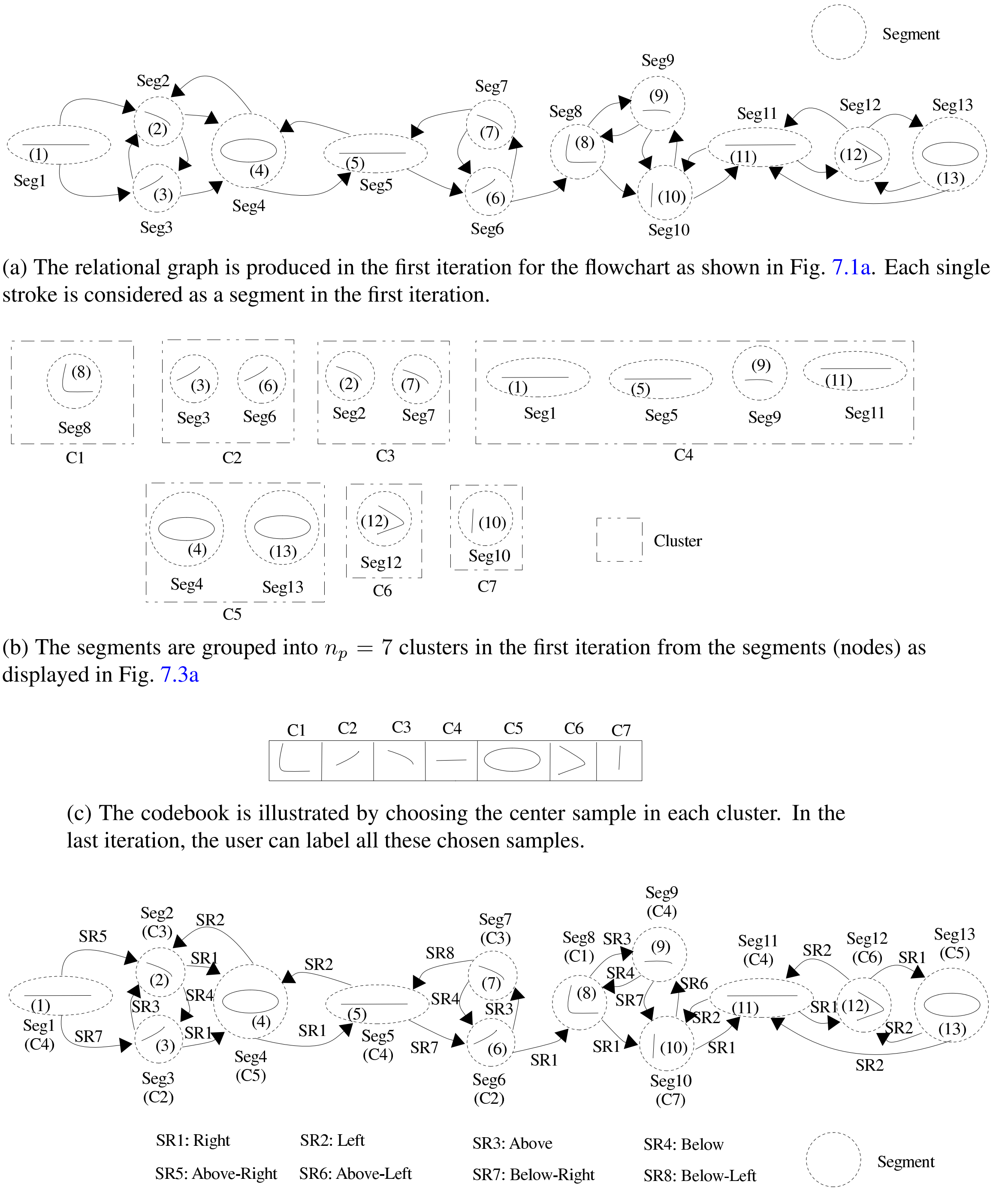
As an example, Fig. 7.3d shows that the nodes and the edges from relational graph as shown in Fig. 7.3b are quantified into $n_p = 7$ different shapes of segment (C1, C2,...,C7) and $n_{sr} = 8$ different spatial relations (SR1, SR2,...,SR8). To better understand the method, all the spatial relations are marked with spatial significations as displayed in Fig. 7.3d. In reality, such significations do not exist.

In the next section, we introduce a short remind using examples for the graphical symbol (sub-graph) extraction using the MDL principle as described in Chapter 5.

7.2.4 Discover Repetitive Sub-graphs Using Minimum Description Length

In the previous section, we have obtained the relational graph from a graphical language. Using the MDL principle presented in Chapter 5, we will extract repetitive substructures in the graph, which will be considered in our context as lexical units. Formally given a graph G, we choose the lexical unit u which minimizes the description length. We consider this lexical unit u as the graphical symbol.

For explaining a discovery procedure, we extract a lexical unit from the same



(a) The relational graph is produced in the first iteration for the flowchart as shown in Fig. 7.1a. Each single stroke is considered as a segment in the first iteration.

(b) The segments are grouped into $n_p = 7$ clusters in the first iteration from the segments (nodes) as displayed in Fig. 7.3a

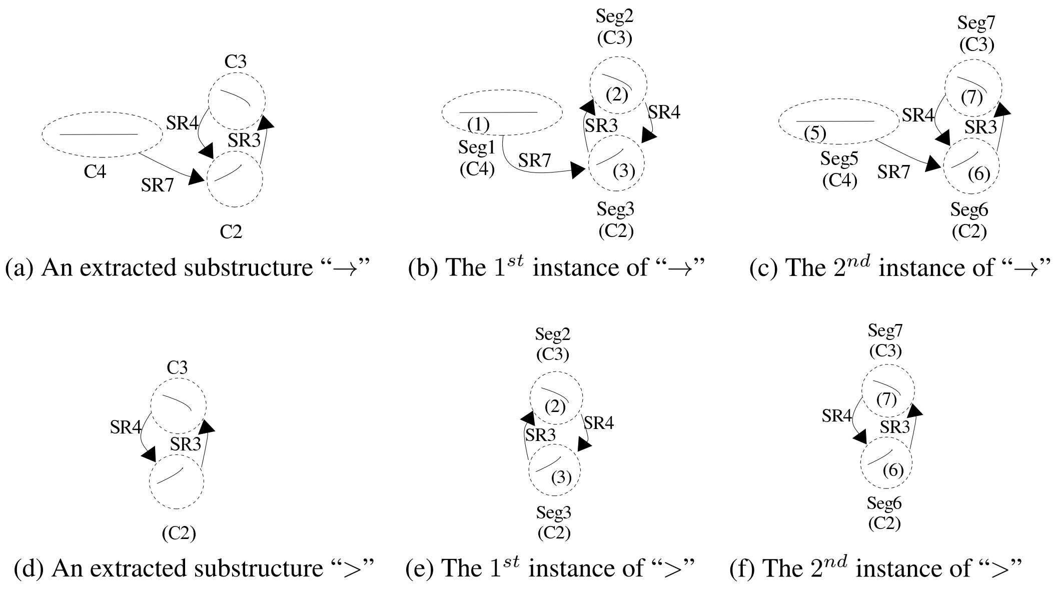
(c) The codebook is illustrated by choosing the center sample in each cluster. In the last iteration, the user can label all these chosen samples.

(d) The relational graph generated from Fig. 7.3a after the quantization of segments into $n_p = 7$ graphemes and the quantization of spatial relations into $n_{sr} = 8$ categories.

Figure 7.3: The learning procedure during the first iteration

example as displayed in Fig. 7.3d. The flowchart as shown in Fig. 7.3d is only one flowchart in the training set, but obviously in real data it exists many other varied flowcharts using the same symbol set. Fig. 7.4a and Fig. 7.4d illustrate two possible lexical units: two instances of "→" and two instance of ">". Considering all the flowcharts in the training set, if the occurrence number of ">" and the occurrence number of "→" are almost equal, the MDL principle will prefer the substructure "→" composed of more nodes and edges which can get a higher compression ratio in the graph. Similarly, if the occurrence number of ">" is much larger than that of "→", we will extract ">" as the lexical unit according to the MDL principle. In each iteration, we can discover $n_u \geq 1$ lexical units as the multi-stroke symbols using the SUBDUE system.

In the next section, we will present the iterative learning by merging the segments in a new symbol instance into a segment.



(a) An extracted substructure "→"

(b) The 1^{st} instance of "→"

(c) The 2^{nd} instance of "→"

(d) An extracted substructure ">"

(e) The 1^{st} instance of ">"

(f) The 2^{nd} instance of ">"

Figure 7.4: Two possible symbols in the first iteration as shown in Fig. 7.3d.

7.2.5 Iterative Learning

In the previous section, we have discovered one new symbol. However, these new symbols will change the original spatial relations which were between their subparts and the rest of the relational graph. Moreover, the new symbols and old segments may be similar. It would be better to redo the learning procedure.

Let's illustrate an example to understand the problem. We suppose that the symbol "$>$" as shown in Fig. 7.4d is extracted rather than "$\rightarrow$" as shown in Fig. 7.4a. The segments in the symbol instance will be integrated to another object. We can consider such object as a new segment. For instance as shown in Fig. 7.5a, two new segments Seg14 (Seg2 and Seg3) and Seg15 (Seg7 and Seg6) are created. We find that the shapes of the new segments Seg14 and Seg15 are similar with that of the old segment Seg12 which has only one stroke (12). Thus, a new codebook is needed to be calculated.

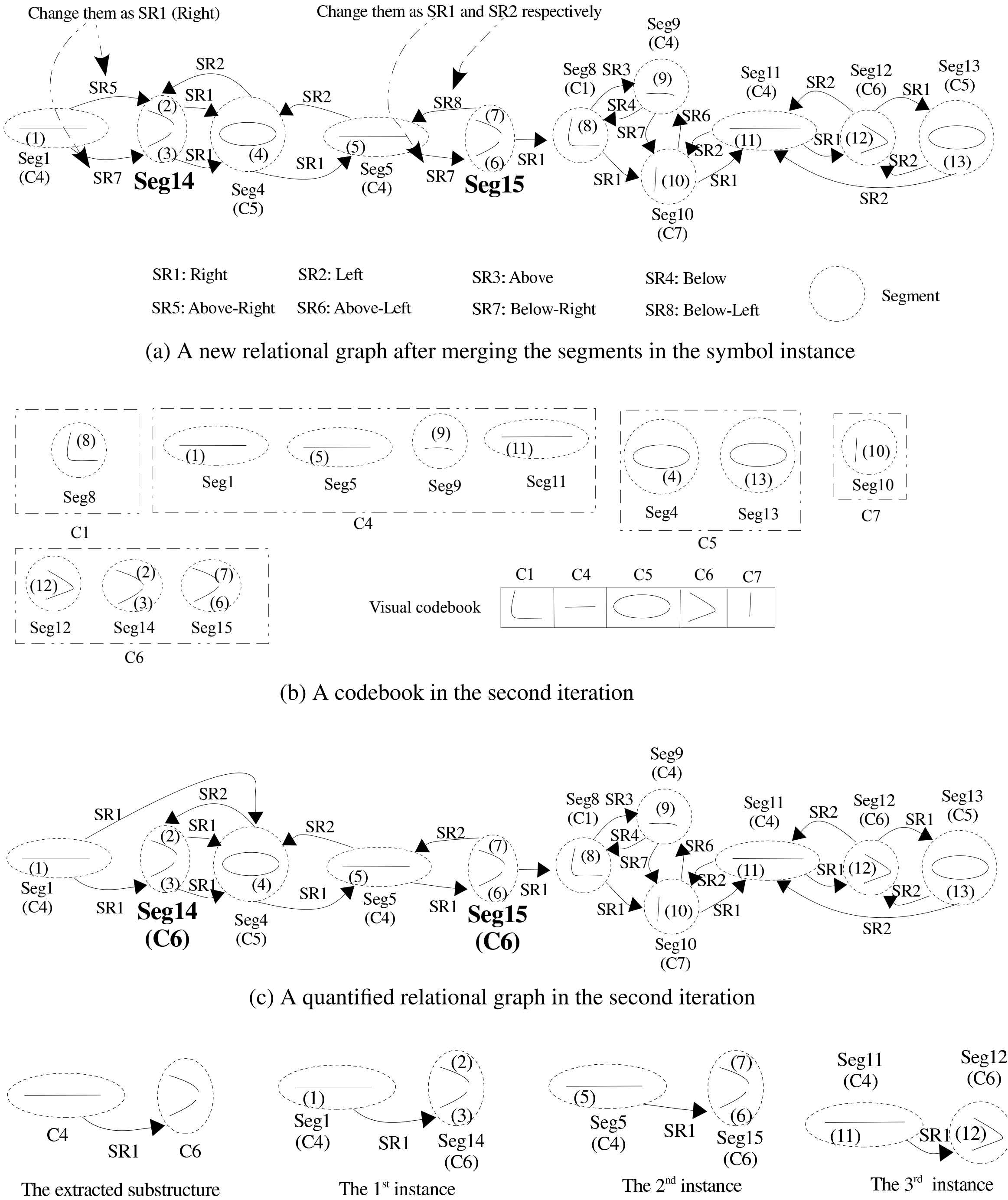
Considering the spatial relations as shown in Fig. 7.5a, Seg1 associates no longer the relation SR5(Above-Right) nor SR7(Below-Right) with Seg14, which is the combination of Seg2 and Seg3. Neither do the relation between Seg5 and Seg15. In fact, Seg14 is put on the right (SR1) side of Seg1. Therefore, it would be better to recalculate the relations between the all segments.

After merging, the second iteration begins. We group the segments into the second iteration codebook as shown in Fig. 7.5b. We can see that three "$>$", which are composed of different numbers of strokes, are in the cluster "C6". Similarly, a new relational graph in the second iteration is generated in Fig. 7.5c. The spatial relations (edges) have been rebuilt. Fig. 7.5d illustrates three extracted arrows from the relational graph in second iteration. We combine the segments from the instances of arrow to the new segments, Seg16, Seg17, and Seg18. The codebook in the third iteration is obtained in Fig. 7.6a. The user can label the symbols in this visual codebook.

The system will stop the iterative learning procedure when the SUBDUE cannot find any new symbols in the relational graph. Nevertheless, the previous steps, quantization of segments and of spatial relations, are time-consuming. Increasing the number of discovered symbols in each iteration n_u can speed up the system running.

7.3 Annotation Using the Codebook

In the previous section, the codebook composed of multi-stroke symbols has been obtained. We choose the center segment in the cluster to generate the visual codebook, and then the user labels these chosen segments stroke by stroke. We will



(a) A new relational graph after merging the segments in the symbol instance

(b) A codebook in the second iteration

(c) A quantified relational graph in the second iteration

(d) An extracted substructure in the relational graph in the second iteration

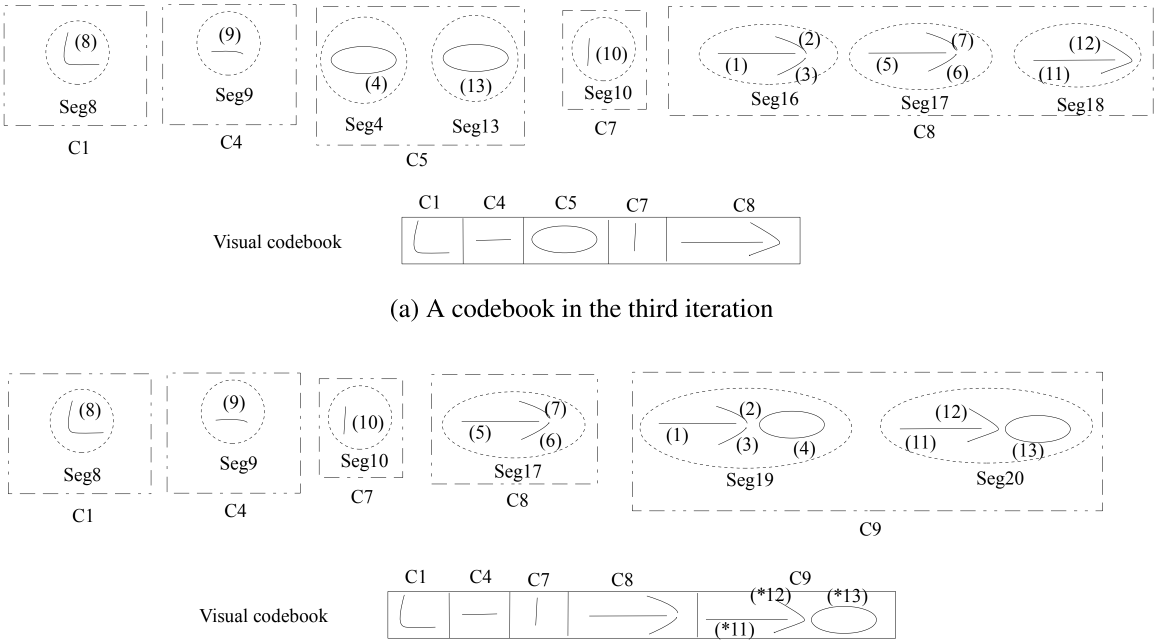
Figure 7.5: A learning procedure in the second iteration

apply the codebook mapping proposed in Chapter 6 to label raw scripts. Several examples will be given to explain.

In the visual codebook, the segments in a cluster are not always the same single symbol. Different symbols may be mixed in a cluster since the label is dependent on the context. Only the shape of segment cannot decide the label.

For instance, the two different labels “Connection” and “Terminator” are mixed in the cluster “C5” of the codebook illustrated in Fig. 7.6a since the label is dependent on the context. If we learn a cluster “$\rightarrow o \rightarrow$”, “o” will be easily recognized as the label “Connection”.

Another frequent phenomenon is that a segment, which contains several symbols (e.g. “$\rightarrow o \rightarrow$”), is over learned. The user can separate and label in the codebook representative. A mapping algorithm has been developed in Section 6.4 to search for the corresponding labeled stroke from the unlabeled segment to the labeled segment. The mapping procedure is involved in normalizing segments into a reference bounding box, and then in searching for the labeled stroke with the closest MHD distance as the corresponding stroke. After the mapping process, symbols are segmented and labeled.



(a) A codebook in the third iteration

(b) A codebook in the fourth iteration

Figure 7.6: codebooks in later iterations

As an example, using the third iteration codebook in Fig. 7.6a, we continue with the learning procedure and the fourth iteration codebook is attained in Fig. 7.6b. Segments in a cluster “C9”, which include two symbols “$\rightarrow$” and “O”, are over learned. A user label segments in a visual codebook stroke by stroke. A labeled segment of “C9” in the visual codebook is shown on the left side in Fig. 7.7a. However, the other segments in “C9” are unlabeled on right side in Fig. 7.7a. A mapping procedure is required to find the corresponding labeled stroke. Considering two segments $\{(*11), (*12), (*13)\}$ and $\{(1), (2), (3), (4)\}$ with different numbers of strokes, Fig. 7.7b shows the mapping procedure which normalizes the segments and looks for the corresponding labeled stroke using the closest MHD distance. The numbers of strokes between two mapping segments are not necessary equal. The mapping pairs $\{\{(1) \rightarrow (*11)\}, \{(2) \rightarrow (*12)\}, \{(3) \rightarrow (*12)\}, \{(4) \rightarrow (*13)\}\}$ are achieved. The symbol “Arrow” $\{\{(1) \rightarrow (*11)\}, \{(2) \rightarrow (*12)\}, \{(3) \rightarrow (*12)\}\}$ and the symbol “Terminator” $\{\{(4) \rightarrow (*13)\}\}$ are segmented and labeled.

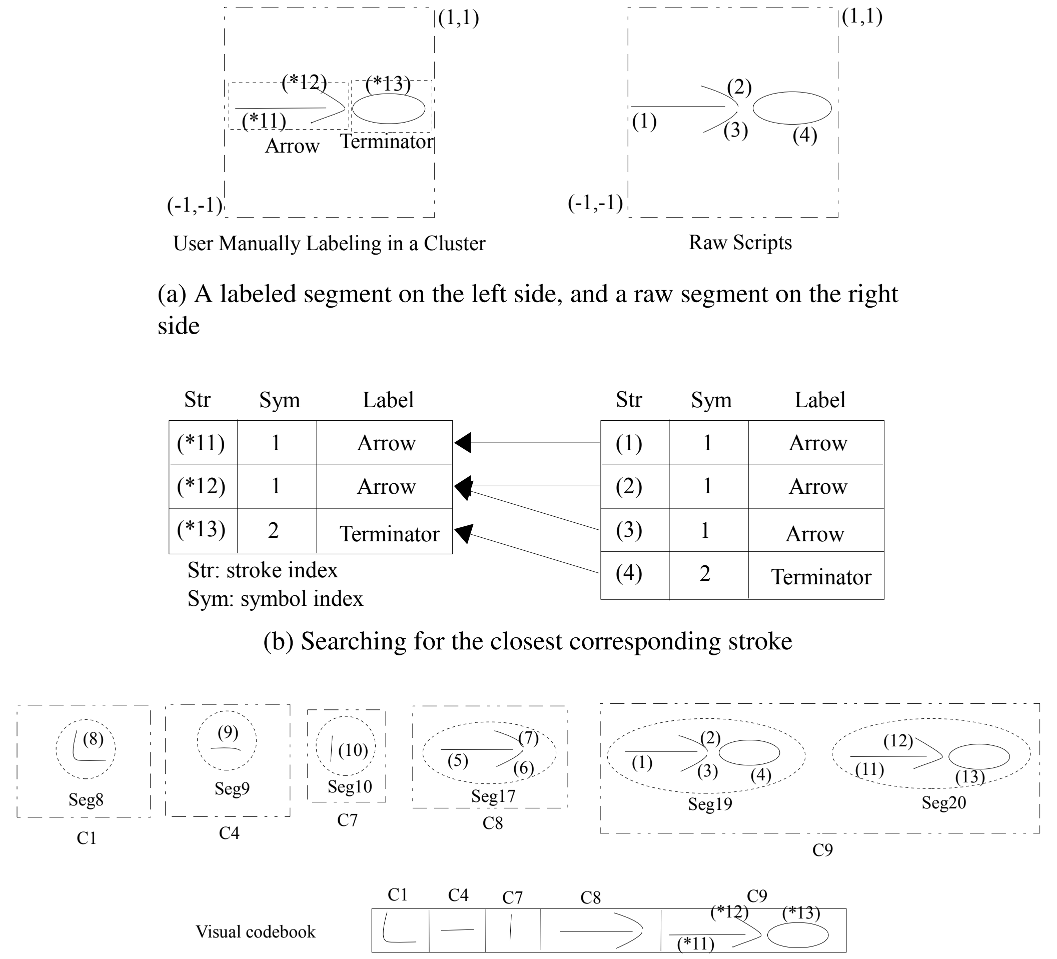
7.4 Experiments

In this section, we first show the labeling cost as presented in Section 6.5 using the examples in this chapter. Then, the labeling procedure using different learned codebooks is tested on the two datasets, *Calc* and *FC* as shown in Section 3.6.

7.4.1 Labeling Cost

In the previous section, the visual codebook is manually labeled. To evaluate the system performance, the chosen segments in the visual codebook are automatically labeled according to the available ground-truths (instead of a manual operation). We then execute the mapping procedure described in Section 7.3 to label all other segments. Since the user labels the segments and raw handwritten scripts in dataset stroke by stroke, Section 6.5 defines the labeling cost at the stroke level. If $C_{label} < 1$, the system reduces the human effort of labeling. The lower labeling cost is preferable. In fact, we can consider C_{label} as the percentage of strokes in dataset which still need a manual operation.

As an example, Fig. 7.7c shows that the handwritten flowchart in Fig. 7.1a is segmented and labeled using the codebook in the fourth iteration as shown in Fig. 7.6b. Some segments in the visual codebook are unknown; it is a part of sym-



(a) A labeled segment on the left side, and a raw segment on the right side

Str	Sym	Label
(*11)	1	Arrow
(*12)	1	Arrow
(*13)	2	Terminator

Str: stroke index
Sym: symbol index

Str	Sym	Label
(1)	1	Arrow
(2)	1	Arrow
(3)	1	Arrow
(4)	2	Terminator

(b) Searching for the closest corresponding stroke

(c) The labeled handwritten flowchart derived from Fig. 7.1a using the fourth iteration codebook in Fig. 7.6b

Figure 7.7: System labeling in the fourth iteration

bol. We leave them as unlabeled. The number of strokes N_{db} in this handwritten flowchart is 13, the number of strokes N_c in the fourth iteration codebook is 9, and the number of strokes $N_{correct}$ in well segmented and labeled symbols is 9. The labeling cost is, therefore, $C_{label} = \frac{9+13-9}{13} = 1$. This labeling cost is not interesting due to the symbol segmentation is not good as shown in Fig. 7.7c.

7.4.2 Results

As shown in Section 3.7.2, we have optimized four parameters to get a higher symbol segmentation: n_p (the number of graphemes), SRF(the spatial relation feature set), n_{sr} (the spatial relation prototype number), n_{cstr} (the number of edges from a reference stroke to an argument stroke). Since the labeling cost depends on the quality of symbol segmentation. We will use partly the optimal parameters obtained from Section 3.7.2.

In this experiment, we test only three parameters, *threshold* (equivalent to n_p), n_u, and n_{it}. Using the SUBDUE system, the number of discovered lexical units is denoted by n_u in each iteration of the whole system. n_{it} means the number of iterations of the whole system as shown in Fig. 7.2. Thus, the total number of discovered lexical units is $n_u * (n_{it} - 1)$.

The other parameters are chosen from Section 3.7.2. The same optimization protocol as shown in Section 5.5 is run in this experiment. We first define an initialization configuration which is the optimal configuration obtained in Section 5.5, and then find an optimal configuration by minimizing labeling cost for each parameter.

Parameter Optimization on the *Calc* Dataset

As mentioned in the previous section, we start with the optimal configuration attained in Section 5.5. The optimal configuration is illustrated as:

1. $n_p = 70$ (Corresponding Threshold: 0.576),
2. $SRFT =$"$F8|I$",
3. $n_{sr} = 10$,
4. $n_{cstr} = 4$.

We will find an optimal configuration of *threshold* ,n_u, and n_{it}. As the first attempt, initializing $n_u = 20$ and $n_{it} = 2$, we test different thresholds during the hierarchical clustering. Fig. 7.8 shows labeling costs according to *threshold*$=$ $[0.50, 0.76]$ (around the previous optimal threshold 0.576). The y-axis shows the labeling cost and the x-axis means the different thresholds. A threshold 0.59 reports a minimum labeling cost of 0.423. This threshold is very close to the threshold (0.576) attained in Section 5.5.

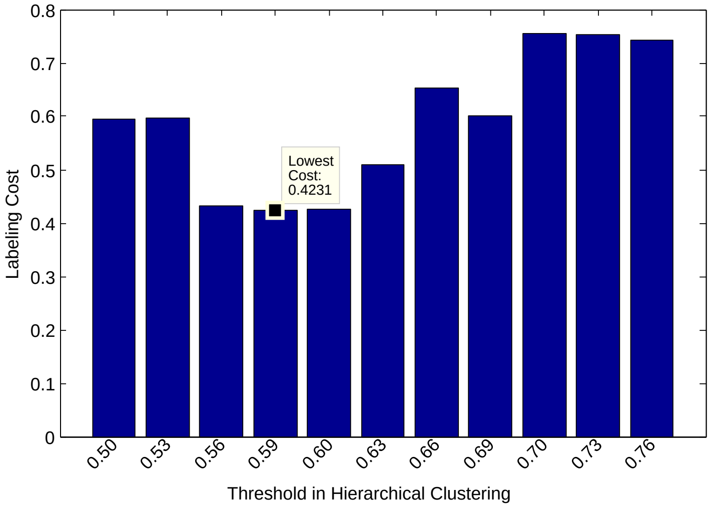


Figure 7.8: Labeling cost with different thresholds during hierarchical clustering (*Calc*, $n_{it} = 2$, $n_u = 20$)

Keeping the $threshold = 0.59$, we test $n_u = [10, 50]$. The x-axis denote the total number of discovered lexical units, $n_u * (n_{it} - 1)$. This number is limited to less than 100. At most 100 symbols will be discovered. Each curve represents each n_u which is the number of discovered lexical units in each iteration. We achieve a minimum labeling cost 42.3% using the same $n_u = 20$ and $n_{it} = 2$ as shown in Fig. 7.9. The labeling cost 42.3% is much lower than the labeling cost 47.4% using a connected-stroke segmentation as studied in Chapter 6. The unsupervised segmentation outperforms the connected-stroke symbol on the training part of *Calc* dataset.

Since the optimal $n_u = 20$ is not changed, we can consider that *threshold*=0.59, $n_u = 20$, and $n_{it} = 2$ are optimal configuration. Using this optimal configuration on the test part of *Calc*, a labeling cost of 0.624 is obtained. This labeling cost is higher than the labeling cost obtained in Fig. 6.6.1. It means that the configuration cannot be automatically best fitted for all the corpus. Some automatic configuration method is still need to be developed.

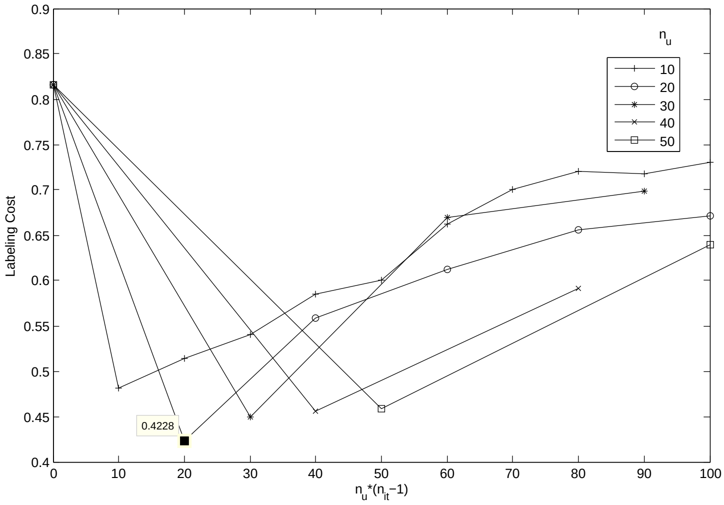


Figure 7.9: Labeling cost with different n_u (*Calc*, threshold=0.59)

In the next section, we run the same experiment protocol on the training part of *FC* dataset in order to get the optimal configuration. The obtained configuration will be assessed on the test part of *FC* dataset.

Parameter Optimization on the *FC* Dataset

To obtain the optimal configuration for the *FC* dataset, we inherit the optimal configuration obtained in Section 5.5:

1. $n_p = 70$ (Corresponding Threshold: 0.53),
2. $SRFT =$"$F8$",
3. $n_{sr} = 30$,
4. $n_{cstr} = 3$.

Based on the optimal configuration, we try to find the optimal configuration of three parameters: *threshold*, n_u, and n_{it}. Similarly, we attempt to compute the labeling cost for different thresholds in the dendrogram with an initialization configuration: $n_u = 20$ and $n_{it} = 2$. The threshold values are located in the range of $[0.50, 0.76]$. The corresponding labeling costs are shown in Fig. 7.10. The minimum labeling cost of 88.6% is reported using the $threshold = 0.53$. This is exactly the same with the $threshold$ obtained in Section 5.5. It means the segmentation quality is vital to decrease the labeling cost.

Keeping *threshold*$=$ 0.53, we test different n_u and n_{it} in Fig. 7.11. We can find a big labeling cost decrease when $n_u = 30$ and $n_{it} = 3$. Our method succeed

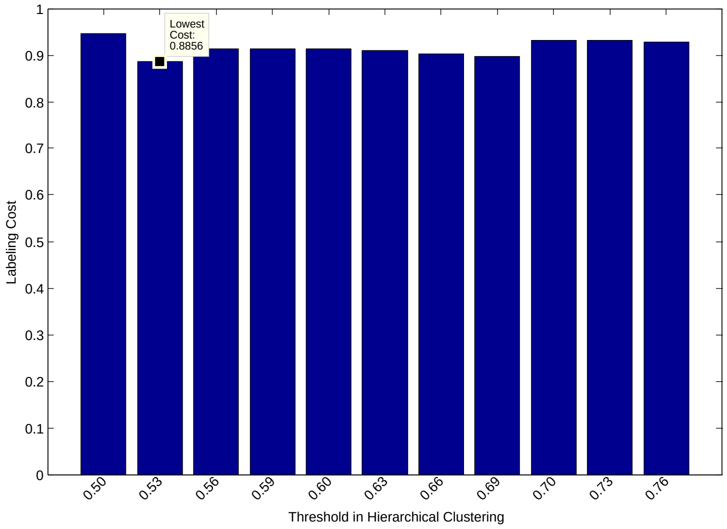


Figure 7.10: Labeling cost with different thresholds during hierarchical clustering (FC, $n_u = 20$, $n_{it} = 2$)

to produce many correct symbol segments so that the labeling cost has reduced to 74.1%. 26.9% annotation workload have been saved. Since n_u and n_{it} have been changed, we continue to test the *threshold*. It may can reduce the labeling cost again.

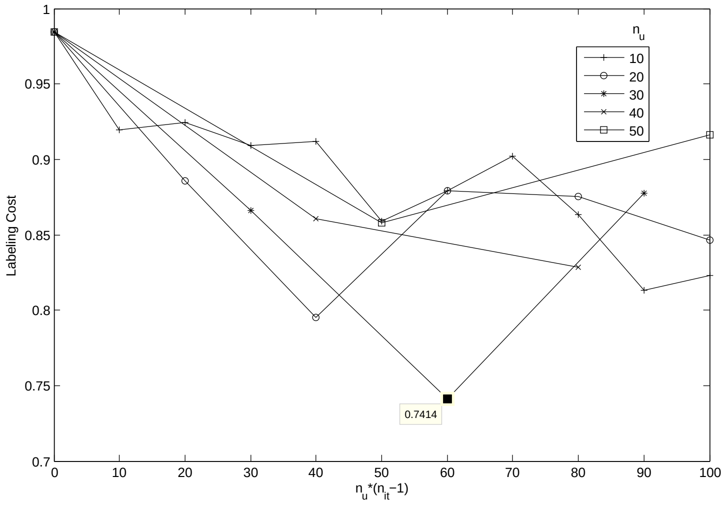


Figure 7.11: Labeling cost with different n_u (FC, *threshold*=0.53)

Using $n_u = 30$ and $n_{it} = 3$, Fig. 7.12 shows labeling cost according to different thresholds that control the codebook size in each iteration. In this figure, 0.53 is still the best threshold. Since the three parameters become stable, we can consider the optimal configuration as: $threshold = 0.53$, $n_u = 30$, and $n_{it} = 3$.

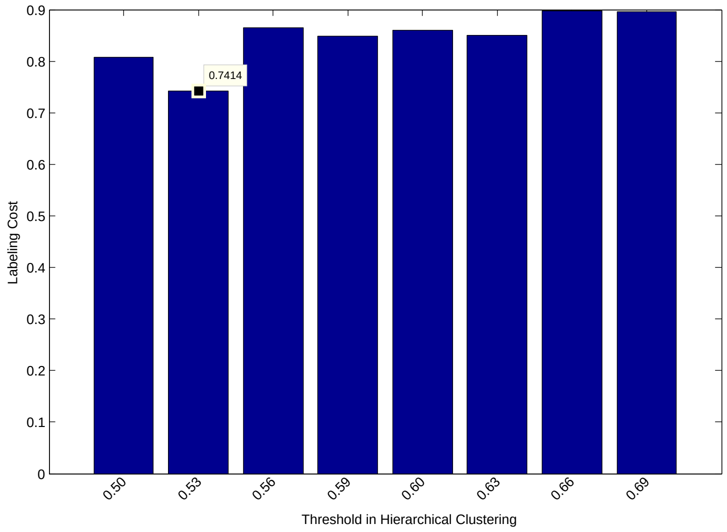


Figure 7.12: Labeling cost with different thresholds during hierarchical clustering (FC, $n_u = 30$, $n_{it} = 3$)

The labeling cost of 0.741 is attained on the training part of *FC*. Comparing the lowest labeling cost 0.94 using connected-stroke segmentation as shown in Chapter 6, our labeling cost 0.741 is much lower and better. It means that our proposed method can find many correct symbol segments in the codebook so that the lower labeling cost is revealed.

On the test part of *FC* Corpus, we use the optimal configuration to compute labeling cost. We achieve a labeling cost 0.878, which is also lower than the labeling cost 0.972 as shown in Section 6.6.4. However, the labeling cost 0.878 is much higher than a labeling cost 0.741 on the training part. We still need a method that automatically optimizes system parameters on a dataset.

7.5 Conclusion

In the previous chapter, we proposed a system to reduce the symbol labeling cost with the codebook mapping. This method is relied on the symbol segmentation quality. In order to generate the symbol segmentation, the previous chapter uses the connected-stroke symbol segmentation to reduce the labeling cost. However, it works very bad on the challenging *FC* dataset. This chapter proposed to combine the symbol extraction using the MDL principle as studied in Chapter 5. The system extracts n_u symbols in each iteration so that the new discovered symbols become the new segments. Therefore, an unsupervised symbol segmentation can be produced. Producing the segment codebook from this segmentation, we can reduce the labeling cost via the codebook mapping.

Our approach reports the lower labeling cost of 42.3% on the training part of *Calc* dataset. It means that our work can ease the human workload largely. This mathematical expression dataset contains 54.9% single-stroke symbols. The single-stroke codebook reports the labeling cost 81.6% in 1^{st} iteration. Our multi-stroke symbol discovery procedure can reduce the labeling cost by 39.3%. The result is very attractive and interesting on this database. Nevertheless, the labeling cost is higher on the more challenging FC database which contains more multi-stroke symbols: 79.7% symbols are multi-stroke on the training part. The crucial problem is how to find out a better symbol segmentation. Indeed our experiments show that many clusters contain the most frequent combinations of sub-parts of symbol on complex database. To avoid such nonsense combinations, we have to study the more precise graphical symbol detection criteria based on the MDL principle.

Moreover, one critical point of our approach is the choice of *threshold*, the number of extracted lexical units (n_u) and the number of iteration n_{it}. For this moment these parameters are fixed during the training phase which needs ground-truthed data; it would be more interesting to be able to tune them at each iteration separately using only current unlabeled data.

8

Conclusions

The creation of training dataset at the symbol level is a tedious work. In this thesis, we proposed an iterative framework to automatically extract graphical symbols using the MDL (Minimum Description Length) principle. The proposed framework can reduce isolated symbol labeling workload. The framework contains three main phases: (a) quantifying the graphical symbols, (b) quantifying the spatial relations, and (c) discovering the graphical symbols using the MDL principle.

We propose to model a graphical language as relational graphs between strokes (or symbols). The nodes are defined as the strokes and the edges are defined as the spatial relations. We try to quantify the nodes and the edges so that we can extract frequent sub-graphs as the graphical symbols using the MDL principle. The strokes (nodes) and the spatial relations (edges) can be quantified via clustering.

(a) To quantify the graphical symbols, we have chosen the hierarchical clustering, which requires the distance between two graphical symbols. We have investigated three distances: the classical Dynamic Time Warping (DTW) distance, the proposed DTW A* distance, and the Modified Hausdorff Distance (MHD). The classical DTW distance is a famous distance to compare two sequences by keeping the continuity constraint. It is good to match two single-stroke symbols which are two sequences. When matching two multi-stroke symbols, we proposed a new distance DTW A*, which keeps the continuity constraint of stroke-to-stroke warping and minimize the sum of associated pair distance. Nevertheless, the DTW A* distance is too slow to run in clustering. We propose to use the MHD distance to compare two graphical symbols. The MHD distance uses the average distance (instead of the maximum distance) of point-to-set minimum distances to avoid the effect of outliers.

To assess the clustering quality, we use two criterion, *Purity* and Normalized Mutual Information (NMI), to compare the *Classical DTW* distance and the MHD distance on two datasets: synthetic single-line mathematical expressions (*Calc*) and real more general two-dimension flowcharts (*FC*). We found that the *Classical DTW* distance slightly outperforms the MHD distance on the *Calc* dataset. Nevertheless, on the *FC* dataset, the MHD distance works much more better than the *Classical DTW* distance. In fact, more strokes per symbol exist on the *FC* dataset so that the stroke order becomes an important problem. The MHD distance does not consider the stroke order during the symbol comparison. The MHD distance therefore works much better on the *FC* dataset.

(b) To quantify the spatial relations, we first model them at three levels: distance relations, orientation relations, and topological relations. We define a pairwise spatial relation as a relationship from a reference symbol to an argument symbol. The distance relation means how far apart two objects are. The orientation relation illustrates some directional information, e.g. west, east, south, north, etc. In our case, we define eight fuzzy spatial relations from the reference symbol to the argument symbol: *left*, *right*, *above*, *below*, *above-left*, *above-right*, *below-left*, and *below-right*. Only an intersection relation is defined among the topological relations. Using the extracted features at the three levels, we can embed the extracted pairwise spatial relations into the fixed-length feature space. The *k-means* clustering is then used to compute the spatial relation prototypes. So the spatial relations can be quantified (classified).

(c) The nodes (symbols) and the edges (spatial relations) have been quantified in the relational graphs. We discover the sub-graphs as the multi-stroke symbols using the sub-graph mining SUBDUE (SUBstructure Discovery Using Examples) system based on the MDL principle.

During the symbol extraction, a symbol segmentation based on the MDL principle will be produced. In addition, we have mentioned two other symbol segmentation methods. The first is the ground-truth segmentation where each segment is composed of only one symbol. The second is the connected-stroke segmentation often used in recognition systems. To compare the three segmentation quality, we have proposed the recall rate at the multi-stroke symbol level R_{MRecall}, which evaluates how many multi-stroke symbols are found in the segmentation.

After these three possible segmentation steps, the same labeling process is used: unsupervised clustering to build the codebook which is then labeled by the user and map to the raw data. The proposed iterative learning framework will update the spatial relations and the codebook size in each iteration. We call this framework as "MDL+iteration". To assess how much symbol labeling work has been saved, we have proposed the labeling cost criteria.

Tab. 8.1 compares the performances between the three symbol segmentation methods considering the two measures on the two handwriting datasets, *Calc* and *FC*. It contains two parts, $R_{MRecall}$ and the labeling cost, according to the three segmentation methods respectively.

As shown in the first part of $R_{MRecall}$, the ground-truth segmentation certainly give the perfect recall rate 100% since each segment contains exactly one symbol. The connected-stroke segmentation method contributes the recall rate of roughly 44% on the training and test parts of *Calc* dataset. Almost half symbols have been found. On the more complex *FC* dataset, the recall rates, only 14.6% and 17.4%, have been reported on the two parts respectively. Not too many symbols have been found. The main reasons are that the symbols are separated and most symbols contain connected strokes on the *Calc* dataset, but on the *FC* dataset the symbols are often touching and some symbols are compound of untouched strokes.

The proposed segmentation method based on the MDL principle can largely improve the recall rates at the multi-stroke symbol level. This method found 78% multi-stroke symbols on the training and test parts of *Calc* dataset. We improve $R_{MRecall}$ by 34% on the *Calc* dataset. On the more challenging *FC* dataset, we can achieve the recall rates 55.5% and 45% on the two parts respectively. Approximately half multi-stroke symbols have been found.

Considering the second part of results (the labeling cost) as shown in Tab. 8.1, the symbol segmentation using the ground-truth shows certainly the lowest labeling

	Calc		FC	
Segmentation	Training	Test	Training	Test
	$R_{MRecall}$ (Multi-Stroke Symbols)			
Ground-truth	100%	100%	100%	100%
Connected-stroke	44.1%	44.7%	14.6%	17.4%
MDL (Alone)	78%	78%	55.5%	45%
	Labeling Cost			
Ground-truth	6.4%	13.1%	4.3%	13.5%
Connected-stroke	46.9%	50.4%	97.5%	97.2%
MDL+Iteration	42.8%	62.4%	74.1%	87.8%

Table 8.1: Result Summarization on the Two Datasets

cost, but in a real case, we do not know the real ground-truth. It shows the best possible labeling cost with the used clustering algorithm. The usage of connected strokes increases a lot the labeling cost. We can also observe this method work better on the *Calc* dataset than on the *FC* dataset. The reason comes from the big difference in the segmentation recall rates.

Lets us now consider the iterative learning framework as presented in Chapter 7 using the MDL principle. It obtains better results on the two training parts compared to the test parts respectively. The reason is that there are many parameters in our system. The number of iteration n_{it} and the number discovered lexical units n_u are optimized only for the training part, not for the test part. Finally, we can see that our proposed method can reduce more the labeling cost comparing the labeling cost using connected strokes. On the *FC* dataset, the recall rate is low so that there is still large possible progress in two perspectives: discovering the symbols and clustering according to its homogeneity.

Our framework is a good first step in this huge task. Some perspective works could be done to improve the proposed framework. As we can see that the proposed framework is a complex system, which contains many parameters. The first problem is how to reduce the number of parameters. For instance, how to automatically determine the number of iterations n_{it} of the whole system. During the

SUBDUE symbol discovery, we have to determine when the discovery procedure will be stopped. Furthermore, extracting symbol and spatial relation knowledge in the graphical language is a non-trivial task. Although we succeed to extract symbol segmentation, the human is still difficult to do same as the human understanding. More complex knowledge with less description could be explored in the future.

9

Résumé Français

9.1 Introduction

Tout langage graphique comporte d'une part des symboles élémentaires, par exemple un alphabet ou des formes graphiques propres à un langage métier (organigramme, schéma électrique, ...), et d'autre part des règles de composition permettant de donner globalement un sens au document produit. La connaissance des symboles élémentaires et de leurs relations nous permet d'interpréter ces messages manuscrits (tracés). De même les systèmes de reconnaissance ont besoin de ces connaissances symboliques pour leur apprentissage [72–74]. Cette connaissance est disponible sous forme de bases d'apprentissage contenant des documents complètement étiquetés au niveau des leur symboles et de leurs relations. De nombreux systèmes de reconnaissance profitent ainsi d'un large corpus de données réelles permettant l'apprentissage de classifieurs, citons par exemples les K-PPV (*K*-Plus Proche Voisins) [7], ANN (Réseaux de Neurones) [8], SVM (Systèmes à Vastes Marges) [9], HMM (Modèle de Markov Cachés) [10]. Enfin ces corpus permettent de comparer les performances des différents systèmes de reconnaissance existants. Les données étiquetées sont donc utiles pour l'apprentissage, le test et l'évaluation des systèmes de reconnaissance.

La Figure 9.1 montre un modèle classique de reconnaissance de symboles pour des expressions mathématiques. Tout d'abord le classifieur est entraîné en utilisant la base d'apprentissage (la vérité-terrain), ce classifieur est ensuite capable alors reconnaître des symboles graphiques non étiquetés comme ceux à gauche de la Figure 9.1.

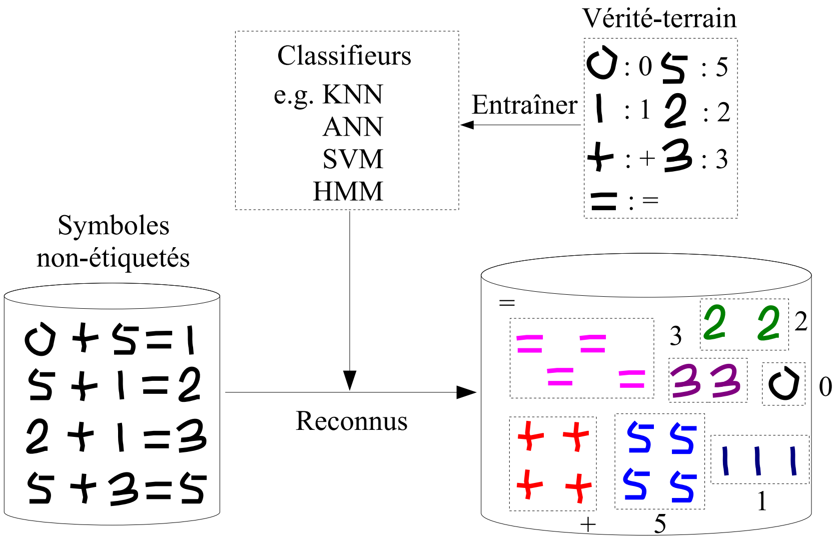


Figure 9.1: Reconnaissance traditionnelle de symboles

Cependant, la collecte d'échantillons d'écritures et l'étiquetage au niveau de chaque trait sont des tâches difficiles et fastidieuses surtout sur un langage graphique inconnu (i.e. sans système de reconnaissance automatique existant). Ce constat a motivé le développement d'un système de plus haut niveau permettant d'automatiser cette procédure d'étiquetage fastidieuse.

Notre approche consiste dans un premier temps à regrouper les symboles en des ensembles homogènes. Ces ensembles de symboles peuvent ensuite être facilement étiquetés ce qui réduit le coût de l'étiquetage symbolique. Sans connaissances symboliques a priori du langage graphique cette tâche nécessite des approches non supervisées permettant de découvrir l'alphabet des symboles. Dans l'exemple de la Figure 9.2 imaginons que les 20 symboles de gauche soient à étiqueter, si nous pouvons regrouper ces 20 symboles en 7 ensembles de symboles, tels que représentés sur la figure de droite, alors il suffira d'effectuer 7 assignations d'étiquettes et de les propager sur les 20 symboles.

Cet apprentissage non supervisée est difficile. Tout d'abord, aucune segmentation de l'écriture en symboles n'est disponible. Ces travaux sont appliqués à de

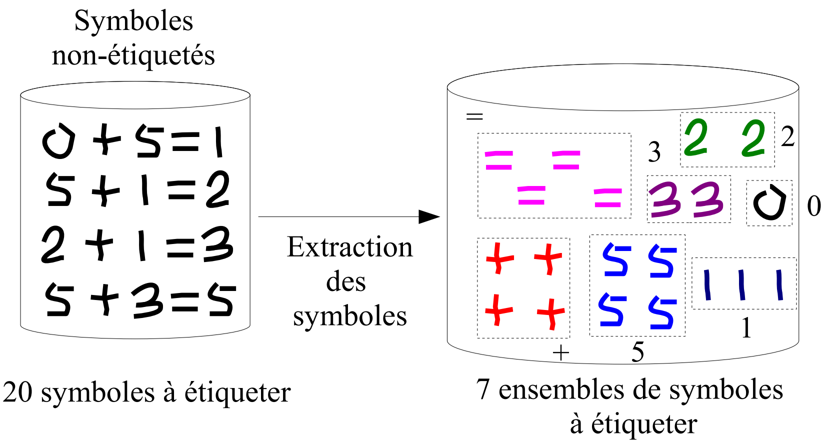


Figure 9.2: Réduction du travail d'étiquetage symbolique

l'écriture manuscrite en ligne, l'élément de base de l'écriture est donc le trait, qui correspond à une séquence de points (x, y) entre un poser et un lever de stylo. Nous supposerons ici qu'un trait n'appartient qu'à un seul symbole, si ce n'est pas le cas, il conviendrait de pratiquer une étape préalable de segmentation. Par contre, un symbole peut comporter plusieurs traits n'étant pas nécessairement écrits consécutivement.

L'exemple de la Figure 9.3 montre qu'un trait horizontal peut représenter à lui seul un symbole (moins, "$-$"), ou être composé avec d'autres traits (plus, "$+$") ou encore avec lui-même (égal, "$=$"). Cet exemple montre que détecter les différentes formes de trait n'est pas suffisant : la difficulté réside dans la recherche des combinaisons de traits qui sont des symboles. En d'autres termes, nous avons besoin d'une méthode non supervisée permettant de trouver une segmentation correspondant à une segmentation en symboles. Notre approche se base sur l'hypothèse que les symboles graphiques sont des combinaisons fréquentes de traits. Par exemple dans la Figure 9.2, les combinaisons "$=$", "$+$", "5" se répètent quatre fois, et "1" est répété trois fois. En comparant un motif fréquent "$+1$" (trois traits répétés trois fois) et une sous partie fréquente "$+$" (deux traits répétés quatre fois), ces deux hypothèses de segmentation peuvent être un symbole. La question se pose alors de savoir comment choisir les bonnes combinaisons de traits pour former les symboles. Dans cette thèse, nous introduisons un critère, le principe de longueur minimum de description (Minimum Description Length, MDL) [11], pour permettre d'extraire automatiquement les meilleurs représentants du lexique des symboles.

La fréquence d'une hypothèse de symbole dépend du nombre d'occurrences

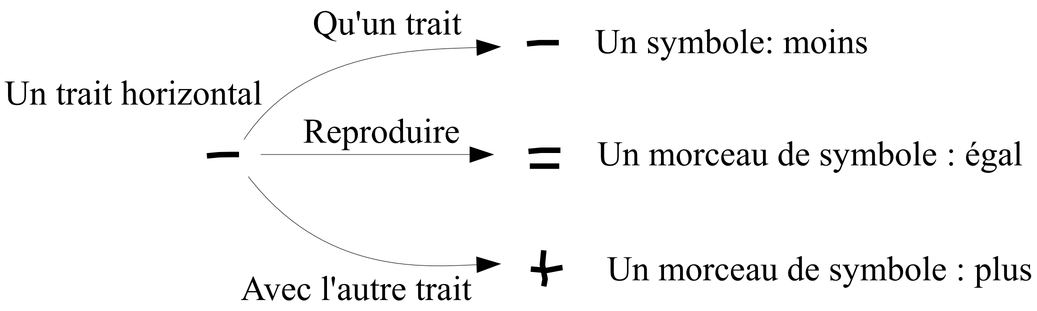


Figure 9.3: Un trait peut être un symbole ou une partie de symbole

de cette hypothèse. Pour être capable de compter (i.e. chercher) rapidement les différentes instances d'un même symbole mono- ou multi-traits, nous proposons d'organiser le langage bidimensionnel en un graphe relationnel de traits. Le problème du décompte (ou de la recherche) d'une hypothèse de symbole devient alors un problème de recherche de sous-graphes.

Par exemple, les deux premières expressions mathématiques de la Figure 9.2 sont représentées par les deux graphes de la Figure 9.4. Nous pouvons y voir les symboles représentés par les sous-graphes encadrés. Pour éviter toute ambiguïté entre les traits se ressemblant fortement (appartenant donc au même graphème), chaque trait est indexé par son numéro d'indice (.). Nous pouvons constater que pour produire ce type de graphes, il faut être capable de définir ou apprendre les relations liant ces traits (e.g. *Right*, *Intersection*, *Below*, etc.) appelées “relations spatiales”. Une fois ces relations spatiales apprises et appliquées entre les traits, nous pouvons voir que les symboles multi-traits “+”, “5”, and “=” existent et sont définis par un sous-graphe de traits reliés par des relations. L'objectif est de rechercher automatiquement ces symboles multi-trait dans les graphes en utilisant un critère MDL.

Supposons maintenant que grâce à ce critère de découverte des symboles, nous obtenions une segmentation correcte des traits (comme dans la partie gauche de la Figure 9.5). Il s'agit maintenant de trier ces hypothèses de symboles en fonction de leur formes : regrouper les symboles qui se ressemblent fortement. Ce regroupement non supervisé (*clustering* en anglais) nécessite de définir une distance entre deux symboles graphiques multi-traits. En fonction du nombre de traits composant les symboles, nous pouvons diviser les types de distances en deux catégories : une distances entre deux symboles mono-trait (le cas simple) ou une distance entre deux

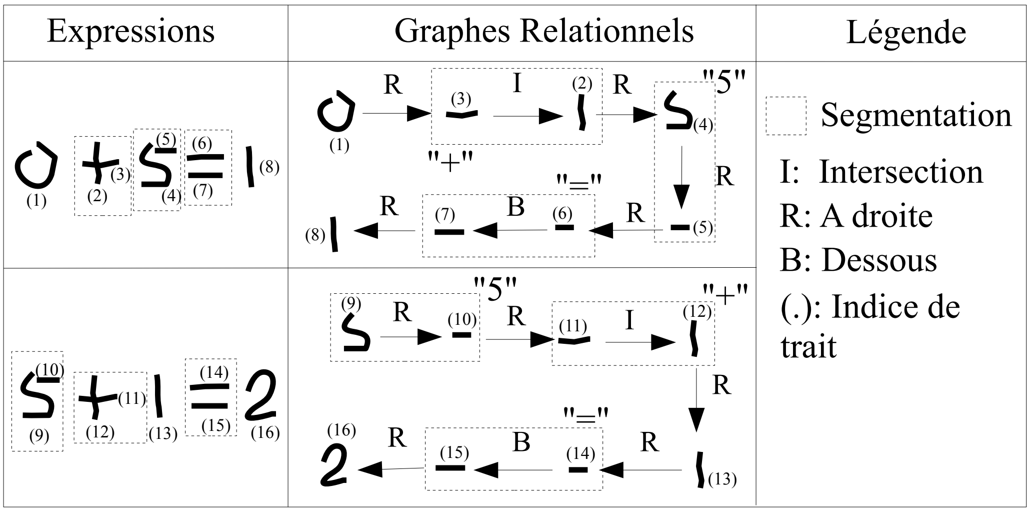


Figure 9.4: Expressions mathématiques et leur graphe relationnel correspondant.

symboles multi-traits (cas plus complexe).

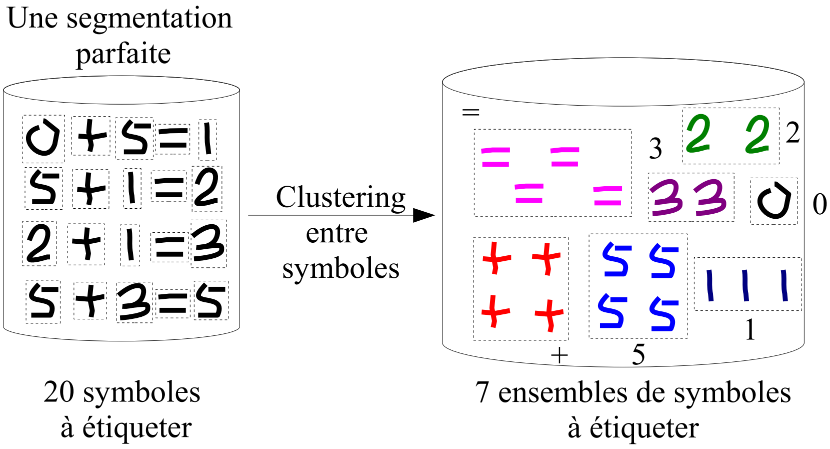

Une segmentation parfaite
Clustering entre symboles
=
3
2
0
1
+
5
20 symboles à étiqueter
7 ensembles de symboles à étiqueter

Figure 9.5: Les symboles correctement segmentés sont regroupés en ensembles homogènes.

Deux symboles mono-trait peuvent être comparés par la distance DTW (Dynamic Time Warping) [12] très utilisée dans les domaines traitant un signal temporel (de l'écriture en-ligne ou un signal audio par exemple). Par contre la distance entre deux symboles multi-trait est plus complexe car deux scripteurs peuvent écrire un même symbole de différentes façons : en changeant le nombre de traits, l'ordre des traits et leur sens d'écriture. Nous préférerons une distance indépendante du nombre de traits, de leur ordre et de leur direction.

Par exemple considérons les symboles "+" de la Figure 9.6, ils peuvent être écrits de quatre façons différentes. Le nombre de possibilités augmente rapidement avec le nombre de traits d'un symbole. Dans la plupart des systèmes traitant de l'écriture en-ligne, les traits d'un symbole sont concaténés en conservant leur ordre naturel d'écriture. Ainsi la métrique DTW peut être directement utilisée. Néanmoins, la distance entre deux symboles visiblement identiques mais avec des ordres ou directions de traits différents sera très importante. Nous discuterons de ce problème et proposerons des solutions dans la Section 9.3.

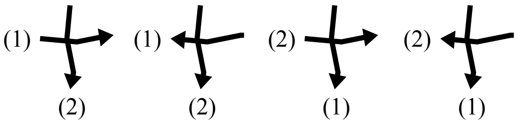


Figure 9.6: Quatre écritures possibles du symbole "+"

Une fois la distance inter-symboles définie, nous utilisons une technique de regroupement non supervisée pour créer des ensembles homogènes de symboles. Pour chaque ensemble nous choisissons un exemple représentatif du groupe. Ces représentants sont ensuite regroupés dans un dictionnaire visuel. Ce dictionnaire pourrait alors être affiché dans une interface adaptée pour permettre un étiquetage manuel de chaque ensemble de symboles. La limite de cette approche est qu'il faut que les regroupements effectués ne contiennent que des symboles de la même classe. Sinon, l'utilisateur aura à corriger un certain nombre d'erreurs ce qui augmente le coût de l'étiquetage. De plus les segmentations proposées ne seront pas toujours parfaites. Par exemple la Figure 9.7 montre que la séquence "+1" est considérée comme un symbole. En fait ce segment est composé d'instances de deux symboles distincts. Le dictionnaire visuel de la Figure 9.8 permet à l'utilisateur d'étiqueter facilement les regroupements et de détecter ce problème. Il pourra alors séparer l'hypothèse "+1" en deux instances de symboles isolés "+" et "1". Une fois chaque représentant du dictionnaire étiqueté, il faut transférer les étiquettes de chaque trait à toutes les données de la base. Ce transfert n'est pas toujours trivial lorsque tous les symboles d'un regroupement n'ont pas le même nombre traits.

Dans ces travaux de thèse nous présenterons tout d'abord différentes approches de regroupement non supervisé. Pour implémenter ces approches, les distances entre symboles graphiques seront discutées et une approche originale sera présentée. Ensuite nous proposerons de modéliser un langage graphique par un graphe relationnel où les noeuds sont des traits et les arcs des relations spatiales. Nous

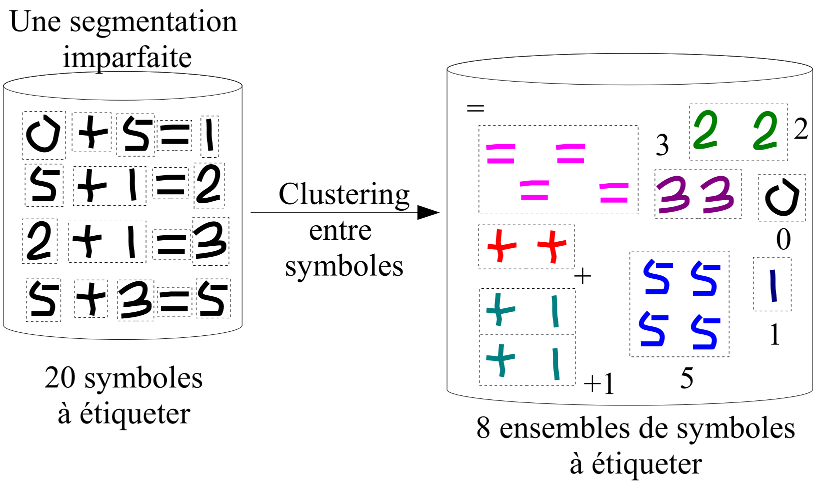

Une segmentation imparfaite
20 symboles à étiqueter
Clustering entre symboles
=
3
2
0
+
1
+1
5
8 ensembles de symboles à étiqueter

Figure 9.7: Exemple de segmentation imparfaite d'un symbole.

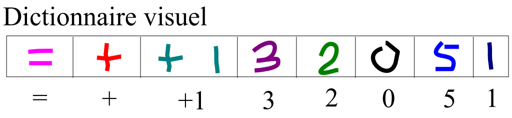


Figure 9.8: Dictionnaire visuel pour l'étiquetage manuel.

proposons alors d'utiliser une approche d'extraction de sous-graphes basée sur le principe de la longueur minimum de description (MDL) pour détecter les symboles fréquents. Enfin nous présenterons des résultats expérimentaux d'étiquetage automatique de deux bases d'écritures manuscrites en-ligne.

9.2 Techniques de Clustering

Il existe plusieurs méthodes de clustering dans l'état de l'art : *k-moyennes* [39], Carte Auto Adaptative (Self-Organizing Map, SOM) [40], Neural Gas (NG) [41], Growing Neural Gas (GNG), clustering hiérarchique [42], etc. L'algorithme le plus connu et le plus utilisé, l'algorithme des *k-moyennes* consiste à rechercher k vecteurs moyens (prototypes) divisant les n échantillons en k clusters. Chaque échantillon est donc assigné au groupement (cluster) le plus proche considérant une certaine distance. L'espace des échantillons est donc divisé en k cellules de Voronoi (Voronoi cells). Néanmoins, les k prototypes sont indépendants.

Les algorithmes SOM, NG, et GNG fonctionnent de la même manière mais ajoutent des relations de voisinage entre les prototypes. SOM impose une topologie (généralement en deux dimensions) à ce réseau de prototypes. Si nous visualisons les prototypes de ce réseau, nous obtenons une carte auto adaptative en deux dimensions des échantillons. L'algorithme NG utilise quant à lui une structure plus flexible entre les prototypes. En effet les relations de voisinage des prototypes peuvent évoluer pendant l'apprentissage. L'algorithme GNG peut ajouter ou supprimer des prototypes (donc des voisins) pendant l'apprentissage.

Contrairement aux autres approches, le clustering hiérarchique de type ascendant fusionne à chaque étape de l'apprentissage les deux prototypes les plus proches pour générer un dendrogramme. Cette structure hiérarchique permet en fin d'apprentissage de choisir facilement le nombre de prototypes finaux.

Dans cette thèse, nous utiliserons seulement les algorithmes des *k-moyennes* et le clustering hiérarchique, mais d'autres approches peuvent sans problème les remplacer. Dans la section suivante, nous discuterons du choix de la métrique utilisée pour évaluer la distance entre deux symboles multi-traits dans les algorithmes de clustering.

9.3 Distance Entre Deux Symboles Multi-Traits

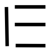
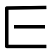
Le processus de reconnaissance d'écritures en-ligne peut être divisé en deux étapes principales, la segmentation de symboles et la reconnaissance des symboles isolés [2]. Dans cette section, nous discutons de la similarité entre deux symboles isolés. Le symbole isolé est composé d'un ensemble de traits. Chaque trait (*stroke* en anglais) est représenté par une séquence de points, du point de départ (posé) au point d'arrivée (levé). Un trait est donc orienté. Calculer la distance entre deux symboles isolés est donc un problème de comparaison de deux ensembles de séquences orientées de points.

Différentes personnes peuvent écrire un même symbole avec des sens différents pour chaque trait et utilisant des ordres différents entre les traits. Dans la recherche d'identification de scripteurs, cette caractéristique peut distinguer efficacement les scripteurs [39]. Néanmoins, pour comprendre ou communiquer avec un même symbole écrit par les personnes différentes, le sens et l'ordre des traits doivent être ignorés. Nous lisons des symboles manuscrits sans avoir accès ni au sens ni à l'ordre. Par exemple, un symbole contenant un trait horizontal"–" peut être écrit de deux manières, de gauche à droite “$\rightarrow$” ou l’inverse “$\leftarrow$”.

Lors de la comparaison de deux symboles chacun écrit d’un seul trait, l’algorithme DTW (Dynamic Time Warping) permet une mise en correspondance point à point en respectant des contraintes de séquentialité pendant l’alignement entre les deux séquences [12]. Cet alignement décrit dans la section suivante respecte la contrainte de continuité temporelle de l’alignement.

Si l’on calcule l’alignement de deux traits avec deux directions opposées, la distance DTW $dist_{DTW}(\rightarrow, \leftarrow)$ produit naturellement une valeur importante du fait de l’inversion du sens de parcours. Une solution consiste à choisir la plus petite distance DTW entre les deux directions possibles d’un trait : $\min(dist_{DTW}(\rightarrow, \leftarrow), dist_{DTW}(inv(\rightarrow), \leftarrow))$ où $inv(.)$ est un opérateur d’inversion de l’ordre de parcours des points d’un trait. Cependant, le nombre de combinaisons augmente très rapidement par rapport au nombre de traits dans un symbole. Le Tableau 9.1 montre la complexité des ordonnancements et des sens de parcours des traits d’un tracé en-ligne. D’une manière générale, le nombre de séquences est donnée par la formule:

$$S = N! \times 2^N. \tag{9.1}$$

Pour calculer la distance DTW entre deux symboles multi-traits, une solution directe consiste à concaténer les différents traits en prenant en compte un certain ordre de tracé. Ainsi pour calculer la distance entre  (4 traits) et  (2 traits) il faut calculer 384 $\times$ 8 = 3092 appariements possibles. Ce grand nombre d’appariements est dû à deux causes complémentaires : l’ordonnancement respectif des traits (en nombre N!) et le sens de parcours de chaque trait (pour chaque ordonnancement, 2^N variantes de parcours).

Pour calculer la distance entre deux symboles multi-traits une autre approche consiste à considérer ces deux ensembles de séquences de points comme deux ensembles de points, en ignorant l’information temporelle. Nous pouvons alors utiliser une métrique du domaine du traitement d’images, comme par exemple la distance de Hausdorff [35] qui permet de mesurer l’éloignement de deux ensembles de séquences de points comme deux ensembles de points, $S_1 = \{p_1(i)\}$ et $S_2 = \{p_2(j)\}$:

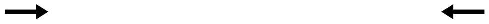
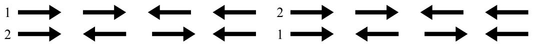
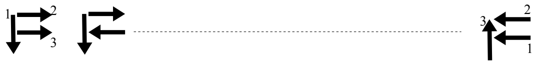
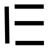
Nombre de traits(N)	Exemple	Nombre de séquences(S)	Illustration des tracés
1		2	
2		8	
3		48	
4		384	…………………………

Table 9.1: Variabilité des ordonnancements et des sens de parcours des traits dans un tracé en-ligne

$$d_H(S_1, S_2) = \max\{d_h(S_1, S_2), d_h(S_2, S_1)\}, \tag{9.2}$$

où $d_h(S_A, S_B) = \max\limits_{p_A \in S_A} \min\limits_{p_B \in S_B} d(p_A, p_B)$ et $d(p_A, p_B)$ est de la distance euclidienne entre deux points. Mais cette distance ne vérifie pas la contrainte de continuité temporelle intra-séquence.

De nombreux travaux [50–52] étendent cette contrainte de continuité d'une dimension (time warping) aux deux dimensions spatiales (two-dimensional warping). Ils mettent en correspondance deux ensembles de points (pixels) avec la contrainte de continuité spatiale. Mais la continuité de séquences n'est pas considérée.

Dans cette thèse, nous discutons de l'alignement entre deux ensembles de séquences de points avec la contrainte de continuité intra-séquence. Le meilleur alignement doit être trouvé à partir d'un grand nombre possible d'appariements. Une procédure de recherche directe serait très lente à cause des nombreuses possibilités. Pour estimer rapidement le meilleur alignement, l'algorithme de recherche A* (A étoile) [57] permettant de réduire le nombre d'appariements est utilisé. Cet algorithme est basé sur une évaluation heuristique du coût de chaque appariement. Pour optimiser l'algorithme A*, nous proposons aussi une stratégie particulière de parcours du graphe des possibilités d'alignement.

Dans la suite de cette thèse nous introduisons la problématique de la distance entre deux ensembles de séquences. Puis l'algorithme A* et notre stratégie de recherche sont présentés. Dans la dernière section de cette thèse nous présentons des résultats qualitatifs et concluons notre travail.

9.3.1 Définition de la Problématique

Nous commençons ici par présenter la version classique de l'alignement de deux séquences de points en utilisant DTW. Puis nous présentons la problématique de l'extension de l'algorithme DTW à un ensemble de séquences.

DTW entre Deux Séquences de Points

Notre objectif est de comparer deux symboles isolés. Nous commençons pas présenter le cas simple où les deux symboles contiennent chacun un seul trait. Deux traits $S_1 = (p_1(1), ..., p_1(N_1))$ et $S_2 = (p_2(1), ..., p_2(N_2))$ seront donc comparés. L'algorithme DTW (Dynamic Time Warping) permet de calculer la distance entre deux séquences de données qui varient temporellement. Cette méthode a d'abord été appliquée dans le domaine du traitement de la parole afin de mettre en correspondance des échantillons acoustiques. Comme pour la parole, les données manuscrites en-ligne contiennent une information temporelle. Bien que coûteuse en temps de calcul, cette distance a déjà montré son efficacité dans plusieurs travaux [12, 53, 54].

Les grands principes de l'algorithme DTW [12] sont résumés ci-après. Soit le chemin $P(h) = (i(h), j(h)), 1 \leqslant h \leqslant H$ permettant de décrire l'alignement point à point où h est de l'indice d'ordre d'appariement du $i(h)$ème point et du $j(h)$ème point des traits S_1 et S_2 respectivement.

$P(h)$ doit respecter la contrainte de frontière et la contrainte de continuité. La première contrainte de frontière est définie par :

$$\begin{aligned} P(1) &= (i(1), j(1)) = (1, 1), \\ P(H) &= P(i(H), j(H)) = (N_1, N_2). \end{aligned} \tag{9.3}$$

Les deux points de départ et les deux points de fin doivent être appareillés respectivement dans les deux traits. L'Équation 9.4 ci-dessous explicite la seconde contrainte, la contrainte de continuité temporelle de l'alignement :

$$(\triangle\, i(h), \triangle\, j(h)) = (i(h) - i(h-1), j(h) - j(h-1)) = \begin{cases} (1,0) ou \\ (0,1) ou \\ (1,1). \end{cases} \tag{9.4}$$

Cette relation montre que le décalage entre paires de points dans l'appariement est au maximum de 1. De plus tous les points sont utilisés au moins une fois. Calculer la distance entre deux séquences consiste donc à chercher parmi tous les appariements possibles celui minimisant la somme des distances point à point :

$$D(S_1, S_2) = \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))), \tag{9.5}$$

où $d(.,.)$ est en général la distance euclidienne entre deux points. La solution à l'Équation 9.5 peut être résolue récursivement par programmation dynamique en définissant la matrice $D(i, j; h)$ des distances cumulées :

$$D(i, j; h) = d(p_1(i), p_2(j)) + \min \begin{cases} D(i-1, j; h-1) \\ D(i, j-1; h-1) \\ D(i-1, j-1; h-1), \end{cases} \tag{9.6}$$

avec $D(i, j; 0) = 0$ pour l'initialisation.

La Figure 9.9 donne un exemple de deux séquences de points à comparer. Les points de départ des traits sont représentés par les ronds rouges. Nous cherchons le chemin avec le coût minimum avec l'Équation 9.6. Nous calculons d'abord la matrice d'accumulation montrée dans la Figure 9.10. Le meilleur chemin (i.e. alignement) peut être déduit par retour arrière à partir du couple de points terminal : P(1),...,P(8) = (1,1),(2,2),(3,3),(4,4),(5,5),(6,6),(7,6),(8,6). Si nous définissons un couple de points de départ et un couple de points de fin, le meilleur alignement sera trouvé. Dans la section suivante, nous introduirons la comparaison entre deux ensembles de séquences.

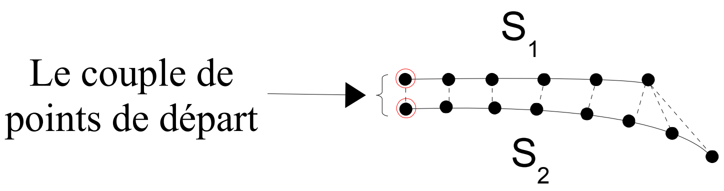
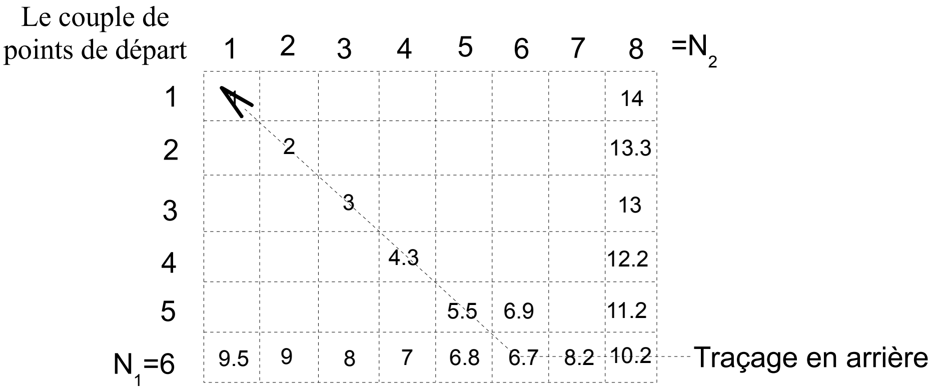


Figure 9.9: Deux séquences de points (deux traits)



	1	2	3	4	5	6	7	8
1								14
2		2						13.3
3			3					13
4				4.3				12.2
5					5.5	6.9		11.2
N_1=6	9.5	9	8	7	6.8	6.7	8.2	10.2

Figure 9.10: Représentation de la matrice d'accumulation $D(i, j; h)$ de l'Équation 9.6 et d'un chemin de mise en correspondance.

DTW avec Deux Ensembles de Séquences

Cette section décrit l'alignement entre deux ensembles de séquences de points. Nous utilisons pour illustrer notre explication la matrice des distances point à point. Soit deux symboles contenant un certain nombre de traits, les côtés de cette matrice représentent les traits des deux symboles respectivement en ligne et en colonne. L'ordre des traits dans la matrice n'est pas obligatoirement l'ordre du tracé effectif.

Le principe de notre approche consiste à construire itérativement un alignement par partie jusqu'à utilisation de tous les points. Si nous choisissons un couple de points de départ, quatre directions d'alignement sont possibles, elles correspondent aux quatre orientations diagonales qui définissent les quatre possibilités d'aligner deux traits. À chaque fois il faut chercher le meilleur chemin consommant au moins un des deux traits. Pour trouver ce meilleur chemin, les quatre matrices d'accumulation (la Figure 9.11) sont calculées pour chacune de ces directions. Les bornes de ces matrices sont définies par la fin de chaque trait.

Par exemple, soit deux symboles contenant respectivement deux et trois traits, les traits de ces deux symboles sont représentés respectivement en ligne et en colonne dans la matrice de la Figure 9.11. Nous pouvons voir qu'à partir d'un couple de points de départ (le rectangle bleu dans la figure) il existe quatre directions possibles d'alignement.

Dans chacune de ces quatre matrices d'accumulation, nous pouvons appliquer l'algorithme DTW classique comme présenté dans la section précédente. Toutefois, nous permettons à DTW de ne pas finir forcément dans l'angle opposé au point de

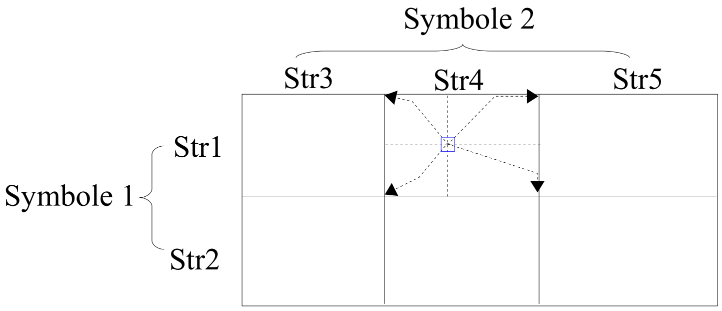


Figure 9.11: Exemple de matrice de distance entre deux symboles avec 2 et 3 traits respectivement.

départ.

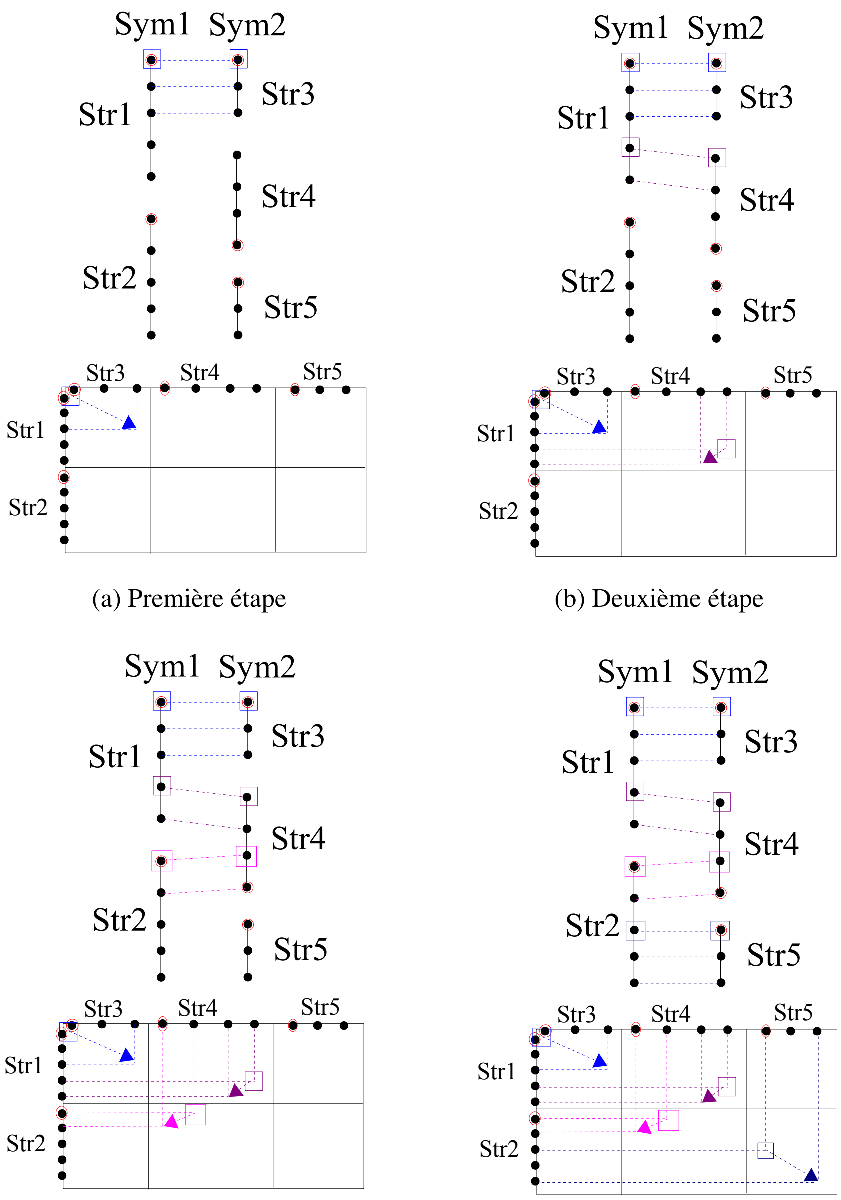
En effet, dans la Figure 9.10 nous pouvons constater que l'alignement se terminant par P(8)=(8,6) n'est pas le meilleur en termes de distances cumulées. Nous pouvons couper l'alignement en choisissant la valeur minimum sur les côtés de la matrice : $D(1, N_2), D(2, N_2), ..., D(N_1, N_2)$ et $D(N_1, 1), D(N_1, 2), ..., D(N_1, N_2)$. Concrètement, nous calculons d'abord la matrice d'accumulation complète jusqu'à la fin des deux traits. Puis l'alignement sera arrêté sur la consommation complète d'un des deux traits. Un nouveau couple de points de fin est donc défini. Par exemple dans la Figure 9.10, nous choisissons le sous-chemin : (1,1), (2,2), (3,3), (4,4), (5,5), (6,6).

Dans cette stratégie, les couples de points de départ sont choisis pour associer les deux sous-séquences en respectant la contrainte de continuité dans chaque étape. À chaque étape nous répétons le choix de couple de points parmi les points non-utilisés. La procédure sera finie lorsque tous les points seront utilisés une fois. Notre objectif est de trouver la somme minimum de coût du chemin d'appariements (Équation 9.5). Enfin la distance entre deux ensembles de séquences, $Sym1 = (p_1(1), ..., p_1(N_1))$ et $Sym2 = (p_2(1), ..., p_2(N_2))$, est normalisée par le nombre d'associations :

$$D(Sym1, Sym2) = \frac{1}{H} \min_{P(h)} \sum_{h=1}^{H} d(p_1(i(h)), p_2(j(h))). \tag{9.7}$$

La Figure 9.12 montre une solution pour associer deux ensembles de séquences, c'est-à-dire un chemin. Cette solution contient quatre sous-chemins de DTW. Les sens d'alignement entre deux traits ne sont pas forcément identiques. Nous allons chercher l'ensemble de sous-chemins qui minimise le coût d'association (somme des distances point à point).

Nous pouvons maintenant comprendre la complexité du problème. En effet, pour commencer chaque sous-chemin il existe un grand nombre de possibilités pour choisir le couple de départ. Á partir de ces point de départ il existe encore beaucoup de chemins potentiels (choix du point d'arrivée). Pour rechercher la meilleure combinaison de sous-chemins, nous choisissons l'algorithme A* [57] qui permet d'accélérer la recherche, celle-ci est décrite dans la section suivante.



(a) Première étape

(b) Deuxième étape

(c) Troisième étape

(d) Quatrième étape

Figure 9.12: Une solution pour associer deux ensembles de séquences (vues matricielles et graphiques)

9.3.2 Algorithme A*

L'algorithme A* recherche itérativement un chemin dans un graphe (ici un arbre n-aire) en partant d'un état de départ (aucun point aligné) à un état d'arrivée (tous les points alignés). Tout l'arbre de recherche n'est pas généré (c'est ce qui fait l'efficacité de A*). A chaque itération, seule la meilleure hypothèse est explorée par génération des états suivants.

Pour accélérer la recherche, l'algorithme de recherche A* utilise une évaluation heuristique $f(x) = g(x) + h(x)$ du coût de chaque étape x : un coût $g(x)$ donnant le coût du meilleur chemin arrivant à x plus $h(x)$ pour estimer le coût permettant d'atteindre le but final. La référence [57] précise cet algorithme. Dans cette section, nous définissons les fonctions $g(x)$ et $h(x)$ adaptées à notre problème ainsi que le choix des points de départ permettant de créer les étapes qui suivront x.

Le Coût de Chemin g(.) et l'Heuristique h(.)

Nous définissons chaque étape x par le chemin $P_x(h) = (i_x(h), j_x(h)), 1 \leqslant h \leqslant H_x$ entre deux symboles $Sym1 = (p_1(1), ..., p_1(N_1))$ et $Sym2 = (p_2(1), ..., p_2(N_2))$. Le coût du chemin est défini par la somme de la distance entre deux points d'appariement :

$$g(x) = \sum_{1}^{H_x} d(p_1(i(h)), p_2(j(h))). \tag{9.8}$$

Concernant l'heuristique $h(.)$, elle correspond au coût nécessaire minimum pour aller de l'étape actuelle x à l'étape du but. L'heuristique $h(.)$ doit être admissible, i.e. $h(.)$ ne surestime jamais la distance à l'étape finale. Pour l'admissibilité, nous définissons l'ensemble des points non-utilisés $NUPt(Sym, x)$ pour un symbole Sym à l'étape x. L'heuristique $h(x)$ est donc définie par

$$h(x) = \frac{1}{2}(h_{sub}(x, Sym1, Sym2) + h_{sub}(x, Sym2, Sym1)), \tag{9.9}$$

où

$$\begin{aligned} h_{sub}(x, SymA, SymB) &= \sum_{p_1(i) \in NUPt(SymA,x)} d(p_1(i), ppv(p_1, SymB)) \\ ppv(p_1, SymB) &= \operatorname*{arg\,min}_{p_2(j) \in NUPt(SymB,x)} d(p_1, p_2(j)). \end{aligned} \tag{9.10}$$

Cette heuristique est admissible car nous choisissons toujours la distance minimum entre tous les points non-utilisés (le coût de l'alignement de ces points sera forcément supérieur).

Le Choix des Points de Départ

Pour générer les étapes suivantes, il faut choisir un couple de points non-utilisés qui permettra de démarrer la mise en correspondance suivant quatre directions au maximum. Pour chaque direction, une nouvelle étape est obtenue. Bien que l'algorithme A* puisse réduire la complexité de recherche, il existera encore de nombreuses possibilités si toutes les combinaisons entre les points non-utilisés sont choisies (cas le plus général). Dans cette section, une stratégie est donc proposée pour limiter la complexité en limitant les possibilités de points de départ.

Nous définissons les *segments* non-utilisés à l'étape x pour chaque trait dans un symbole par $Segs(Sym, x)$ ainsi que les points de frontière de ces segments par $FSeg(Sym, x)$. L'ensemble des nouveaux couples de points $\{(p_i, p_j)\}$ entre deux symboles, $Sym1$ et $Sym2$, sont produits depuis $FSeg(Sym1, x)$ vers les points plus proches dans $Segs(Sym, x)$ et inversement :

$$\begin{aligned}
&\{(p_i, p_j)\} = \\
&\{\forall p_i \in FSeg(Sym1, x), \forall seg \in Segs(Sym2, x), (p_i, ppv(p_i, seg))\} \\
&\qquad\qquad\qquad\qquad\qquad\qquad \cup \\
&\{\forall p_j \in FSeg(Sym2, x), \forall seg \in Segs(Sym1, x), (ppv(p_j, seg), p_j)\}
\end{aligned} \tag{9.11}$$

La Figure 9.13 montre les couples de départ possibles à partir de l'alignement de la Figure 9.12b. Il existe six couples de points qui peuvent s'étendre dans quatre directions respectivement. Toutes ces possibilités sont exploitées par l'algorithme de recherche A* pour trouver le meilleur alignement complet.

Notons que le choix des points de départ (mais aussi des points d'arrivée) agit sur deux propriétés de DTW A* :

- la qualité de la solution finale : si certaines possibilités sont trop limitées, elles ne pourront pas apparaître dans les alignements finaux,
- la rapidité d'exécution : moins il y a de possibilités, moins il y a de branches à évaluer et plus la recherche est rapide.

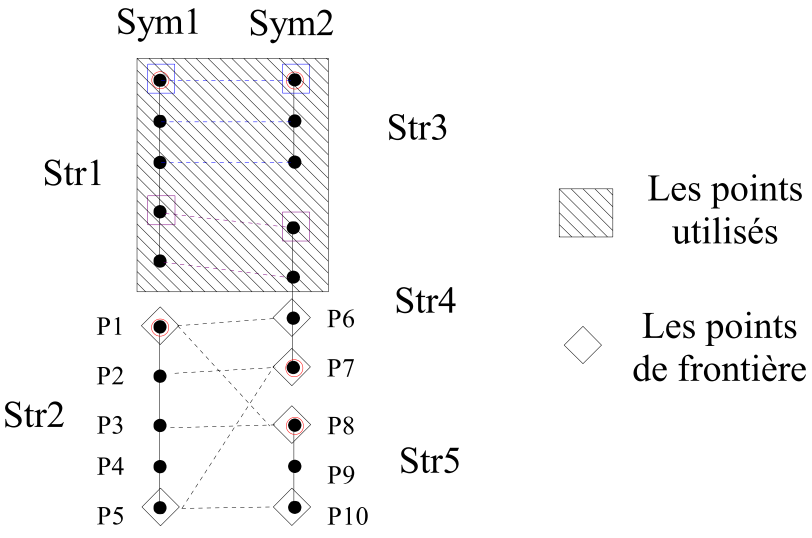


Figure 9.13: Six couples de points de départ (P1,P6), (P1,P8), (P5,P7), (P5,P10), (P2,P7) et (P3,P8) pour continuer la mise en correspondance démarrée dans la Figure 9.12b

9.3.3 Etude Expérimentale

L'algorithme est encore coûteux en temps et en mémoire pour le calcul malgré la réduction des points de départ présentée dans la section précédente. Nous proposons dans cette section une étude qualitative qui montre la qualité de l'alignement obtenu.

L'Alignement entre Deux Sembles de Séquences

Avant la mise en correspondance, tous les symboles sont re-échantillonnés en un nombre fixe de 20 points et sont mis à la même échelle. Les seules coordonnées (x,y) sont utilisées pour calculer la distance euclidienne entre deux points. La Figure 9.14 compare les résultats obtenus par l'algorithme DTW classique et par notre DTW A* entre deux formes similaires mais écrites utilisant des sens d'écriture et des ordres de traits différents. La Figure 9.14a présente un exemple de deux séquences avec deux sens différents.

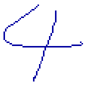
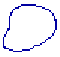
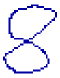
La distance trouvée avec l'algorithme DTW A* proposé ici, visible sur la Figure 9.14c est plus petite que celle obtenue avec la DTW classique, Figure 9.14b.

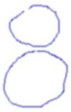
Les exemples présentés des cas 2 (Figure 9.14d) au cas 4 (Figure 9.14j) comparent un symbole composé d'un trait avec le même symbole composés de deux traits écrits dans différents sens et ordres. Notre algorithme choisit le meilleur alignement parmi les différents sens et ordres possibles. Le dernier cas, Figure 9.14m, montre la capacité du système à aligner deux symboles multi-traits avec inversion du sens et de l'ordre. Ces exemples montrent bien que notre algorithme est indépendant du sens et de l'ordre des traits composant les symboles.

La Figure 9.15 montre un exemple plus compliqué de comparaison de deux allographes de x. Notre algorithme a trouvé la meilleure solution en cinq étapes. Les deux premières étapes montrent l'alignement de la branche en haut à gauche. La branche en bas à droite est alignée dans la troisième étape, etc. Notre algorithme peut couper les traits en sous-segments qui minimisent la distance DTW avec le segment correspondant dans l'autre symbole.

Distance entre Symboles Différents

Nous choisissons six symboles pour illustrer l'utilisation de cette distance (Équation 9.7) dans une application de type classifieur basé plus-proche-voisin. Les exemples sont ré-échantillonnés en 30 points. Nous pouvons constater que les formes '4' et '8' sont bien classifiées bien que les allographes n'aient pas le même nombre de traits. Le temps de calcul et le nombre d'hypothèses (la mémoire requise) sont donnés à titre indicatif et dépendent de la complexité de la forme et surtout du nombre de points utilisés pour le ré-échantillonnage.

	 (2 str)	 (1 str)	(2 str)	 (1 str)

 (2 str)	dist=0,29
	temps=127 sec
	77.304 hyp
		dist=0,37
		temps<1sec
		3.749 hyp
			dist=0,23
			temps=787sec
			238.193 hyp
				dist=0,17
				temps<1sec
				218 hyp

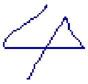
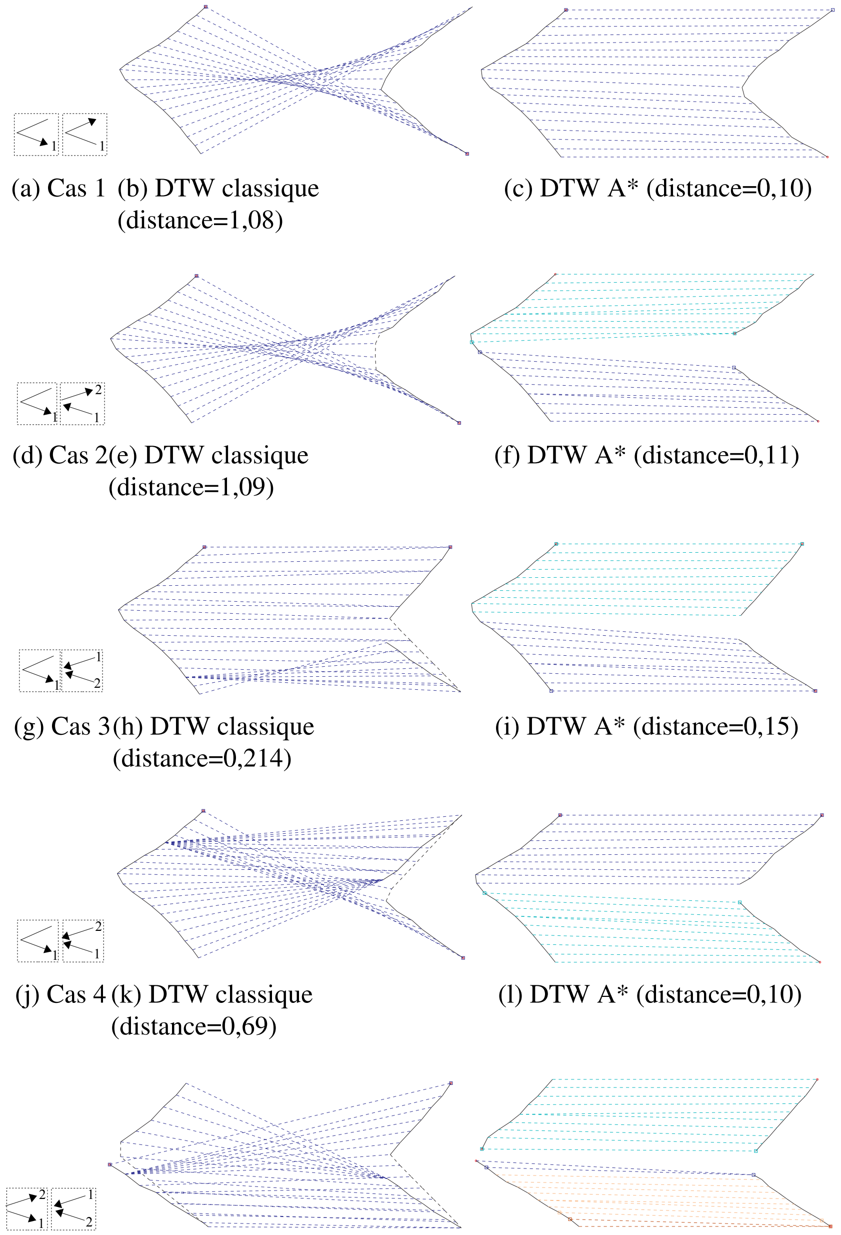
 (1 str)	dist=0,15
	temps<1 sec
	112 hyp
		dist=0,44
		temps<1 sec
		699 hyp
			dist=0,27
			temps=176 sec
			88.820 hyp
				dist=0,37
				temps<1sec
				16 hyp

Table 9.2: Distance entre deux symboles et quatre symboles (dist=distance, sec=seconde, str=trait et hyp=hypothèse).

9.3.4 Conclusion

Dans cette thèse, nous avons proposé une distance entre deux ensembles de séquences en préservant la continuité intra-séquence. En procédant directement,



(a) Cas 1 (b) DTW classique (distance=1,08) (c) DTW A* (distance=0,10)

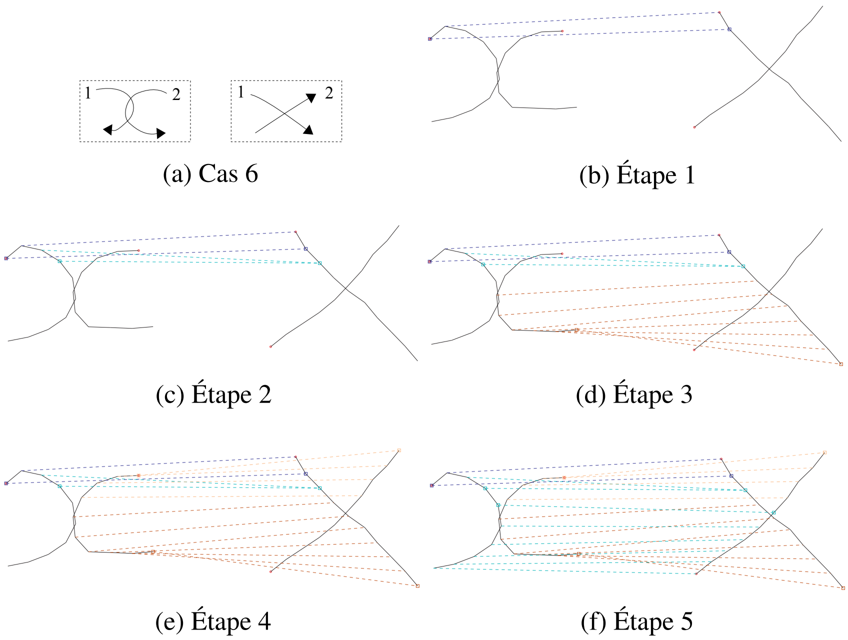
(d) Cas 2(e) DTW classique (distance=1,09) (f) DTW A* (distance=0,11)

(g) Cas 3(h) DTW classique (distance=0,214) (i) DTW A* (distance=0,15)

(j) Cas 4 (k) DTW classique (distance=0,69) (l) DTW A* (distance=0,10)

(m) Cas 5(n) DTW classique (distance=0,55) (o) DTW A* (distance=0,10)

Figure 9.14: Tests sur la mise en correspondance entre deux ensembles de sequences



(a) Cas 6

(b) Étape 1

(c) Étape 2

(d) Étape 3

(e) Étape 4

(f) Étape 5

Figure 9.15: La meilleure solution entre deux x

il faudrait trouver le meilleur alignement parmi un nombre exponentiel de combinaisons. Notre approche nommée DTW A*, basée sur la distance DTW et l'algorithme de A*, permet de réduire cette complexité en explorant les sous-alignements possibles. Nous limitons notamment les points de départ des alignements possibles pour réduire la combinatoire. Cette stratégie permet aussi d'obtenir des alignements qui n'aurait pas été possible par un simple ré-ordonnancement des traits. Les résultats qualitatifs présentés montrent l'intérêt de DTW A*. Il s'agit maintenant de réaliser une implémentation permettant la mise en œuvre d'un classifieur. En particulier il sera utile d'arrêter le calcul de la distance dès qu'elle dépasse la plus petite déjà disponible.

Signalons toutefois que l'algorithme DTW A* malgré les optimisations qui y ont été apportées est pénalisé par un temps d'exécution important. Dans la suite de cette thèse, nous lui préférerons un algorithme plus rapide basé sur la Distance de Hausdorff Modifiée (DHM) [48]. Dans la section suivante, nous présentons la méthode permettant globalement de découvrir des symboles multi-traits.

9.4 Découverte des Symboles Multi-Traits

Dans cette section, nous donnons un aperçu général de la méthode proposée pour extraire l'ensemble des symboles graphiques qui composent le lexique utilisé dans un corpus de documents manuscrits. La méthode comporte trois étapes principales: i) la quantification des traits élémentaires (stroke) en graphèmes prototypes, ii) la construction d'un graphe décrivant les relations spatiales entre les traits, et finalement iii) la définition des éléments du lexique en s'appuyant sur les graphèmes et leurs relations.

Comme le montre la Figure 9.16, à partir d'un nouveau document graphique, il est d'abord important de rechercher les différents graphèmes qui correspondent aux différentes formes élémentaires en usage dans le document. Pour cela, une technique de clustering est mise en œuvre pour construire les représentants de ces graphèmes, ces prototypes permettent de définir les éléments de base d'un dictionnaire. Celui-ci est ensuite utilisé pour annoter chaque trait élémentaire à l'aide de son représentant. Dans un second temps, un graphe de relations spatiales est construit, celui-ci se base sur l'approche des SRT (Symbol Relation Tree approach) [31]. Dans une troisième étape, les motifs fréquents présents dans le graphe relationnel sont extraits en s'appuyant sur le principe de longueur de description minimale (MDL) [13, 18]. Ce principe permet d'identifier un modèle à partir de données, en considérant le problème du choix du modèle comme celui permettant de déceler les régularités des données. Les éléments ayant produit le code le plus compact sont considérés comme les éléments de base du langage considéré et donc les symboles ou encore l'alphabet de ce langage graphique.

Dans l'exemple de la Figure 9.16, nous supposons qu'une base de documents manuscrits contienne le diagramme «$\square \rightarrow \square \rightarrow \square$». Chaque trait est marqué de manière unique par un index (i) de 1 à 15. A un moment donné, il est possible de disposer du lexique de graphèmes suivant $\{$a‘$-$’, b‘$\neg$’, c‘$|$’, d‘$\backslash$’, e‘$/$’$\}$. Celui-ci pouvant résulter comme indiqué plus haut d'un clustering non supervisé. Il est alors possible de coder le schéma en s'appuyant sur ce lexique. Cela correspond à une étape de quantification sur l'ensemble des graphèmes prototypes du schéma étudié. Ensuite, les relations spatiales qui sont présentes entre ces composantes permettent de produire un graphe relationnel. Les relations spatiales que l'on peut considérer

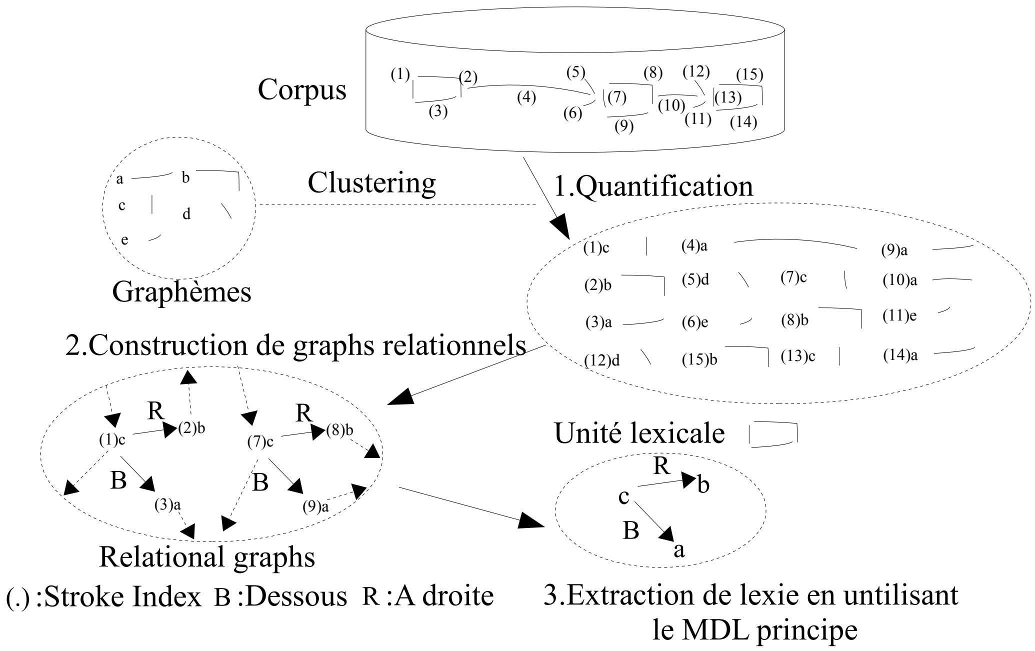
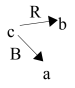

Corpus
(1) (2) (3) (4) (5) (6) (7) (8) (9) (10) (11) (12) (13) (14) (15)
a b c d e
Graphèmes
Clustering
1.Quantification
(1)c (2)b (3)a (12)d (4)a (5)d (6)e (15)b (7)c (8)b (13)c (9)a (10)a (11)e (14)a
2.Construction de graphs relationnels
R (1)c (2)b B (3)a
R (7)c (8)b B (9)a
Relational graphs
(.) :Stroke Index B :Dessous R :A droite
Unité lexicale
R c b B a
3.Extraction de lexie en untilisant le MDL principe

Figure 9.16: Schéma général de la méthode découverte des symboles graphiques

sont nombreuses ainsi que nous le verrons dans la Chapter 5. Dans cet exemple simplifié, la sous-structure () apparaît fréquemment, elle peut être mise à jour à partir du graphe relationnel. Cette sous-structure correspond à la présence d'un trait de type a '−' placé en dessous d'un trait de type c'|' qui possède sur sa droite un trait de type b'¬'. Nous pourrions considérer cette sous-structure comme un élément du lexique du langage graphique. Dans ce cas, cet élément correspond à un symbole rectangle ('□').

Dans cette section, nous détaillons comment obtenir le lexique à partir du graphe relationnel et nous évaluons la qualité du lexique dans une tache de segmentation hiérarchique.

9.4.1 Découverte non supervisée des symboles graphiques

Pour découvrir les symboles graphiques, nous commençons par produire les graphèmes prototypes à partir d'un clustering hiérarchique. Après avoir quantifié chaque trait, le graphe relationnel est construit, il permet de modéliser le contexte spatial de chaque trait. Les structures répétitives que l'on considérera comme les unités lexicales seront ensuite extraites par application d'un algorithme par codage du graphe par principe de longueur de description minimale.

9.4.2 Quantification des Traits

Nous nous intéressons ici à des tracés graphiques de type en-ligne disponibles sous la forme d'une séquence de traits, eux-mêmes constitués d'une séquence de points provenant du plan 2D. Du fait de la variabilité des formes produites par les tracés manuscrits, il est utile de rapprocher chaque forme d'un tracé de référence disponible dans le lexique des graphèmes. Pour cela, nous utiliserons la Distance Modifiée de Hausdorff (DMH) [48], elle permet de mesurer la dissimilarité entre deux formes. Rappelons que le lexique résulte d'une opération de clustering. Plutôt que d'utiliser un classique algorithme de type *k-moyennes* qui est très sensible à l'initialisation et qui nécessite de fixer le nombre de classes, nous avons préféré une technique de clustering hiérarchique ascendante qui permet d'obtenir une structure d'arbre propice pour fixer a posteriori le nombre de classes, c'est-à-dire de prototypes de graphèmes. Ce nombre n_p de graphèmes étant fixé et les prototypes correspondants calculés, tous les traits sont marqués du label virtuel du prototype le plus proche dans l'espace des caractéristiques, la distance utilisée étant encore ici la DMH. Cela réalise donc l'étape de quantification des traits. A partir de là, le graphe relationnel définissant les relations spatiales entre les traits est construit.

9.4.3 Construction du Graphe Relationnel

On présente dans cette section la méthode de construction du graphe relationnel, celle-ci est inspiré du formalisme SRT [31] permettant de décrire les relations spatiales entre les traits d'un document graphique. Dans ce graphe, les nœuds représentent les traits, chacun étant étiqueté par son prototype de graphèmes, les arcs correspondant à des relations spatiales spécifiques. Une relation spatiale met en correspondance un trait de référence avec un trait cible. Il en résulte un graphe orienté, celui-ci est décrit plus en détails dans la Section 5. Nous allons maintenant présenter dans la section suivante, la façon de procéder pour extraire de façon hiérarchique les sous-structures, c'est-à-dire les sous-graphes, à partir de ce graphe relationnel. L'ensemble des sous-structures fréquentes, en considérant différents niveaux de hiérarchie, permettra de composer le lexique recherché, c'est-à-dire l'ensemble des symboles utilisés dans le langage qui a servi à composer tous les documents dont on dispose dans la base de documents.

9.4.4 Extraction du Lexique par Utilisation du Principe de Longueur de Description Minimale

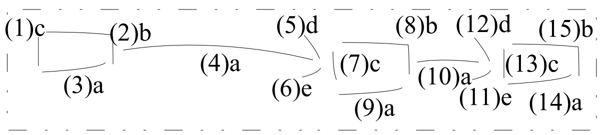
Nous avons vu dans la section précédente comment construire le graphe relationnel décrivant un document graphique. Dans cette section, nous présentons un algorithme s'appuyant sur les travaux [13] et utilisant le principe de longueur de description minimale (MDL) [18] pour l'extraction des sous-structures répétitives (sous-graphes) dans un graphe, celles-ci étant considérées dans notre contexte comme des unités lexicales. De façon non supervisée, l'apprentissage d'un langage par le principe MDL propose de retenir les unités lexicales qui minimisent globalement la description du lexique et du graphe décrit par le lexique [11]. Formellement, étant donné un graphe G, cela revient à retenir les unités lexicales u qui minimisent la longueur de description de la grandeur.

$$DL(G, u) = I(u) + I(G|u) \tag{9.12}$$

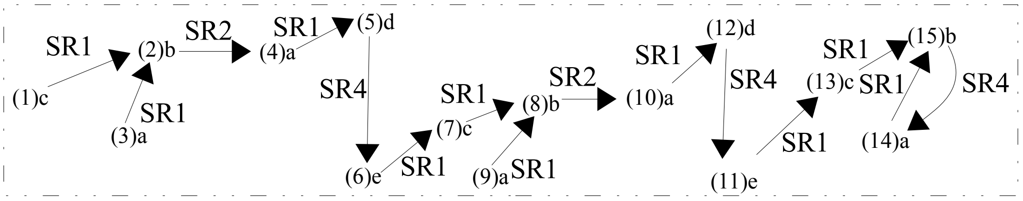
Où $I(u)$ représente le nombre bits nécessaire pour encoder le lexique u et $I(G|u)$ est le nombre de bits pour encoder le graphe G en utilisant le lexique u. Une définition précise de $DL(G, u)$ est proposée dans [23]. Pour solutionner ce problème, la méthode SUBDUE (SUBstructure Discovery Using Examples) [23] extrait de façon itérative les meilleures unités lexicales (sous-structures) en utilisant le principe MDL. Il est possible de concevoir des unités lexicales composites grâce à l'application itérative de la méthode [24].

Pour illustrer la mise en œuvre de cette procédure itérative et la structure hiérarchique résultante, nous proposons de continuer à nous appuyer sur l'exemple du diagramme de la Figure 9.17. A ce niveau, nous avons pu décrire une représentation graphique par un graphe relationnel utilisant les étiquettes (noeuds) issues du clustering hiérarchique et les arcs définissant des relations spatiales privilégiées. A partir de ce schéma graphique, nous souhaitons extraire les motifs répétitifs, et cela sous la forme d'une éventuelle structure multi niveaux. Dans l'exemple proposé, nous voyons apparaître la structure hiérarchique suivante : "$\rightarrow$ □".

Il y a trois instances du rectangle "□" dans ce diagramme, il s'agit du motif le plus fréquent. Probablement que le sous-graphe "□" correspondant sera extrait dès la première itération et permettra de définir la première unité lexicale, soit LU_1



(a) Un schéma graphique



(b) Le graphe relationnel correspondant

Figure 9.17: Le graphe relationnel (b) correspondant au schéma (a)

("□"). Lors de la seconde itération, un autre motif répétitif devrait être extrait, il s'agit de la seconde unité lexicale, soit LU_2 ("→"). On convient de noter LU_i l'unité lexicale extraite lors de l'$i$ème itération. Bien entendu, l'extraction ne se fait pas sur la base d'un seul schéma tel que celui de la Figure 9.18 mais résulte d'une analyse de l'ensemble des productions graphiques afin de pouvoir s'appuyer un calcul de fréquence significatif de la production langagière et donc de l'alphabet sous-jacent. S'il se trouvait que la sous-structure LU_2("→") est plus fréquente que la sous-structure LU_1 ("□"), alors ce serait elle qui serait extraite lors de la première itération, et cela selon le principe MDL.

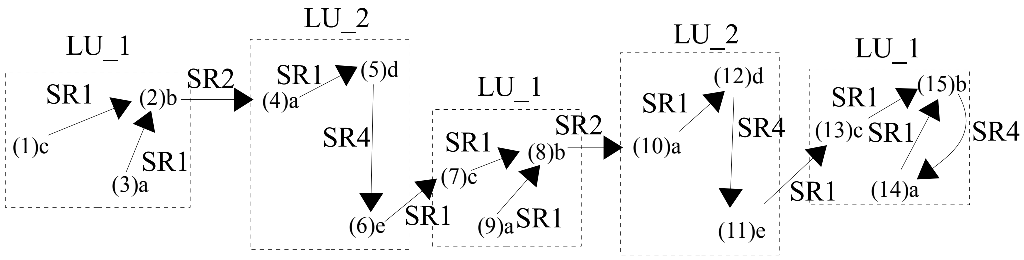


Figure 9.18: Extraction de deux unités lexicales, LU_1 (3 instances de "□") et LU_2 (2 instances de "→"), obtenues lors de la première et seconde itération

Après chaque itération, les sous-structures extraites sont remplacées par un nœud global étiqueté par l'unité lexicale virtuelle mise à jour. Ainsi, à l'issue de l'itération 2, le graphe relationnel est défini par la Figure 9.19. En continuant une itération de plus, l'algorithme proposerait une nouvelle unité lexicale LU_3

résultant du sous-graphe contenant LU_1 et LU_2. Cet exemple illustre bien la notion de hiérarchie dans les unités lexicales. Lorsqu'il n'est plus possible de réduire la longueur de description, la construction du lexique s'arrête et celui-ci est composé d'une liste d'éléments L =(LU_1, LU_2, LU_3,...) contribuant au calcul de $DL(G, u)$.

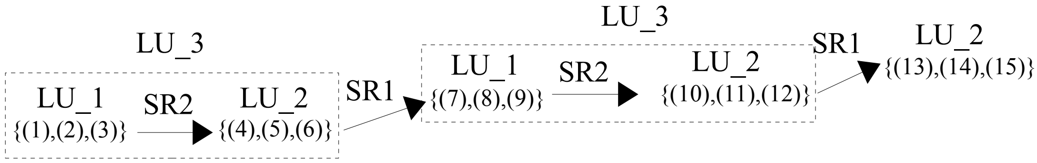


Figure 9.19: Extraction d'une unité lexicale hiérarchique, LU_3 composée des éléments LU_1 et LU_2

9.4.5 Évaluation des Segmentations

Nous allons introduire dans cette thèse une mesure de rappel au niveau symbole multi-traits [11, 33], celle-ci se base sur l'intersection entre la segmentation de la vérité terrain et la segmentation hiérarchique obtenue précédemment. Le taux de rappel R_{MRecall} évalue le pourcentage de segmentation correcte qui est présent dans la vérité terrain des symboles multi-traits.

9.4.6 Conclusion

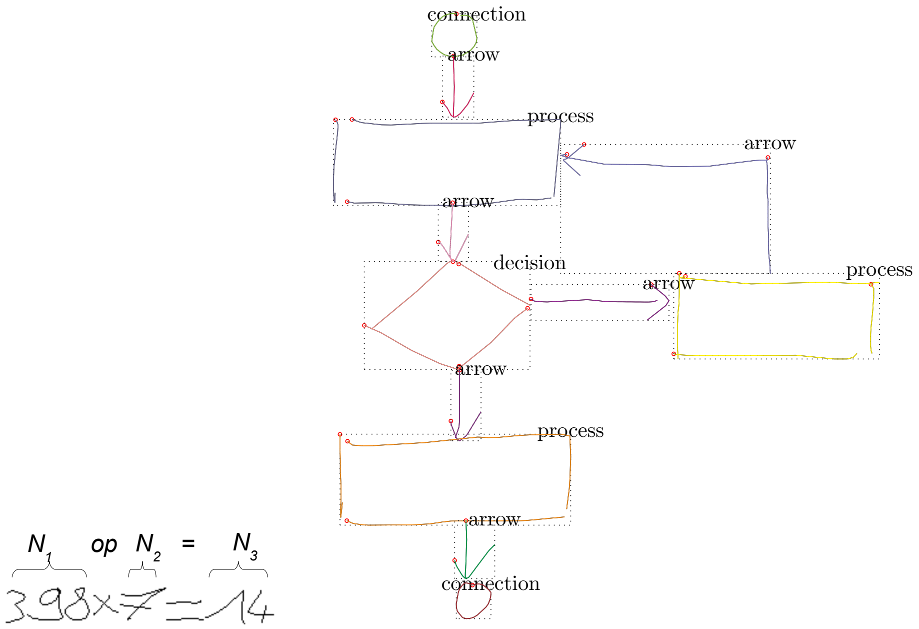
Nous avons proposé dans cette section une méthode permettant d'extraire des symboles multi-traits en se basant sur une représentation de graphe relationnel et sur le principe MDL. L'approche proposée comporte trois étapes : i) la quantification des traits élémentaires en graphème prototype, ii) la quantification des relations spatiales entre les paires de traits, iii) et pour finir l'extraction des symboles multi-traits (les unités lexicales) à partir du graphe des relations spatiales par mise en œuvre du principe MDL. Il en résulte un ensemble d'unités lexicales qui possèdent la propriété de se définir par une structure hiérarchique, ce qui correspond bien à la perception humaine de la composition de symboles. Dans la mesure où la construction du lexique conduit à une segmentation en symboles, nous avons proposé une mesure permettant d'évaluer la qualité des symboles extraits. Pour mener à bien des expérimentations pour évaluer cette méthodologie, nous allons introduire dans la section suivante deux bases de données intégrant des données manuscrites graphiques de type en-ligne.

9.5 Description des Bases Utilisées

La première base d'expérimentation que nous considérerons sera la base *Calc* (Calculette) qui correspond à un ensemble d'expressions mathématiques simples produites par synthèse automatique à partir de symboles manuscrits isolés [59]. Les expressions de *Calc* respectent la grammaire N_1 *op* $N_2 = N_3$ ou N_1, N_2 où N_1, N_2 et N_3 sont des nombres composés de 1, 2 ou 3 digits manuscrits. La distribution du nombre des digits pour les $N_i = \{1, 2, 3\}$ est de 70% de 1 digit, 20% de 2 digits et 10% de 3 digits, chaque digit est tiré suivant une loi uniforme sur $[0..9]$. De même op représente l'un des opérateurs de l'ensemble $\{+, -, \times, \div\}$. La Figure 9.20a montre un exemple d'expression de cette base Calc, dans ce cas N_1, N_2 et N_3 comportent 3 digits, 1 digit et 2 digits respectivement, tandis que l'opérateur est le symbole 'x'. Le nombre de classes de symboles est de 15, à savoir les 10 digits, les 4 opérateurs arithmétiques et le symbole '$=$'.

La seconde base d'expérimentation est une base de graphiques manuscrits de type organigramme, elle est dénommée *FC* (flowchart) [60]. Ces organigrammes comportent 6 types de symboles qui représentent les opérations basiques suivantes : donnée, fin, process, choix, connexion, flèche, à l'exclusion de tout texte. Un exemple est présenté en Figure 9.20b. Pour cette application, le nombre de classes est donc de 6.

Le Tableau 9.3 présente quelques statistique sur deux bases. Chacune est composée d'une partie apprentissage et d'une partie de test. Bien que la base *Calc* ait plus de classes que la base *FC*, dernière a plus de traits par symbole. De plus, la Figure 9.21 montre la distribution du nombre de symboles par rapport du nombre de traits par symbole. La plupart des symboles (54,9%) de la base *Calc* sont des symboles mono-trait, et 40,2% de symboles contiennent deux traits. Par contre la base *FC* contient évidemment plus de symboles multi-traits. En plus de la forte proportion de symboles multi-traits, la composition spatiale entre les symboles de la base *FC* est plus variée par comparaison aux expressions mathématiques mono-lignes de la base *Calc*. La base *FC* représente donc un challenge plus difficile pour



(a) (b)

Figure 9.20: Deux corpus pour deux langages graphiques: (a) une expression synthétique de la base *Calc*, (b) un exemple d'organigramme de programmation de la base *FC*.

l'extraction des symboles isolés.

N. = Nombre

		N. de Symboles	N. de traits	Traits/Symbole	N. de Classes	N. de Scripteurs
Calc	Apprentissage	5472	8185	1.50	15	180
	Test	3035	4547	1.50	15	100
FC	Apprentissage	3641	8827	2.42	6	31
	Test	2494	6059	2.43	6	15

Table 9.3: Nombre de symboles et nombre de classes dans les deux bases d'écriture manuscrite.

Dans la section suivante, nous présentons les résultats sur les deux bases de données respectivement. Puis nous concluons nos travaux.

9.6 Résultats et Discussions

Dans cette section, nous comparons différentes approches d'extraction des symboles dans le cadre de l'aide à l'étiquetage au niveau symboles d'une base d'écriture

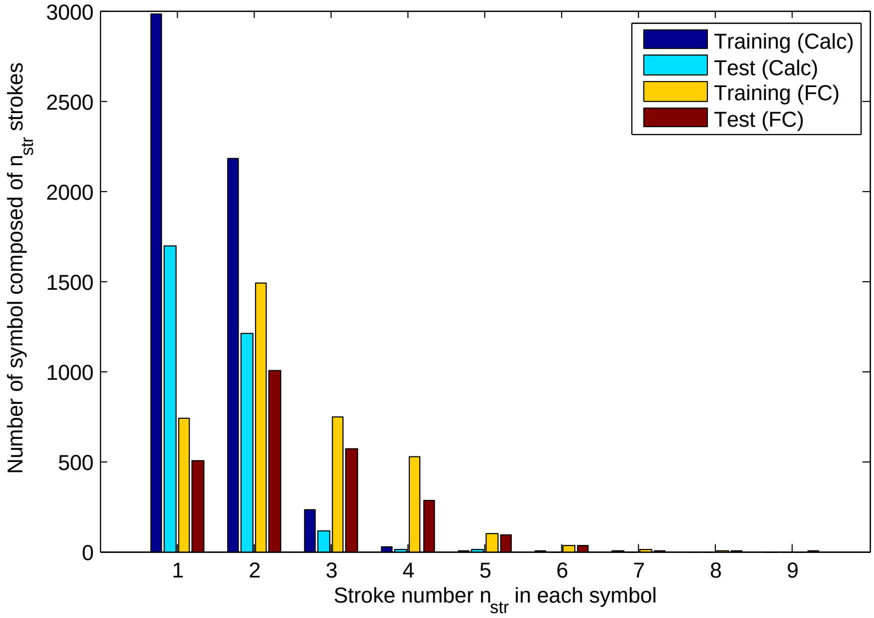
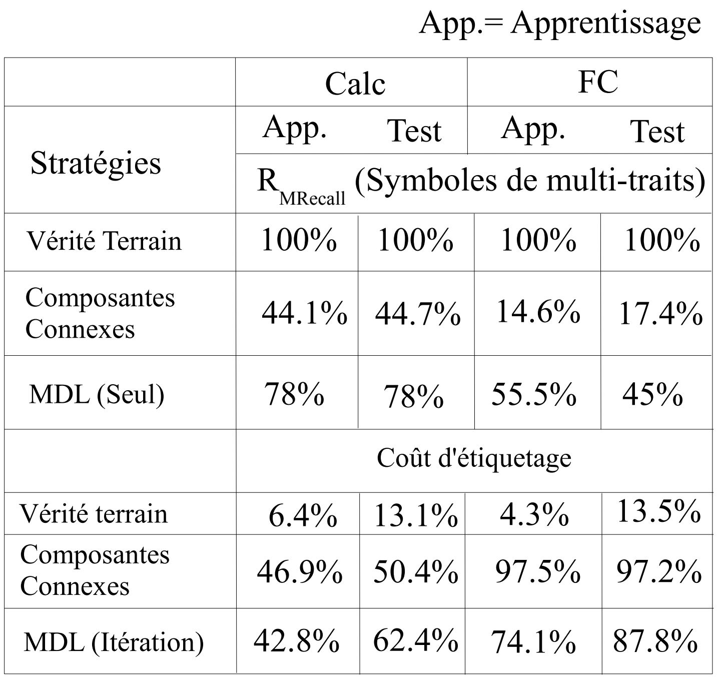


Figure 9.21: Distribution des symboles par nombre de traits

manuscrite. Nous évaluons donc les résultats en considérant deux mesures présentées précédement. La première compte le nombre de symboles multi-traits correctement retrouvés automatiquement (taux de rappel R_{MRecall}). La seconde mesure compte le nombre d'opérations manuelles nécessaires à l'opérateur humain pour étiqueter toute la base : il faut étiqueter chaque trait du dictionnaire visuel puis cette information est automatiquement transférée à toute la base, l'utilisateur devra ensuite corriger chaque trait des symboles mal catégorisés par le clustering. Le coût d'étiquetage est donc égale au nombre de traits des représents du dictionnaire et au nombre de traits mal étiquetés automatiquement.

Trois stratégies de segmentations sont comparées. La première utlisera directement la segmentation disponible dans la vérité terrain de la base. Il s'agira d'une référence maximum car il n'est pas possible de faire une meilleure segmentation. La seconde utilise les composantes connexes (i.e. les traits se touchant sont dans le même symbole) comme critère de segmentation. Cette approche simple est courement utilisées dans les systèmes existants, il s'agira d'une référence de base. La dernière stratégie est notre meilleure approche : utilisation de l'exploration des graphes relationnels pour découvrir les symboles fréquents (voir la section 9.4.4).

Le Tableau 9.4 présente les résultats de ces trois approches pour les deux mesures considérées, sur les deux bases d'écriture manuscrites présentées précédemment.



App.= Apprentissage

	Calc		FC	
Stratégies	App.	Test	App.	Test
	$R_{MRecall}$ (Symboles de multi-traits)			
Vérité Terrain	100%	100%	100%	100%
Composantes Connexes	44.1%	44.7%	14.6%	17.4%
MDL (Seul)	78%	78%	55.5%	45%
	Coût d'étiquetage			
Vérité terrain	6.4%	13.1%	4.3%	13.5%
Composantes Connexes	46.9%	50.4%	97.5%	97.2%
MDL (Itération)	42.8%	62.4%	74.1%	87.8%

Table 9.4: Résultats sur les deux bases de données

La première partie de résultats $R_{MRecall}$ dans le Tableau 9.4 montre que beaucoup de symboles dans les deux bases sont automatiquement extraits par notre stratégie et confirme que la base *FC* est plus complexe puisque seulement la moitié des symboles y sont retrouvés. Notre segmentation proposée fonctionne bien que la stratégie de composantes.

Concernant les résultats de taux d'étiquetage manuel, la segmentation utilisant la vérité terrain montre évidement le plus faible coût d'étiquetage, mais dans un cas réel nous ne connaissons pas cette vérité terrain. L'utilisation des composantes connexes fait augmenter de beaucoup le taux d'étiquetage, ce qui montre qu'une approche simple ne permet pas de résoudre le problème. Nous pouvons aussi constater que cette approche fonctionne mieux dans un contexte de base où les symboles sont généralement séparés (*Calc*), ce qui n'est pas le cas dans la base *FC*. Notre stratégie de segmentation proposée donne de meilleurs résultats que les composantes connexes dans la partie apprentissage. Dans la partie de test, la stratégie de composantes connexes fonctionne mieux que notre segmentation proposée. Cette différenc peut s'expliquer par le fait qu'il y a beaucoup de paramètres dans notre système et que les paramètres utilisés en test sont ceux optimisés sur la base d'apprentissage. Enfin nous pouvons constater que sur la base plus complexe

FC, notre stratégie permet de réduire le coût d'étiquetage par rapport à la segmentation en composantes connexes. Le taux de rappelle étant nettement plus faible que le taux d'étiquetage, nous pouvons affirmer qu'il existe une forte marge de progression dans les deux axes : la découverte des symboles et le clustering en symboles homogènes.

9.7 Conclusions

La création d'une base d'apprentissage étiquetée au niveau de chaque symbole est d'une tâche coûteuse. Dans cette thèse, nous avons proposé un schéma qui extrait des symboles graphiques en utilisant le principe MDL. Le schéma proposé peut réduire le travail d'étiquetage des symboles. Notre approche contient trois étapes principales : quantifier des traits ou symboles graphiques, construire un graphe relationel, découvrir des symboles multi-traits.

Dans les données en ligne, les éléments basiques sont des traits. Nous proposons de modéliser un langage graphique par un graphe relationnel. Les noeuds sont définis par les traits et les arcs sont définis par les relations spatiales. Les traits et les relations spatiales sont quantifiés par clustering. Nous pouvons donc extraire des symboles multi-traits (sous-graphes) dans les graphes. Après la découverte des symboles multi-traits, nous produisons un dictionnaire visuel des différents regroupements obtenus par clustering. Cette opération nécessite la définition d'une une distance adaptée aux symboles multi-traits. Nous proposons une distance nommée DTW A* basée sur la distance d'alignement élastique DTW mais permettant l'alignement optimal de deux symboles multi-traits. Cette distance DTW A* est indépendante du nombre de traits, de leurs sens et de leur ordre d'écriture. Signalons toutefois que cet algorithme DTW A* malgré les optimisations qui y ont été apportées est pénalisé par un temps d'exécution important. Dans cette thèse, nous préférerons une métrique plus rapide mais moins stable basée sur la Distance de Hausdorff Modifiée (DHM).

Après la quantification des traits et la construction de graphes relationnels, nous recherchons les sous-graphes fréquents en utilisant le principe MDL. Ces sous-graphes sont alors considérés comme des hypothèses de symboles multi-traits. Et puis ces symboles multi-traits sont groupés dans un dictionnaire visuel où nous étiquetons ces symboles à la main. Le système transfère les étiquettes de chaque trait à toutes les données de la base.

Nous avons testé notre méthode d'étiquetage automatique sur deux bases d'écritures manuscrites, *Calc* (assez simple) et *FC* (plus difficile car avec plus de variabilité). Nous avons réussi à réduire significativement le travail d'étiquetage manuel des symboles. La comparaison de notre approche avec une stratégie plus classique (utilisation des composantes connexes) montre que notre système est adapté aux données complexes mais qu'il reste une marge de progression importante.

List of Tables

List of Figures

Abbreviations

ANN	**A**rtificial **N**eural **N**etworks
DAG	**D**irected **A**cyclic **G**raph
DTW	**D**ynamic **T**ime **W**arping
HD	**H**ausdorff **D**istance
HMM	**H**idden **M**arkov **M**odel
KNN	**K**-**N**earest **N**eighbor
MDL	**M**inimun **D**escription **L**ength
MHD	**M**odified **H**ausdorff **D**istance
NMI	**N**ormalized **M**utual **I**nformation
OCR	**O**ptical **C**haracter **R**ecognition
SRT	**S**ymbol **R**elation **T**ree
SOM	**S**elf **O**rganizing **M**ap
SVM	**S**upport **V**ector **M**achine
NG	**N**eural **G**as
GNG	**G**rowing **N**eural **G**as

Symbols

n_p	Prototype Number During Clustering for Multi-stroke symbols
n_{sr}	Prototype Number During Clustering for Spatial Relations
n_{str}	Number of strokes
n_{seg}	Number of segments
n_{cstr}	Number closest stroke(s) or symbol(s) from a reference stroke or symbol for generating a relational graph
n_u	Number of discovered lexical units using SUBDUE in each iteration.
n_{it}	Number of iteration in a codebook mapping system.

Publications

[1] Jinpeng Li, Harold Mouchère and Christian Viard-Gaudin. Reducing Annotation Workload Using a Codebook Mapping and its Evaluation in On-Line Handwriting, ICFHR2012.

[2] Jinpeng Li, Harold Mouchère and Christian Viard-Gaudin. Une distance entre deux ensembles de séquences avec la contrainte de continuité (A distance between two sequence sets with the continuity constraint), Semaine du document numérique et de la recherche d'information 2012, CIFED2012, Bordeaux, France (Oral presentation)

[3] Jinpeng Li, Harold Mouchère and Christian Viard-Gaudin. Quantifying spatial relations to discover handwritten graphical symbols, Document Recognition and Retrieval XIX, Part of the IS&T/SPIE 24th Annual Symposium on Electronic Imaging, 22-26 January 2012, San Francisco, CA USA. (Oral presentation)

[4] Jinpeng Li, Harold Mouchère and Christian Viard-Gaudin. Unsupervised Handwritten Graphical Symbol Learning Using Minimum Description Length Principle on Relational Graph, International Conference on Knowledge Discovery and Information Retrieval, KDIR 2011, Paris, France. (Oral presentation)

[5] Jinpeng Li, Harold Mouchère and Christian Viard-Gaudin. Symbol Knowledge Extraction from a Simple Graphical Language, 11th International Conference on Document Analysis and Recognition, ICDAR2011, Beijing, China. (Oral presentation)

Bibliography

[1] R. Plamondon and S.N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 63–84, 2000.

[2] Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: A survey. *International Journal on Document Analysis and Recognition*, 3(1): 3–15, 2000.

[3] Claudie Fraure and Zi Xiong Wang. Automatic perception of the structure of handwritten mathematical expressions. In *Computer Porcessing of Handwriting*, 1990.

[4] M. Okamoto and A. Miyazaki. An experimental implementation of a document recognition system for papers containing mathematical expressions. *Structured Document Image Analysis*, pages 36–53, 1992.

[5] Jaekyu Ha, R. M. Haralick, and I. T. Phillips. Understanding mathematical expressions from document images. In *Proceedings of the Third International Conference on Document Analysis and Recognition*, Washington, DC, USA, 1995. IEEE Computer Society.

[6] Steve Smithies and et al. A handwriting-based equation editor. *Proceedings of Graphics Interface*, pages 84–91, 1999.

[7] Brijmohan Singh, Ankush Mittal, and Debashis Ghosh. An evaluation of different feature extractors and classifiers for offline handwritten devnagari character recognition. *Journal of Pattern Recognition Research*, 2:269–277, 2011.

[8] Sumit Saha and Tanmoy Som. Handwritten character recognition by using neural-network and euclidean distance metric. In *International Journal of Computer Science and Intelligent Computing*, 2010.

[9] Abdul R. Ahmad, M. Khalia, C. Viard-Gaudin, and E. Poisson. Online handwriting recognition using support vector machine. In *IEEE Region 10 Conference : proceedings : analog and digital techniques in electrical engineering (TENCON)*, volume A, pages 311–314 Vol. 1, 2004.

[10] Yongqiang Wang, Qiang Huo, and Yu Shi. A study of discriminative training for hmm-based online handwritten chinese/japanese character recognition. In *Frontiers in Handwriting Recognition (ICFHR), 2010 International Conference on*, pages 518 –523, nov. 2010.

[11] Carl De Marcken. *Unsupervised Language Acquisition*. PhD thesis, Massachusetts Institute of Technology, 1996.

[12] Vuokko Vuori. *Adaptive Methods for On-Line Recognition of Isolated Handwritten Characters*. PhD thesis, Helsinki University of Technology (Espoo, Finland), 2002.

[13] Diane J. Cook and Lawrence B. Holder. Substructure discovery using minimum description length and background knowledge. *Journal of Artificial Intelligence Research*, 1:231–255, February 1994.

[14] S. Vajda, A. Junaidi, and G. A. Fink. A semi-supervised ensemble learning approach for character labeling with minimal human effort. In *International Conference on Document Analysis and Recognition*, pages 259–263, 2011.

[15] J. Richarz, S. Vajda, and G. A. Fink. Annotating handwritten characters with minimal human involvement in a semi-supervised learning strategy. In *Proc. Int. Conf. on Frontiers in Handwriting Recognition*, Bari, Italy, 2012. to appear.

[16] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y. Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 609–616, New York, NY, USA, 2009. ACM.

[17] Chris Fox Alexander Clark and Shalom Lappin. *The Handbook of Computational Linguistics and Natural Language Processing*. Wiley-Blackwell, 2010.

[18] J. Rissanen. Modeling by shortest data description. *Automatica*, 14(5):465 – 471, 1978.

[19] Andrew R. Barron, Jorma Rissanen, and Bin Yu. The minimum description length principle in coding and modeling. *IEEE Transactions on Information Theory*, 44(6):2743–2749, 1998.

[20] Carl De Marcken. Linguistic structure as composition and perturbation. In *In Meeting of the Association for Computational Linguistics*, pages 335–341. Morgan Kaufmann Publishers, 1996.

[21] Nelson W. Francis and Henry Kučera. *Frequency Analysis of English Usage: Lexicon and Grammar.*, volume 18. Houghton Mifflin, Boston, April 1982.

[22] Romain Raveaux, Jean-Christophe Burie, and Jean-Marc Ogier. Graphics recognition. recent advances and new opportunities. chapter A Segmentation Scheme Based on a Multi-graph Representation: Application to Colour Cadastral Maps, pages 202–212. Springer-Verlag, Berlin, Heidelberg, 2008.

[23] Diane J. Cook and Lawrence B. Holder. *SUBstructure Discovery Using Examples, ailab.wsu.edu/subdue/.* 2011.

[24] Istvan Jonyer, Lawrence B. Holder, and Diane J. Cook. Graph-based hierarchical conceptual clustering. *International Journal on Artificial Intelligence Tools*, 2:107–135, 2000.

[25] Eliseo Clementini. *A Conceptual Framework for Modelling Spatial Relations*. PhD thesis, INSA, LYON, 2009.

[26] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Unsupervised handwritten graphical symbol learning-using minimum description length principle on relational graph. In *Proceeding of the International Conference on Knowledge Discovery and Information Retrieval, KDIR*, 2011.

[27] Santosh K.C., Laurent Wendling, and Bart Lamiroy. Unified Pairwise Spatial Relations: An Application to Graphical Symbol Retrieval. In *Graphics Recognition. Achievements, Challenges, and Evolution*, pages 163–174. Springer Berlin / Heidelberg, 2010.

[28] Max J. Egenhofer. A formal definition of binary topological relationships. In *Third International Conference on Foundations of Data Organization and Algorithms (FODO)*, pages 457–472, 1989.

[29] François Bouteruche, Sébastien Macé, and Eric Anquetil. Fuzzy relative positioning for on-line handwritten stroke analysis. In *Tenth International Workshop on Frontiers in Handwriting Recognition (IWFHR'06)*, La Baule, France, October 2006.

[30] Adrien Delaye, Sébastien Macé, and Eric Anquetil. Modeling relative positioning of handwritten patterns. In *14th Biennial Conference of the International Graphonomics Society (IGS 2009)*, pages 152–156, 2009.

[31] Taik Heon Rhee and Jin Hyung Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42(12):3192 – 3201, 2009.

[32] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. Improving online handwritten mathematical expressions recognition with contextual modeling. In *International Conference on Frontiers in Handwriting Recognition*, pages 427–432, 2010.

[33] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Symbol knowledge extraction from a simple graphical language. In *International Conference on Document Analysis and Recognition*, 2011.

[34] V.J. Katz. *A history of mathematics: an introduction*. Addison-Wesley, 2009.

[35] D. P. Huttenlocher, G. A. Klanderman, and W. A. Rucklidge. Comparing Images Using the Hausdorff Distance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(9):850–863, 1993.

[36] Isabelle Bloch. Fuzzy relative position between objects in image processing: A morphological approach. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(7): 657–664, 1999.

[37] M. Egenhofer and J. Herring. *Categorizing Binary Topological Relationships Between Regions, Lines, and Points in Geographic Databases*. Department of Surveying Engineering, University of Maine, Orono, ME, 1991.

[38] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Quantify spatial relations to discover handwritten graphical symbols. In *Document Recognition and Retrieval XIX*, 2012.

[39] Guo Xian Tan, Christian Viard-Gaudin, and Alex C. Kot. Automatic writer identification framework for online handwritten documents using character prototypes. *Pattern Recogn.*, 42(12):3313–3323, 2009.

[40] T. Kohonen. Self-organization and associative memory. *Springer*, Berlin.

[41] Thomas Martinetz and Klaus Schulten. *A "neural gas" network learns topologies*. Elsevier, 1991.

[42] G. N. Lance and W. T. Williams. A General Theory of Classificatory Sorting Strategies: 1. Hierarchical Systems. *The Computer Journal*, 9(4):373–380, 1967.

[43] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. *Introduction to Information Retrieval.* Cambridge University Press, New York, NY, USA, 2008.

[44] Bartosz Broda and Wojciech Mazur. Evaluation of clustering algorithms for polish word sense disambiguation. In *International Multiconference on Computer Science and Information Technology (IMCSIT)*, pages 25–32, 2010.

[45] Marius Bulacu and Lambert Schomaker. Combining multiple features for text-independent writer identification and verification. In *In Proc. of 10th International Workshop on Frontiers in Handwriting Recognition*, pages 281–286, 2006.

[46] Marius Bulacu, Lambert Schomaker, and Axel Brink. Text-independent writer identification and verification on offline arabic handwriting. In *International Conference on Document Analysis and Recognition*, pages 769–773, 2007.

[47] Rajiv Jain and David Doermann. Offline Writer Identification using K-Adjacent Segments. In *International Conference on Document Analysis and Recognition*, pages 769–773, 2011.

[48] M.-P. Dubuisson and A.K. Jain. A modified Hausdorff distance for object matching. In *Computer Vision Image Processing*, 1994.

[49] Teuvo Kohonen and Panu Somervuo. Self-organizing maps of symbol strings. *Neurocomputing*, 21(1-3):19–30, 1998.

[50] Seiichi Uchida and Hiroaki Sakoe. A survey of elastic matching techniques for handwritten character recognition. *The Institute of Electronics, Information and Communication Engineers (IEICE) Transactions*, 88-D(8):1781–1790, 2005.

[51] E. Levin and R. Pieraccini. Dynamic planar warping for optical character recognition. *Acoustics, Speech, and Signal Processing, 1992. IEEE International Conference on*, 3:149–152 vol.3, 1992.

[52] Seiichi Uchida and Hiroaki Sakoe. Handwritten character recognition using monotonic and continuous two-dimensional warping. In *International Conference on Document Analysis and Recognition*, pages 499–502, 1999.

[53] Enrique Vidal Ruiz, Francisco Casacuberta, and Hector Rulot Segovia. Is the dtw "distance" really a metric? an algorithm reducing the number of dtw comparisons in isolated word recognition. *Speech Communication*, 4(4):333–344, 1985.

[54] Hiroyuki Narita, Yasumasa Sawamura, and Akira Hayashi. Learning a kernel matrix for time series data from dtw distances. *Neural Information Processing*, pages 336–345, 2008.

[55] Eitan Gurari. Backtracking algorithms. *CIS 680: DATA STRUCTURES: Chapter 19: Backtracking Algorithms*, 1999.

[56] Jinpeng Li, Harold Mouchère, and Christian Viard-Gaudin. Une distance entre deux ensembles de séquences avec la contrainte de continuité. In *Colloque International Francophone sur l'Ecrit et le Document (CIFED2010), Bordeaux, France*, 2012.

[57] Peter Hart, Nils Nilsson, and Bertram Raphael. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, February 1968.

[58] Bo Gun Park, Kyoung Mu Lee, and Sang Uk Lee. Color-based image retrieval using perceptually modified hausdorff distance. *J. Image Video Process.*, pages 4:1–4:10, January 2008.

[59] Ahmad Montaser Awal. *Reconnaissance de structures bidimensionnelles : application aux expressions mathématiques manuscrites en-ligne*. PhD thesis, Ecole polytechnique de l'université de Nantes, France, 2010.

[60] Ahmad Montaser Awal, Guihuan Feng, Harold Mouchere, and Christian Viard-Gaudin. First experiments on a new online handwritten flowchart database. In *Document Recognition and Retrieval XVIII*, 2011.

[61] Zhaoxin Chen. A dynamic time warping - a* handwriting recognition system. Master's thesis, Polytech'Nantes, 2012.

[62] Clark F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 1995.

[63] Zach Solan, David Horn, Eytan Ruppin, and Shimon Edelman. Unsupervised learning of natural languages. *Proceedings of the National Academy of Sciences of the United States of America*, 102(33):11629–11634, 2005.

[64] Noam Chomsky. Three models for the description of language. *IRE Transactions on Information Theory*, 2:113–124, 1956.

[65] Glenn Carroll, Glenn Carroll, Eugene Charniak, and Eugene Charniak. Two experiments on learning probabilistic dependency grammars from corpora. In *Working Notes of the Workshop Statistically-Based NLP Techniques*, pages 1–13. AAAI, 1992.

[66] E. Mark Gold. Language identification in the limit. *Information and Control*, 10(5):447–474, 1967.

[67] Salim Jouili and Salvatore Tabbone. Graph embedding using constant shift embedding. In *Proceedings of the 20th International conference on Recognizing patterns in signals, speech, images, and videos*, ICPR'10, pages 83–92, Berlin, Heidelberg, 2010. Springer-Verlag.

[68] G. Chartrand. *Introductory Graph Theory*. Dover Publications, 1985.

[69] J. C. Baird. *Psychophysical analysis of visual space*. Oxford, London: Pergamon Press, 1970.

[70] A. Jain, K. Nandakumar, and A. Ross. Score normalization in multimodal biometric systems. *Pattern Recognition*, 38(12):2270–2285, December 2005.

[71] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction To Algorithms*. MIT Press, 2001.

[72] Jianying Hu, Michael K. Brown, and William Turin. Hmm based on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18:1039–1045, 1996.

[73] C. C. Tappert, C. Y. Suen, and T. Wakahara. The state of the art in online handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808, 1990.

[74] A. Graves, S. Fernandez, J. Schmidhuber, M. Liwicki, and H. Bunke. Unconstrained online handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems 21*, 2007.

L'UNAM l'université nantes angers le mans
PÔLE DE RECHERCHE ET D'ENSEIGNEMENT SUPÉRIEUR

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Jinpeng Li

Extraction de connaissances symboliques et relationnelles appliquée aux tracés manuscrits structurés en-ligne

Symbol and Spatial Relation Knowledge Extraction Applied to On-Line Handwritten Scripts

Résumé

Notre travail porte sur l'extraction de connaissances sur des langages graphiques dont les symboles sont a priori inconnus. Nous formons l'hypothèse que l'observation d'une grande quantité de documents doit permettre de découvrir les symboles composant l'alphabet du langage considéré. La difficulté du problème réside dans la nature bidimensionnelle et manuscrite des langages graphiques étudiés. Nous nous plaçons dans le cadre de tracés en-ligne produit par des interfaces de saisie de type écrans tactiles, tableaux interactifs ou stylos électroniques. Le signal disponible est alors une trajectoire échantillonnée produisant une séquence de *traits*, eux-mêmes composés d'une séquence de points. Un symbole, élément de base de l'alphabet du langage, est donc composé d'un ensemble de *traits* possédant des propriétés structurelles et relationnelles spécifiques. L'extraction des symboles est réalisée par la découverte de sous-graphes répétitifs dans un graphe global modélisant les *traits* (nœuds) et leur relations spatiales (arcs) de l'ensemble des documents. Le principe de description de longueur minimum (MDL : Minimum Description Length) est mis en œuvre pour choisir les meilleurs représentants du lexique des symboles. Ces travaux ont été validés sur deux bases expérimentales. La première est une base d'expressions mathématiques simples, la seconde représente des graphiques de type organigramme. Sur ces bases, nous pouvons évaluer la qualité des symboles extraits et comparer à la vérité terrain. Enfin, nous nous sommes intéressés à la réduction de la tâche d'annotation d'une base en considérant à la fois les problématiques de segmentation et d'étiquetage des différents *traits*.

Mots clés

Langages graphiques, Extraction de connaissances symboliques, Exploration de graphes, Longueur de Description Minimale, Clustering, Dynamic Time Warping, Apprentissage de relations spatiales

Abstract

Our work concerns knowledge extraction from graphical languages whose symbols are a priori unknown. We are assuming that the observation of a large quantity of documents should allow to discover the symbols of the considered language. The difficulty of the problem is the two-dimensional and handwritten nature of the graphical languages that we are studying.
We are considering online handwriting produced by interfaces like touch-screens, interactive whiteboards or electronic pens. The signal is then available as a sampled trajectory of the pen or finger tip, producing a sequence of strokes, themselves composed of a sequence of points. A symbol, the basic element of the alphabet of the language, is composed of a set of strokes with specific structural and relational properties.
The extraction of symbols is performed by unveiling the presence of repetitive subgraphs in a global graph modeling the strokes (nodes) and their spatial relationships (arcs) of the entire document set. The principle of minimum description length (MDL) is used to select the best representatives of the symbol set.
This work was validated on two experimental datasets. The first one is a dataset of simple mathematical expressions, the second is composed of graphical flowcharts. On these datasets, we can assess the quality of the extracted symbols and compared them to the ground truth. Finally, we were interested in reducing the annotation workload of a database by considering both the problems of segmentation and labeling of the different strokes.

Key Words

Graphical languages, Symbol knowledge extraction, Graph Mining, Clustering, Minimum Description Length, Dynamic Time Warping, Spatial Relation Learning