Features and Algorithms for Visual Parsing of Handwritten Mathematical Expressions

by

Lei Hu

A dissertation submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in Computing and Information Sciences

B. Thomas Golisano College of Computing and
Information Sciences

Rochester Institute of Technology
Rochester, New York
May, 2016

Features and Algorithms for Visual Parsing of Handwritten Mathematical Expressions

by
Lei Hu

Committee Approval:
We, the undersigned committee members, certify that we have advised and/or supervised the candidate on the work described in this dissertation. We further certify that we have reviewed the dissertation manuscript and approve it in partial fulfillment of the requirements of the degree of Doctor of Philosophy in Computing and Information Sciences.

Dr. Richard Zanibbi Date
Dissertation Advisor

Dr. Nathan Cahill Date
Dissertation Committee Member

Dr. Harold Mouchère Date
Dissertation Committee Member

Dr. Linwei Wang Date
Dissertation Committee Member

Dr. Bo Yuan Date
Dissertation Committee Member

Dr. Daniel Phillips Date
Dissertation Defense Chair

Certified by:

Dr. Pengcheng Shi Date
Director, Computing and Information Sciences

Features and Algorithms for Visual Parsing of Handwritten Mathematical Expressions

by

Lei Hu

Abstract

Math expressions are an essential part of scientific documents. Handwritten math expressions recognition can benefit human-computer interaction especially in the education domain and is a critical part of document recognition and analysis.

Parsing the spatial arrangement of symbols is an essential part of math expression recognition. A variety of parsing techniques have been developed during the past three decades, and fall into two groups. The first group is graph-based parsing. It selects a path or sub-graph which obeys some rule to form a possible interpretation for the given expression. The second group is grammar driven parsing. Grammars and related parameters are defined manually for different tasks. The time complexity of these two groups parsing is high, and they often impose some strict constraints to reduce the computation.

The aim of this thesis is working towards building a straightforward and effective parser with as few constraints as possible. First, we propose using a line of sight graph for representing the layout of strokes and symbols in math expressions. It achieves higher F-score than other graph representations and reduces search space for parsing. Second, we modify the shape context feature with Parzen window density estimation. This feature set works well for symbol segmentation, symbol classification and symbol layout analysis. We get a higher symbol segmentation F-score than other systems on CROHME 2014 dataset. Finally, we develop a Maximum Spanning Tree (MST) based parser using Edmonds' algorithm, which extracts an MST from the directed line of sight graph in two passes: first symbols are segmented, and then symbols and spatial relationship are labeled. The time complexity of our MST-based parsing is lower than the time complexity of CYK parsing with context-free grammars. Also, our MST-based parsing obtains higher structure rate and expression rate than CYK parsing when symbol segmentation is accurate. Correct structure means we get the structure of the symbol layout tree correct, even though the label of the edge in the symbol layout tree might be wrong. The performance of our math expression recognition system with MST-based parsing is competitive on CROHME 2012 and 2014 datasets.

For future work, how to incorporate symbol classifier result and correct segmentation error in MST-based parsing needs more research.

Acknowledgements

First of all, I would like to thank my thesis advisor Dr. Richard Zanibbi, who gave me the opportunity to pursue my Ph.D in his lab and provided funding for my studies. In particular, I would like to thank him for his immeasurable amount of guidance, support and patience during my doctoral study.

Also, I would thank Dr. Nathan Cahill, Dr. Linwei Wang, Dr. Bo Yuan and Dr. Harold Mouchère for serving on my thesis committee and their helpful suggestions for improvements to this thesis.

I am grateful to Dr. Pengcheng Shi for his support and help during my study in RIT.

I would like to thank all members of the Document and Pattern Recognition Lab at Rochester Institute of Technology, especially Kenny Davila for sharing me his symbol classifier and proofreading my thesis; and Siyu Zhu for being a supportive officemate.

I want to say thank you to all of my friends, especially Dr. Hongda Mao for the support and encouragement during the thesis writing.

Last but not least, I would like to thank my parents, my grandparents, and the rest of my family for their unconditional love and support. They are always my source of strength.

This thesis is dedicated to my parents (Qingsong Hu and Yongfang Wang).

Contents

List of Tables

List of Figures

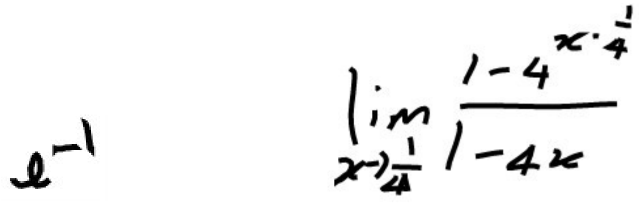
Chapter 1

Introduction

1.1 Statement

Math expressions are an essential part of scientific documents. Handwritten math expressions recognition can benefit human interaction especially in the education domain and is a critical part of document recognition and analysis [89]. The result of math expression recognition can be used for further retrieval or help recognizing and analyzing the structure and contend of the related document.

I will focus on the online data in this thesis. The motivation of this research is to make people can write math expressions on the electronic tablet and let the computer recognize them automatically. The expression can be online or offline (image) [18, 89]. The online input data is a set of strokes; each stroke consists of some points which are recorded in time sequence. Figure 1.1 shows two expressions.



(a) simple expression (b) complicated expression

Figure 1.1: Two expressions.

Online math expression recognition has attracted more and more attention recently, and there have been some competitions (Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME) 2011 [56], CROHME 2012 [57], CROHME 2013 [59], CROHME 2014 [58]) about online math expression recognition. These competitions publish publicly available datasets and performance evaluation libraries.

Math expression recognition usually consists of three parts: symbol segmentation, symbol recognition and layout analysis [18, 89]. Figure 1.2 shows the overview of a feed-forward math expression recognition system (not all systems are feed-forward). The segmentation segments the set of strokes into a symbol candidate list. Then classification assigns a class label to each symbol candidate. With the class label and location information of each symbol, layout analysis recovers the symbol layout of the math expression.

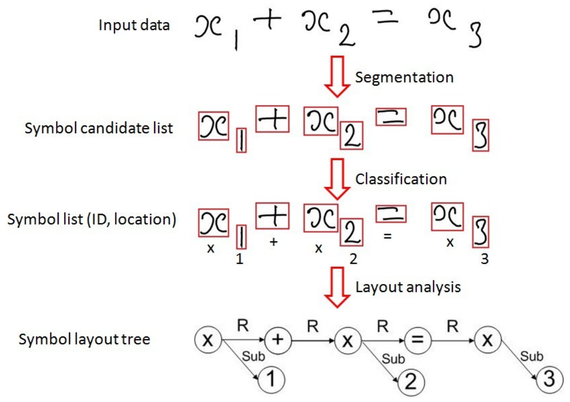
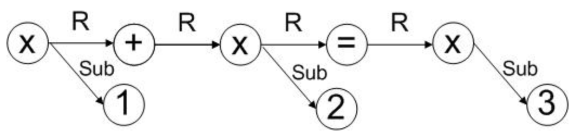
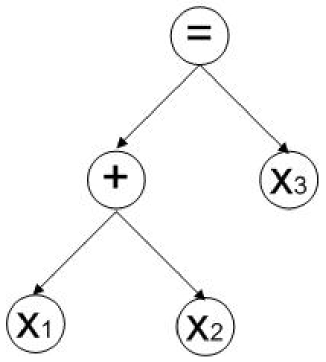


Figure 1.2: The overview of recognition of mathematical expression.

The output of math expression recognition system could be layout tree (Figure 1.3a) or operator tree (Figure 1.3b). Layout tree indicates the spatial relationship (such as horizontally adjacent, superscript, above, below, and inside) between symbol pairs. Getting operator tree requires mathematical content analysis after getting the symbol layout tree. For Figure 1.3 (a), content analysis interprets x_1, x_2, x_3 as variables and $+, =$ as operators. Content analysis also has to recover the math syntax and semantics of the expression, and use them to build the operator tree for the expression.



(a) symbol layout tree



(b) operator tree

Figure 1.3: Symbol layout tree and operator tree for $x_1 + x_2 = x_3$.

1.2 Research Questions

In this thesis, we would answer four main research questions for math expression recognition: (1) Which graph representation is good for math expression? (2) Are visual features enough for symbol segmentation, symbol classification and layout analysis? (3) Is structured learning a good direction? (4) Does parsing need a lot of prior human knowledge?

For a given math expression, people could have a rough idea of its structure after a glimpse even he/she does not know the math symbols at all, and the structure of math expression contains a lot of information about its meaning. Because for math expression not only the math symbols contain information, but also the way in which the math symbols belong together. But for a given text, no matter what language it is written, people will have no idea about the text if he/she does not know the words or characters. Because all the text is one-dimensional, and the structure of the text does not contain much information about the text.

Therefore, we can summarize the above four questions as one: is what you see is what you get true for math expression recognition?

1.2.1 Graph Representation

Before answering the other three questions, a good graph representation is necessary. Because math expression recognition could be taken as selecting a tree from a graph in which the node represents stroke while edge represents stroke pair relation. We did experiments with many different graph representations and the combinations of them (time series, K-nearest neighbor (KNN), maximum spanning tree (MST), line of sight (LOS), Delaunay triangulation). We wanted to find good graph representation for math expression with high recall and reasonable precision, where

$$precision = \frac{\#\text{ edge in (ground truth symbol layout tree} \cap \text{graph representation)}}{\#\text{ edge in graph representation}} \tag{1.1}$$

$$recall = \frac{\#\text{ edge in (ground truth symbol layout tree} \cap \text{graph representation)}}{\#\text{ edge in ground truth symbol layout tree}} \tag{1.2}$$

We found that line of sight graph with convex hull achieved the best F-score. F-score is the harmonic mean of precision and recall. Because it represents the visual structure of the math expression.

1.2.2 Visual Feature

Features are needed before any pattern recognition algorithm. For the three subproblems (symbol segmentation, symbol classification, symbol layout analysis) of math expression recognition, there are a lot manually designed features (mainly manually designed geometric features). What we wanted to know is there any visual feature could compete with the manually designed feature in the three different tasks. We propose Parzen window shape context feature and use it for symbol segmentation, symbol classification, symbol layout analysis with different classifiers.

We found that (1) Parzen window shape context feature performs better than original shape context feature for symbol segmentation and symbol layout analysis; (2) shape context feature could get competitive results in the three tasks against different manually designed features; (3) shape context features can be used as complementary information for different feature sets in the three tasks to improve performance; (4) syntax shape context feature which is calculated based on the symbol's syntax label and MST shape context feature which is calculated based on the edge in symbol layout tree could improve symbol segmentation and symbol layout analysis, but they require high accuracy about the syntax label and edge label.

1.2.3 Structured Learning

In contrast with conventional supervised learning such as classification, where input data (instances) are mapped to atomic labels and regression, where inputs are mapped to scalar numbers or vectors, structured learning is concerned with computer programs that learn to map input data to structured outputs (graphs).

Math expression is structured data which consists of several parts, and not only the parts themselves (symbol labels) contain information, but also the way in which the parts belong together (spatial relations). The output of math expression recognition system is symbol layout tree. So math expression recognition is a perfect vehicle for studying structured learning.

We wanted to know is structured learning a possible solution for math expression recognition? To the best of our knowledge, nobody has used structured learning for math expression recognition before. We used structured Boosting to combine stroke-level parsers sequentially, and used Random Forest to combine parallel symbol-level parsers.

Structured AdaBoost

Structured AdaBoost achieved success in producing dependency tree in natural language processing (NLP). We adapted the Structured AdaBoost for math expression recognition, and tested different weak classifiers and different weight updating schemes. We found the performance could be improved as the iteration increases sometimes during the Structured AdaBoost. But the final performance of Structured AdaBoost is not better than the performance without Structured AdaBoost. The reason could be that math expression recognition is more complex than NLP. There are eight classes for math expression recognition while there are only two for NLP. While Structured AdaBoost needs to deal with both segmentation and parsing in math expression recognition but only needs to solve parsing in NLP, because entities to parse are given.

Holistic Recognition

Structured Boosting combines stroke-level parsers sequentially. It is natural to think about combining ensemble parsers parallel as an alternative. We trained many different independent parselets based on decision tree. Each parselet does holistic recognition and produces a symbol layout tree. Holistic recognition means math expression is recognized as a whole. The recognition results produced by each parselet are combined to generate the final recognition result In our case, when the number of parselets is less than 80, the more parselets, the better performance. When the number of parselets is larger than 80, increasing the number of parselets cannot improve the performance. The performance of parselets combination is based on the performance of single parselet. The best structure rate is close to 50%. The performance is promising and it shows that holistic recognition is highly possible a right direction for math expression recognition.

1.2.4 Maximum Spanning Tree (MST) Based Parsing

The majority of current math expression recognition systems are based on the Cocke Younger Kasami (CYK) parsing. CYK parsing operates on context-free grammars given in Chomsky normal form (CNF). It determines whether a string can be derived from the grammar or not, and could produce the parsing tree if the string could be generated by the grammar. Production rules in context-free grammar need to be designed carefully by people for different datasets. Sometimes the number of rules could be hundreds. What we wanted to know is there any simple and intuitive parsing method could get competitive performance, but require much less human's tweak?

We proposed MST-based parsing with Edmonds' algorithm. It has better time complexity than the CYK parsing. We found the parsing performance of MST-based parsing is better than CYK parsing when the symbol segmentation is perfect.

1.3 Summary and Contributions

The short answer for the four questions is: what you see is what you get is true for math expression recognition. The contributions of the thesis are:

(1) We propose line of sight graph for math expression representation. Line of sight graph achieves the highest F-score among different graph representations with recall higher than 99%, as it represents the visual structure of the math expression.

(2) We propose Parzen window shape context feature and find Parzen window shape context feature can get competitive results in symbol segmentation [33], symbol classification and symbol layout analysis against other features. Using Parzen window shape context feature and line of sight graph gets the higher symbol segmentation F-score (92.43%) than other systems on CROHME 2014 test set.

(3) We propose MST-based parsing with Edmonds' algorithm. Its complexity is lower than the time complexity of CYK parsing based on context-free grammar. Its parsing performance (structure rate of 72.63% and expression rate of 67.70% on CROHME 2012 test set) is more than 15% higher structure rate and 10% higher expression rate than the CYK parsing when the segmentation is correct. Our math expression recognition system with MST-based parsing achieves very competitive performance (expression rate of 33.74%/26.88%) on CROHME 2012/2014 test sets.

(4) We explore structured learning on math expression recognition for the first time and find holistic recognition is a good direction for math expression recognition.

Chapter 2

Handwritten Math Expression Recognition

We first introduce different ways to represent math expressions in Section 2.1. Then the three key problems (symbol segmentation, symbol classification and layout analysis) of math expression recognition are discussed in Sections 2.2 to 2.4. The three problems are addressed in sequence in most systems. Some systems integrate two or three of these key problems and optimize the solutions jointly. The integration is discussed in Section 2.5.

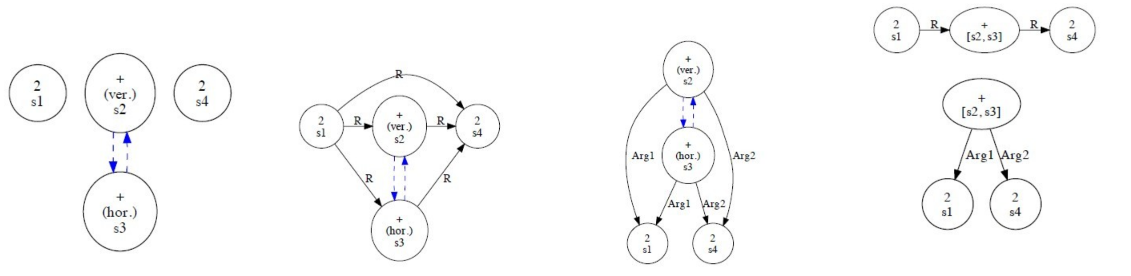
2.1 Math Representations

Math expressions can be represented in many ways, such as Tex, Mathematical Markup Language (MathML), symbol layout tree, operator tree, label graph and so on. MathML is an application of XML for describing mathematical notations and capturing both its structure and content. It aims at integrating mathematical formulae into World Wide Web pages and other documents. Both Tex and MathML represent math expression as a string. For expression $\phi(x)$, its Tex representation is '\phi(x)' and its MathML representation is

```
<mrow>
  <mi xml:id="phi_1">\phi</mi>
  <mrow>
    <mo xml:id="(_1">(</mo>
    <mrow>
      <mi xml:id="x_1">x</mi>
      <mo xml:id=")_1">)</mo>
    </mrow>
  </mrow>
</mrow>
```

Symbol layout tree and operator tree are introduced in Section 1. Zanibbi et al. [91] define label graph as directed graphs over primitives and primitive pairs. Primitive can be stroke here. In a label graph, nodes represent primitives and edges represent relationship between primitive pairs (the relationship can be segmentation and spatial relationships). Figure 2.1 shows the label graph, symbol layout tree and operator tree for expression $2+2$. (a)-(c) are primitive label graphs, while (d) shows symbol layout tree and operator tree. The expression has 4 stroke and they are named in writing order as s1, s2, s3 and s4. Symbol $+$ have two strokes represented by (ver.) and (hor.) Dashed edges represent strokes merged into a symbol. The label of the node is the class of the symbol to which the stroke belongs. The edge R represents the spatial relationship adjacent at right. Edge Arg1 and Arg2 indicate operator arguments.

The label graph can be represented as adjacency matrix. Figure 2.2 shows the adjacency matrices for label graphs from Figure 2.1. Figure 2.2a shows the format. The element l_i on the main diagonal is the symbol class label for the i-th stroke. The off-diagonal element e_{ij} $(i \neq j)$ represent the relationship between i-th stroke and j-th stroke. Figure 2.2b-d are the adjacency matrices for the label graphs in Figure 2.1a-c. '1' and '2' in Figure 2.2 represent 'Arg1' and 'Arg2'. Underscore '-' represents no relationship between the stroke pair, while asterisk '*' indicate the stroke pair belong to the same symbol.



(a) Symbols (b) Layout (c) Syntax (d) Layout and Operator Trees

Figure 2.1: Primitive label graph, symbol layout tree and operator tree for $2 + 2$ [91].

$$\begin{array}{c|cccc|} s1 & \mathbf{l1} & e12 & e13 & e14 \\ s2 & e21 & \mathbf{l2} & e23 & e24 \\ s3 & e31 & e32 & \mathbf{l3} & e34 \\ s4 & e41 & e42 & e43 & \mathbf{l4} \end{array} \quad \begin{vmatrix} \mathbf{2} & - & - & - \\ - & + & * & - \\ - & * & + & - \\ - & - & - & \mathbf{2} \end{vmatrix} \quad \begin{vmatrix} \mathbf{2} & R & R & R \\ - & + & * & R \\ - & * & + & R \\ - & - & - & \mathbf{2} \end{vmatrix} \quad \begin{vmatrix} \mathbf{2} & - & - & - \\ 1 & + & * & 2 \\ 2 & * & + & 2 \\ - & - & - & \mathbf{2} \end{vmatrix}$$

(a) Format (b) from Figure 2.1a (c) from Figure 2.1b (d) from Figure 2.1c

Figure 2.2: Adjacency matrices for label graphs [91]

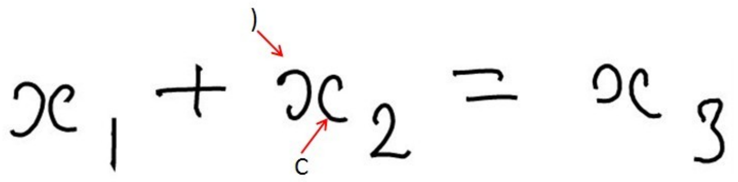
2.2 Symbol Segmentation

The input data for online handwritten expressions is a set of strokes, and a mathematical symbol may contain more than one stroke. Symbol segmentation aims to transform strokes into a set of symbols [16]. It is the basis of symbol classification and structural analysis, therefore the quality of symbol segmentation strongly affects the quality of the whole math expression recognition system.

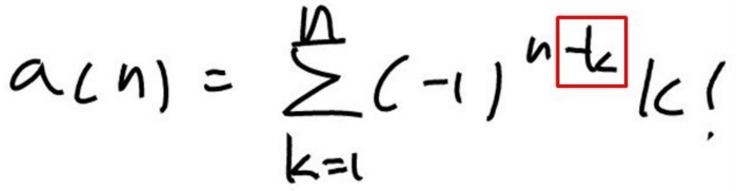
The main challenge of symbol segmentation is that there are no obvious rules to group the strokes which belong to a symbol. Furthermore, mathematical expressions are in two-dimensional structure and handwritten mathematical expressions may contain many heavily-overlapping symbols which would further complicate the symbol segmentation process. Heavily-overlapping symbols are symbols whose bounding box has a big overlap with other symbol's bounding box. Symbol segmentation would be more difficult when delayed strokes are present [8]. Delayed stroke is usually far away from its brother strokes (strokes from the same symbol) in time sequence. For example, some people write the dot of i after they finish writing all the other parts of the expression. In this case, dot is a delayed stroke.

There are two types of error in the symbol segmentation phase: over-segmentation and under-segmentation [76]. over-segmentation segments one symbol as multiple symbols, as showed in Figure 2.3 (a) (the symbol x is segmented into two symbols, and the left stroke is taken as) while the right stroke is taken as C). under-segmentation take multiple symbols as one symbol, as showed in Figure 2.3 (b) (two symbols - and k are taken as one symbol). Figure 2.3 shows both over-segmentation and under-segmentation would cause classification error.

In addition, a segment can be both over-segmented and under-segmented. For example, if the 'c' shape stroke of the rightmost 'k' in Figure 2.3 (b) and '!' are taken as a symbol candidate. There is over-segmentation as two strokes of 'k' are assigned into different symbol candidates; and there is under-segmentation as strokes from different symbols 'k' and '!' are taken as one symbol candidate.



(a) over-segmentation (x is interpreted as) and C)



(b) under-segmentation (- and k are interpreted as one symbol)

Figure 2.3: Two types of segmentation errors

A number of approaches have been proposed for mathematical symbol segmentation.

2.2.1 Methods based on X-Y cut

One group of methods are based on X-Y cut. X-Y cut was proposed by Nagy and Seth [61] at first to cut the document in the vertical and horizontal directions alternately.

Faure and Wang [26] propose a segmentation method based on a modular system which contains two modules: data-driven segmentation module and knowledge-driven segmentation module. In data-driven module, projection on the X and Y axes will be used to segment the strokes. Based on the segmentation, the relation tree will be formed. The relation tree represents the hierarchical structure of the expression. The project does not work for the symbols, such as $\sqrt{}$ and fraction. A mask removal with symbol recognition is used to separate those symbols and their embedded symbols before projection. Knowledge-driven module (knowledge is the heuristics related to the domain of the processed data) is then used to correct the errors in the relation tree.

Okamoto et al. [63] present a segmentation method based on recursive projection profile cutting. This segmentation method leads to over-segmentation. Then the over-segmented symbols are combined based on some rules and segment connectivity matrix. For each segment, segment connectivity matrix provides some other segments could be connected with the segment.

Ha et al. [30] propose a recursive X-Y cut segmentation method for printed expressions based on a top-down process and a bottom-up (merging) process. The top-down process used recursive X-Y cut to cut the expression to connected component level. For each primitive, it has its label and its bounding box information. The classifier is based on binary decision tree. Bottom-up process tries to split or merge the nodes of the expression tree based on the spatial relationship and syntax rules, such as the fraction bar '-' has to have both numerator and denominator.

X-Y cut or projection methods work well for printed expressions but not for handwritten ones, because handwritten expressions have more variation, and it is hard to segment them well just based on gaps along horizontal or vertical directions. Figure 2.4 shows a failure example of X-Y cut. There is no vertical gap between n and - of $\sum$ or no horizontal gap between - and k. Therefore there is no cut to separate n and - of $\sum$ or - and k. These produce at least two under-segmentations.

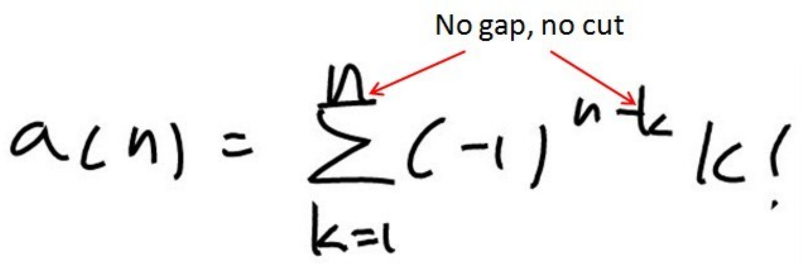


Figure 2.4: Failure example of X-Y cut

2.2.2 Graph-based methods

There have been many graph-based methods [53, 76].

Toyozumi et al. [76] present a symbol segmentation method for handwritten expressions based on candidate character lattice method. Evaluation values are calculated based on the positional relation of two strokes and math structure information. Nearest point distance between two strokes is used to get the evaluation value for stroke positional relation. Spatial grammars are defined on structure symbols (such as $-$, $\sqrt{}$, $\sum$ and so on) and stroke patterns which satisfy the grammars will be counted up to get the probability of existence of structure symbols. Feedback from symbol classifier is used to improve the segmentation rate.

Matsakis [53] proposes a segmentation method based on minimum spanning tree (MST). For the given expression, it will form an MST over the strokes. In the MST each node represents a stroke, and the distance between any two strokes is the Euclidean distance between the centers of the bounding boxes. The segmentation method only considers partitions that form connected subtrees in the MST. This means if there is no edge between two strokes in the MST, then there is no symbol which contains and only contains these two strokes. Figure 2.5 shows an example of segmentation by using MST. For the expression in Figure

2.5, the strokes 2 and - are prevented from being a single symbol because there is no edge between the two strokes. The partition criterion function is a cost function. The cost of a partition is the sum of costs of the symbols identified by that partition. To counteract the tendency to prefer partitions with fewer symbols, the combination weighting is added to the cost of multiple stroke symbols. It is a parameter for determining how easily the system will combine strokes into multiple stroke symbols. The lower the weighting, the easier it is for strokes to be combined. $cost(symbol) = score_1 + (N - 1) * CW$, where N is the number of strokes in the symbol, CW is combination weighting, and $score_1$ is the log likelihood produced by the symbol classifier.

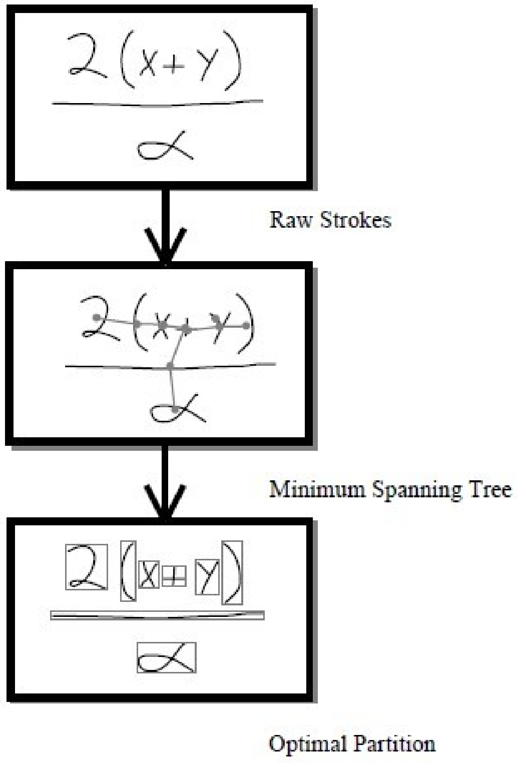


Figure 2.5: An example of segmentation by using MST [53].

2.2.3 Others

There have also been many other methods. Smithies et al. [72] present a progressive segmentation method. It has the time sequential assumption. For 4 strokes, the segmenter generates all possible groupings. Then the first recognized character from the group with highest confidence level will be removed and the process will restart when there is 4 strokes again. Kosmala et al. [40] propose a segmentation method based on HMM. Discrete left to right HMMs without skips and with different numbers of states are used. An addition space model is introduced to model the spaces between symbols.

2.2.4 Summary

Table 2.1 shows the comparison of the existing methods for segmentation of handwritten math expressions.

X-Y cut is not a good candidate for handwritten math expression segmentation because it is often that there is no gap between two strokes from different symbols.

To reduce the segmentation error, some effective heuristics are commonly used. From Table 2.1, we can find most of methods [26, 40, 53, 72, 76] use classification to help segmentation. Some methods [76] use grammars to help segmentation. Some methods [2] merge touched strokes where touched strokes are the strokes have more than one intersection point.

In order to reduce the number of possible symbol candidate and make the segmentation fast, some constraints are used to delete some symbol candidates. From Table 2.1, we can find some systems [40, 72] use time constraint and only deal with time consecutive stroke pair. Some systems specify user's writing order. The system [40] require user writes from left to right and from top to bottom. Some systems [53] have space constraint: only connected subtrees in the Minimum Spanning Tree (MST) over strokes can form the symbol candidates. Some systems [9, 68, 72, 76, 86] specify a symbol at most has 4 or 5 strokes.

Time constraint, space constraint and writing constraint limit system's generalization ability. Our system uses as few constraints as possible to achieve good generalization ability while not constrain user's writing style.

Table 2.1: Existing Methods for Segmentation of Handwritten Math Expressions

Paper	**Segmentation Method**	**Heuristics**	**Constraints**
Shi et al. [68]	Bayesian Framework	segmentation scores (used as evaluation values)	symbol has at most 4 strokes, time consecutive constraint
Toyozumi et al. [76]	graph-based method	classification scores (used as evaluation values) and spatial grammars	symbol has at most 4 strokes
Winker et al. [46]	graph-based method	segmentation scores, classification scores (both scores are used as evaluation values)	symbol has at most 4 strokes, time consecutive constraint
Kosmala et al. [40]	HMM	operator range	time consecutive constraint, writing from left to right and from top to bottom
Matsakis [53]	MST-based method	classification scores (used as evaluation values)	only subtrees of the MST can form a symbol
Smithies et al. [72]	progressive segmentation	classification scores (used as evaluation values)	symbol has at most 4 strokes, time consecutive constraint
MacLean et al. [48]	rectangular partition	ordering assumption and fuzzy grammars	restriction to rectangular partitions of strokes
Alvaro et al. [2]	stochastic context-free grammars based method	stochastic context-free grammars, merge touched strokes	only consider strokes located in the given rectangular region

2.3 Symbol Classification

In many math expression recognition systems, structural analysis assumes that the symbol recognition is error-free [15]. In some other math expression recognition systems, structural analysis uses top 5-10 symbol classification results. Therefore a symbol classifier with high accuracy is essential to a good math expression recognition system.

Recognition of handwritten mathematical symbols is very difficult due to the following features of handwritten mathematical symbols:

(1) The symbol set contains a large number of symbols (roman letters, Arabic digits, Greek letters, and Operator symbol).

(2) The same symbol could differ a lot with different typeface (normal, bold, italic) and different font size (subscripts, superscripts, limit expression).

(3) Many operators, such as integration, summation, product, fraction and so on have a wide range of possible scales.

(4) Different writers have different writing styles.

A number of approaches have been proposed for online handwritten mathematical symbol recognition. Table 2.2 lists the different symbol classifiers and the corresponding features.

A number of methods use nearest neighbor classifier. Smithies et al. [72] proposed a fast user-trained algorithm based on nearest neighbor classification in a feature space of approximately 50 dimensions. Vuong et al. [80] proposed an extended elastic matching algorithm. Elastic matching is achieved through calculating the minimum distance between the template symbol and input symbol with dynamic programming. During the matching process, every point of the input symbol is matched against that in the template symbol. Apart from Euclidean distance between points, the extended elastic matching algorithm also considers slope and curvature information during its matching process. MacLean et al. [47] presented a greedy approximate solution to the dynamic time warping algorithm for recognizing single-stroke symbols and the time complexity of the algorithm is linear.

There have also been many rule-based methods. Fitzgerald et al. [27] used fuzzy logic to extract features, such as Line, C-shape and O-shape, and classify symbols. In symbol recognition phase, the system uses

Table 2.2: Existing Methods and Features for Classification of Handwritten Math Symbols

Paper	**Classifier**	**Feature**
Smithies et al. [72]	nearest neighbor	50 features (details are missing)
Vuong et al. [80]	nearest neighbor	distance between points, slope and curvature
Belaid et al. [12]	rule-based classifier	curvature, direction and drawing length
Fitzgerald et al. [27]	rule-based classifier	Line, C-shape and O-shape
Matsakis [53]	quadratic discriminant classifier	15 PCA components from stroke data
Winkler et al. [84]	HMM	online features: local position, sine and cosine of slope, pen up-down; offline feature: different grids mapping
Alvaro et. al [2]	HMM	15 online features: normalized horizontal and vertical position, normalized first derivatives, second derivatives, curvature, and the four lowest frequency Fourier components; offline feature: normalized gray level, horizontal gray-level derivative, vertical gray-level derivative
Alvaro et. al [1]	recurrent neural network	15 online features: normalized horizontal and vertical position, normalized first derivatives, second derivatives, curvature; offline feature: PCA components of gray-scale values
Awal et. al [8]	multi layer perceptron	7 online features for 50 points (details are not published)
Awal et. al [9]	time delayed neural network (TDNN)	7 online features for 50 points (details are not published)
Davila et. al [23]	AdaBoost.M1 with C4.5 decision trees, Random Forests and Support Vector Machines with linear and Gaussian kernels	global features of the symbols, crossing features, 2D fuzzy histograms of points, and fuzzy histograms of orientations

two types of fuzzy rules: high-level rules and low-level rules. High-level rules define the properties the input symbol must have if it belongs to a particular class. Low-level rules assess the extent to which these properties are present. Belaid et al. [12] proposed an approach based on decision tree classifier. The non-leaf nodes in the decision tree are the set of rules to classify the input symbol. Curvature, direction and drawing length are used as features.

There are several mathematical symbol recognition systems based on statistical approach. These systems establish a model for each symbol class. When a candidate symbol is input, all the models will calculate the likelihood. The model with highest likelihood will give its class label to the input symbol. Matsakis [53] presented a symbol recognition method based on a quadratic discriminant classifier. The preprocessing first uses stroke reversal, stroke ordering, scaling, shifting and resampling to convert a set of strokes in the form of ordered (x, y) coordinates into a single vector in a 74 dimensional space, and then uses Principal Component Analysis (PCA) to reduce the dimension from 74 to 15. For each symbol class, the projected data points are used to estimate the mean (μ) and covariance matrix ($\sum$) for a Gaussian density in the PCA space in the training phase.

Another group of methods use Hidden Markov Model. The input data of online handwritten mathematical symbols is a sequence of points. Because speech is a sequence of acoustic signals and HMM has achieved great success in speech recognition. It is natural to think of using HMM to recognize online handwritten mathematical symbols. Winkler et al. [84] proposed a symbol recognition system based on HMM. Their system extracts both on-line features and off-line features. They builds three semi-continuous left to right HMMs for each symbol to combine the classification results. Two HMMs use the off-line features while one HMM uses the on-line features. The online features they used are the local position, the sine and cosine value of the angle between the horizontal axis and the vector connecting the previous and the current point, and the information whether the current point belongs to a stroke or to an interpolated hidden stroke. Alvaro et. al [2] presents a hybrid symbol classifier based on HMM. They extract both online features and offline features. Each of the two types of features is used to train one HMM. The final classification is determined by the output of the two types of HMMs.

Neural network is a popular classifier for math symbol recognition. Awal et. al [9] present a symbol classifier based on neural network. The classifier is not trained on the isolated symbol training data but in a global learning way. The classifier has a rejection class to reject the bad symbol hypothesis. The training is performed in an iterated way. At each iteration, the classifier will be trained and updated by using the training data (segments) produced by the classifier at the previous iteration. Aguilar et. al [37] present a Multi Layer Perceptron neural network based symbol classifier with rejection option. The classifier can identify wrong segmented symbols (false hypotheses). They propose a fuzzy shape context feature. They extend the shape context definition by considering bins as fuzzy sets. A point may belong up to four bins based on its position. The shape contexts are extracted at eight equally distributed points in time order. Therefore, the shape contexts are influenced by the writing order and writing direction. Alvaro et. al [1] present a online math symbol classifier based on recurrent neural network. They render the online symbol to an image and extract both online features and offline features. The experiment results show the recurrent neural network classifier outperforms the HMM classifier.

Garain et al. [28] presented a symbol recognition system to combine two different kinds of classifier. The first classifier employs a nearest neighbor classification and the second one uses a left to right HMM. Both the classifiers make use of direction change and trajectory length as features. The system combines the two classifiers through three ways: highest rank method, Borda count and logistic regression, and logistic regression gets the best performance.

Davila et al. [23] presented a math symbol classifier using adaptations of off-line features and synthetic data generation. An elastic distortion model is used to generate the synthetic data. The features used in this paper could be divided into four categories: global features of the symbols, crossing features, 2D fuzzy histograms of points, and fuzzy histograms of orientations. They compared the performance of four different classification methods: AdaBoost.M1 with C4.5 decision trees, Random Forests and Support Vector Machines with linear and Gaussian kernels. Experiment results show SVM with RBF Kernel outperforms the other classification methods and the generation of synthetic data for underrepresented classes might lead to improvements of the average per-class accuracy. We use Davila's classifier [23] in some of my experiments.

2.3.1 Summary

Most of symbol classifiers require a preprocessing step to reduce the noise and the variation of size. Handwritten math symbols have many variations, and it is extremely difficult to choose appropriate templates for each symbol class or specify the rules to describe each symbol class. Therefore, nearest neighbor classifier and rule-based classifier cannot deal with handwritten math symbol classification very well. HMM are commonly used recently and get satisfactory classification rate because it is good at dealing with time sequential information and a symbol consists of a sequence of points. At each state of HMM, it can use the information from previous state. This means for each point of the symbol, it can use the information from previous context. Alvaro et. al [1] shows the recurrent neural network classifier outperforms the HMM classifier. The reason is that when dealing with a point, recurrent neural network can use information from both previous points and latter points while HMM can only use the information from previous points.

As for features, the coordinates, slope, curvature and pen up-down are commonly used and effective online features. For the same symbol, people can write it with different stroke order or different stroke direction. This will make the online features not consistent sometimes. Some systems [1, 2, 84] render the online symbol to an image and extraction the offline features besides the online features. Experiment results [1,2]show the combination of online and offline features can get higher classification rate than online features.

2.4 Layout Analysis

Layout analysis aims to recover the symbol structure of the math expression [89]. Spatial structure of math expression is usually represented as a symbol layout tree (Figure 1.3a). Symbol layout tree indicates the spatial relationship (such as horizontally adjacent, superscript, above, below, and inside) between symbol pairs. Layout analysis is challenging and a number of methods have been proposed to recover symbol layout.

2.4.1 Spatial relationship classifier

A subproblem of layout analysis is classifying the spatial relationship between the symbol pair, or between symbol and subexpression, or even between subexpression pair. Table 2.3 compares the existing methods for spatial relationship classification of handwritten math expression. Some methods [5, 32] only classify horizontal relationship (R, Sub, Sup). Most of the methods [3, 6, 48, 69] address all the horizontal, vertical and inside relationship. Some methods [3, 48] convert two classes (above, below) into one class (below) to reduce the class number for better performance.

Table 2.3: Existing Methods for Spatial Relationship classification. Relationship label is abbreviative: horizontal right adjacent (R), subscript (Sub), superscript (Sup), above (A), below (B), inside (I)

Paper	**Classifier**	**Features**	**Relation Types**
Anderson [6]	rule based classifier	minimum, maximum, central x and y coordinates of symbol bounding boxes	H, Sub, Sup, A, B and I
MacLean et al. [48]	rule based method	geometric features	H, Sub, Sup, B and I
Aly et al. [4, 5]	Bayesian classifier	relative size and relative position	R, Sub, and Sup (add A and B [4])
Awal et. al [9]	Bayesian classifier	baseline position and x-height	11 relations based on two or three elements (element can be symbol or sub-expression)
Simistira et al. [69]	SVM	6 geometric features	H, Sub, Sup, A, B and I
Alvaro and Zanibbi [3]	SVM	9 geometric features and polar histogram-based shape features	H, Sub, Sup, B and I

Aly et al. [5] present a method to classify the position relation between adjacent characters in a math expression into one of three classes: horizontal class, subscript class, and super script class. Relative size and relative position are the two features used in the classification. The classifier is a Bayesian classifier. Each of the three classes is assumed to a 2D Gaussian distribution in the 2D feature space. The dataset is InftyCDB-1 and InftyCDB-2. InftyCDB-1 [74] is a ground-truthed math character and symbol image database. It contains 467 pages of 30 pure math articles in English. For each character, the ground truth, such as type, font, quality, link and so on, is attached manually. All the characters belong to words or formula in the database. InftyCDB-2 has the same structure as InftyCDB-1 and is a continuation of InftyCDB-1. InftyCDB-2 not only contains English articles but also French articles and German articles. The class of each symbol and character is assumed to be known. For each symbol, it will be assigned to one of six types based on its X:Y:Z ratio. Different symbol has different bounding box normalization rule. Character size normalization is introduced to avoid the variation of character shapes. Irregular characters need to be processed with extra attention. They extend their work [4] and add two new relationship classes: upper and lower. There are 5 relationship classes in [4]: horizontal, superscript, subscript, above and below.

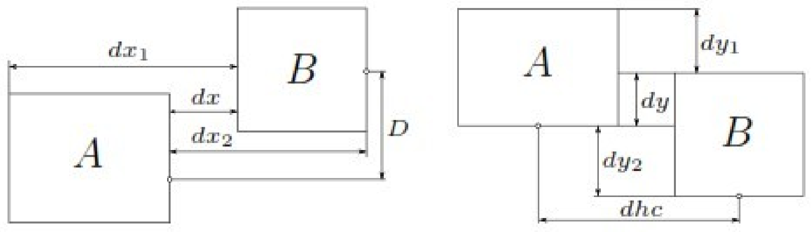
Simistira et al. [69] present a approach for the structural analysis between two on-line handwritten mathematical symbols based on six geometric features. The six geometric features are horizontal distance between left edges, right edges, centroids of the bounding boxes normalized by the maximum width of the two bounding boxes, the vertical distance between top edges, bottom edges, centroids normalized by the maximum height of the two bounding boxes. The six distances are $dx_1, dx_2, dhc, dy_1, dy_2, D$ as showed in Figure 2.6a. They compare two multiclass classification methods which employ support vector machine: one based on the one-against-one technique and the other based on the one-against-all. There are six types of spatial relationships: horizontal, superscript, subscript, above, below, and inside. Experiment results show one-against-one SVM outperform one-against-all SVM on the CROHME 2012 dataset [56]. The overall mean error rate are 6.57% for one-against-all SVM and 2.61% for one-against-one SVM.

Alvaro and Zanibbi [3] classify spatial relationships between symbols and subexpressions based on Support Vector Machine with Gaussian kernel. They present a new normalization for a set of geometric features and a novel set of shape-based features. The features are showed in Figure 2.6. Geometric features are extracted from the bounding boxes of the two symbols or subexpressions and the normalization factor is the distances between the centers of the bounding boxes. Geometric features are mainly about different horizontal and vertical distances between the two bounding boxes. The polar histogram-based shape features are similar to shape contexts and they are invariant under translation and scaling. Each bin has one of the three values (-1, 0, 1) based on the majority vote of the points within that bin (-1 if more points from left subexpression, 0 empty bin, 1 if tie or more points from right subexpression). PCA is only applied to the polar histogram-based shape features but not geometric features They use the MathBrush corpus [50] as dataset and there are five types of spatial relationships: horizontal, superscript, subscript, below, and inside. The experiment results show new polar histogram-based shape feature without information about symbol typographic categories (e.g. ascender) provides comparable results to geometric features with typographic categories information. The combination of both sets of features led to a small improvement in accuracy.

2.4.2 Graph grammars

Graph grammar provide a powerful formalism to describe structural manipulations of multi-dimensional data. A graph grammar is specified by a start graph and a set of production rules. Figure 2.7 shows an example of graph grammar rewriting. Graph g is converted into graph g' based on production rule r. Grammar r specifies that two nodes with label a and c can be converted into one node with label d if there is a directed edge from node a to node c. The grammar r also indicates that only edges outgoing from node a and edges going into node c should be kept shown by dashed nodes in Figure 2.7.

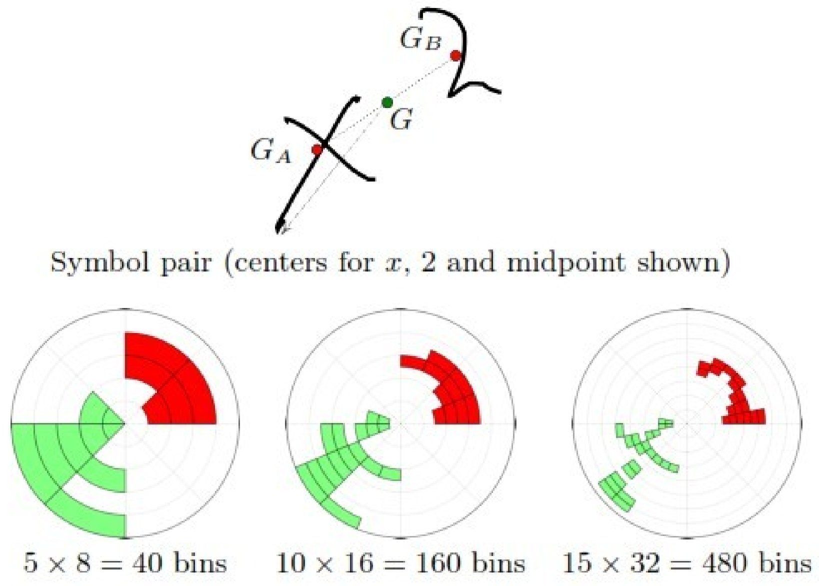
Grbavec et. al [29] present an offline math expression recognition system (EXPRESSO) based on graph rewriting. The system assumes the symbol recognition is error free and contains almost 60 rules. The input to the system is a discrete graph consisting of a labeled attributed node per symbol. Attributes contain the location and meaning of the corresponding subexpression. The output is a single node whose meaning attribute represents the meaning of the expression as a character string. The system has four phases. Build phases adds edges to represent relationship between symbols. Constrain phase applies notational conven-



$$H = \frac{\text{height}(B)}{F};\ D = \frac{\text{cen}_\text{v}(A) - \text{cen}_\text{v}(B)}{F};\ dhc = \frac{\text{cen}_\text{h}(A) - \text{cen}_\text{h}(B)}{F}$$

$$\text{features} = [H, D, dhc, \tfrac{dx}{F}, \tfrac{dx_1}{F}, \tfrac{dx_2}{F}, \tfrac{dy}{F}, \tfrac{dy_1}{F}, \tfrac{dy_2}{F}]$$

(a) geometric features from bounding boxes of subexpressions A and B using normalization F (F is the distances between the centers of the bounding boxes of the subexpressions)


G_B
G
G_A
Symbol pair (centers for x, 2 and midpoint shown)
$5 \times 8 = 40$ bins
$10 \times 16 = 160$ bins
$15 \times 32 = 480$ bins

(b) Varying distance (n) × angle (m) resolution in a polar histogram layout descriptor. Values shown using green (-1), red (+1), and white (0)

Figure 2.6: Geometric features and histogram-based shape features [3]

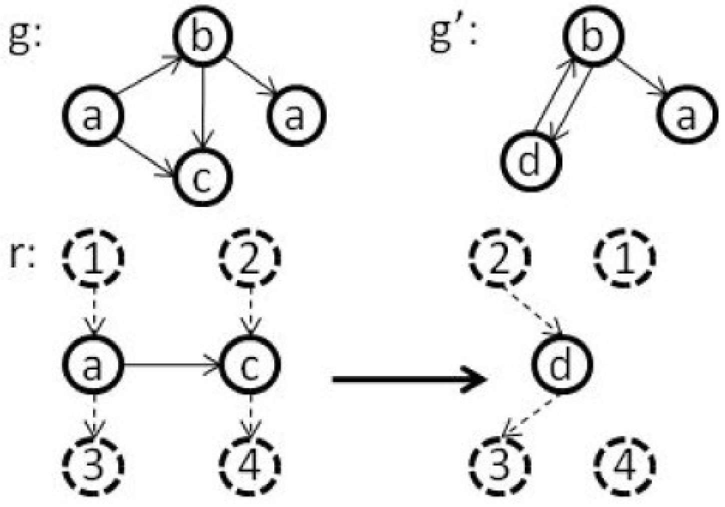


Figure 2.7: Application of rule r to graph g gives graph g' [17].

tions of math to remove contradiction and solve ambiguity. Rank phase uses operator precedence and range to group symbols into subexpressions. Incorporate phase interprets the subexpressions.

Lavirotte et. al [44] present a printed math expression recognition system based on graph grammar. The paper uses context-sensitive graph grammar and graph rewriting to solve the recognition problem. The parsing algorithm is a bottom-up algorithm and applies the first rule which can be applied in regards to the context. For each character, the OCR will produce the symbol label, coordinates of the bounding box, font size and reference of the character (baseline). With those information, the system builds a graph encoding relative positions of characters. A context dependent graph grammar is used to reduce the graph to an irreducible form. Then nodes contain abstract syntax trees of recognized expressions. Graph grammar is used to parse and generate graphs. Grammar rules rewrite the graph by replacing sub-graphs by vertices whose values are the recognized sub-expressions.

Kosmala et. al [41] use a graph grammar approach for the layout analysis. The recognition result is a parsing tree. Symbol label, lexical type, bounding box, approximate baseline and size are used for graph construction. It adapts the graph building method in [44] for online handwritten math expression and improves the graph rewriting method to make it faster.

Julca-Aguilar et. al [36] propose a top-down parsing based on graph grammar. The method models math expressions as languages generated by graph grammars, ant it contains three components: a context-free graph grammar, a symbol hypotheses relation graph (SHRG) and a top-down parsing algorithm. The graph grammar defines which math symbols, subexpressions and relations among them are valid. The SHRG defines the groups of strokes which will be evaluated to determine if they can be interpreted as a symbol or subexpression. The parsing starts by determining partitions of the complete set of strokes. Each partition is a instantiated graph. Then each node of each instantiated graph are further partitioned until getting a terminal symbol. The geometric mean of the relations and classification scores produced by the symbol and relations classifiers is used as the score for a parsing tree.

2.4.3 Stochastic context-free grammars (SCFG)

Figure 2.8a shows an example of 1-dimensional stochastic context-free grammar. A parse of a sentence is its derivation. The derivation of the beginning of the sentence in Figure 2.8a involves production rules $S \to AD, A \to BC, B \to b, C \to c$ and all these production rules do not need the context. The order of production rules is not important ($S \to AD \to BCD \to bCD \to bcD$ is a parse equivalent to $S \to AD \to BCD \to BcD \to bcD$). If a sentence has more than one parse, the grammar is said to be ambiguous. By assigning probabilities to each production rule can help solve the ambiguities. In addition, rule probabilities represent which patterns are more or less frequent in the language in a coarse way.

Chou [21] presents a recognition system for offline math expressions by using Cocke-Younger-Kasami (CYK) algorithm and 2-dimensional stochastic context-free grammar. The recognition method contains two steps: lexical access and parsing. Lexical access is based on template matching classification and produce a list of all character in the image. The list is passed to parsing. An extension of CYK algorithm is used to do the maximum likelihood parsing like Viterbi algorithm to HMM. 2-Dimensional versions of the

Inside/Outside and Baum re-estimation algorithms are used to learn the grammar parameters from training set. In this paper, the terminal symbols in grammar are pixels but not characters. The system takes the entire binary image as a sentence and the parse is a hierarchical structure all the way down to the pixel level.

Yamamoto et. al [87] present an online handwritten math expression recognition system based on SCFG at stroke level. This paper extends classical math expression grammar which are designed to generate strings representing expression to an expression grammar which models the stochastic generation of the 2D structure and the handwritten strokes. The recognition problem is converted to search for an expression which is derived from the expression from the expression grammar and that maximizes the production of all stroke likelihoods and structure likelihoods. The search problem is solved by the CYK algorithm. Each stroke is taken as a symbol and symbol likelihood is calculated as stroke likelihood. The structure likelihood is calculated based on geometric features. Experiments are conducted on their own dataset. Experiment results show segmentation and recognition rate increases along with the strengthening of the grammatical constraint.

Figure 2.8b [87] shows parsing a math expression by using CYK algorithm. For each stroke, stroke likelihood is calculated. The $i-th$ stroke and its likelihood is in the $i-th$ diagonal element of the CYK triangle matrix. How to get the order of each stroke is not mentioned. Then for the $i-th$ stroke, each element of $(i, i+1), (i, i+2), \cdots, (i, i+K)$ is acquired with all candidates and the corresponding likelihoods. Finally, the most likely candidate in the element $(1, n)$ (n is the number of the strokes in the expression) of the CYK matrix is the recognition result.

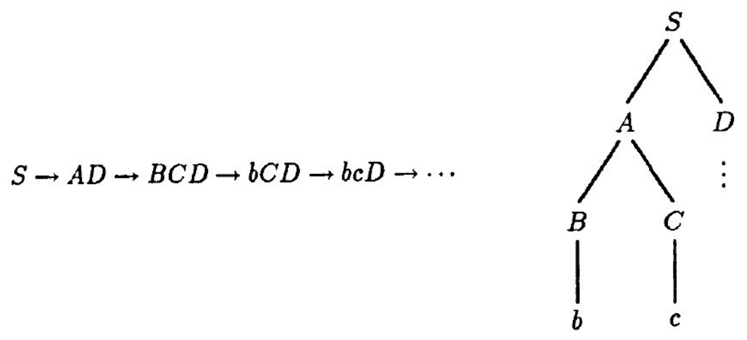
For example, the first and second strokes can be derived with the rule $x \rightarrow x_{[1]}x_{[2]}, S_{\text{SameSymbol}}$. x_1 in element $(1, 1)$ means the stroke) is taken as the 1st stroke of x, while x_2 in element $(2, 2)$ means the stroke (is taken as the 2nd stroke of x. The structure likelihood is 0.5, shown as in element $(1, 2)$ of the second row from the bottom of the pyramid. The candidate x and the product of stroke and structure likelihoods

$0.1 \times 0.1 \times 0.5 = 0.005$ is written in $(1,2)$ element of the matrix with the independence assumption. For the $(1,3)$ element in the matrix, x^y can be derived from x in element $(1,2)$ and y in element $(3,3)$. The structure likelihood that y is superscript of x is 0.6. The total likelihood of x^y in element $(1,3)$ is $0.005 \times 0.1 \times 0.6 = 0.00003$.

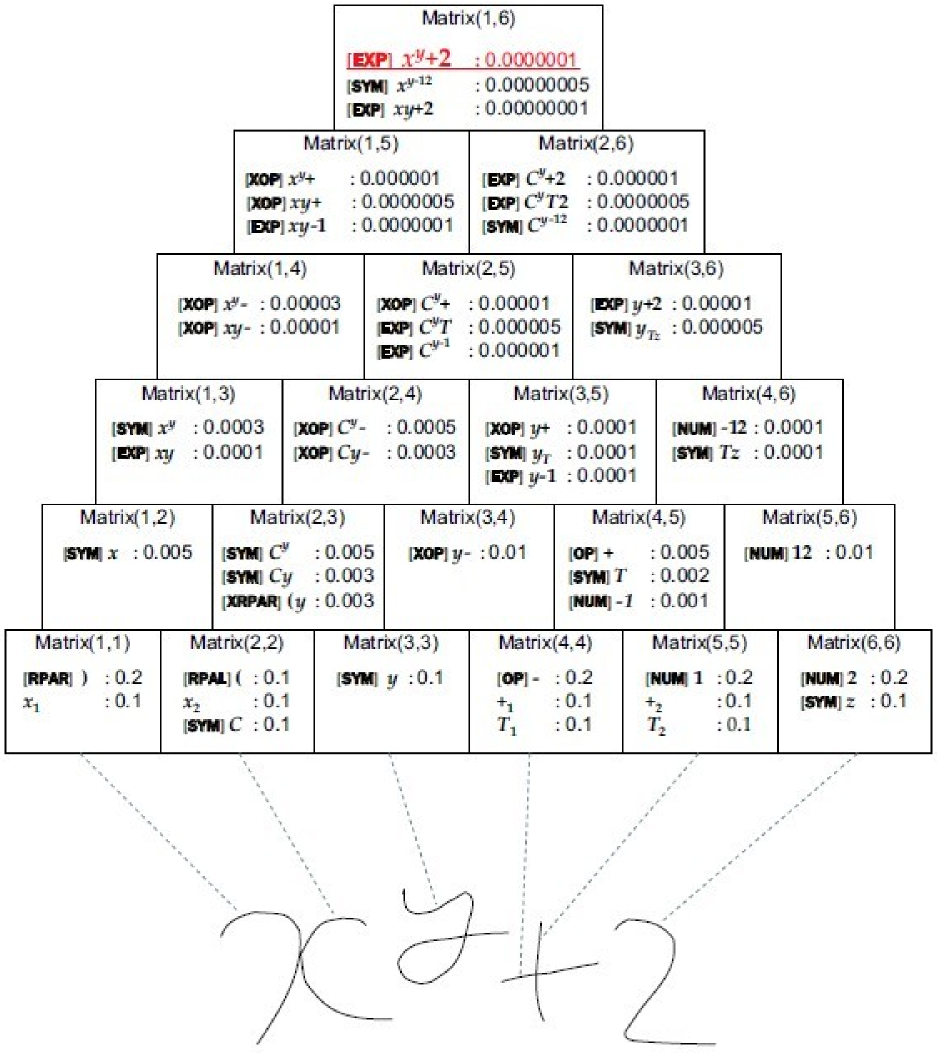
Similar to [87], Le et. al [45] also represents mathematical expressions in context-free Grammar at stroke level and employ CYK algorithm to do the parsing. The recognition becomes a search problem which finds a grammar driven expression with maximal production of segmentation score, symbol classification score, structure score and grammar score. They compute the separation probability of each pair of adjacent strokes and generate all symbol hypotheses of mathematical expressions in the segmentation process. Therefore, they have to prepare all particular writing orders such as those before and after fractions, roots, and parentheses in the grammar to avoid parsing failure. In symbol classification, they define specific rule for dot/comma. They divide symbols into four groups: ascendant, descendant, normal and big symbols before structural relation classification. Two SMVs are trained for 2 relation groups: (above, below, inside) and (horizontal, superscript, subscript). The grammars are designed manually. They mention that production rule ($X \rightarrow A$) are added to Chomsky Normal Form to make the grammar unambiguous, but details are missing.

Celik et. al [17] present a math expression recognition system based on 2D context-free probabilistic graph grammar at symbol level. The 2D context-free probabilistic graph grammar leads the system to find all mathematically valid interpretations and associate probabilities to each possible interpretation of the expression. The likelihood of an interpretation depends on the suitability of the symbols spatial distribution for the rules used and the likelihoods of the recognized symbols.

Alvaro et. al [2] present an online handwritten math expression recognition system based on SCFG at symbol level. The online handwritten expressions is rendered to image first. Then each connected compo-


$S \rightarrow AD \rightarrow BCD \rightarrow bCD \rightarrow bcD \rightarrow \cdots$
S
A
D
B
C
b
c

(a) a derivation and parse tree for a context-free sentence beginning bc... [21].



(b) example of a search for most likely expression candidate using the CYK algorithm [87].

Figure 2.8: Applying SCFG to math recognition.

nent will be taken as a symbol candidate and passed to the symbol classifier. Hidden Markov models are used to recognize mathematical symbols. The symbol classifiers provide multiple candidates and the final classification is done in the parsing phase. The paper [2] defines the 2D extension of SCFG and the corresponding version of the Cocke-Younger-Kasami (CYK) parsing algorithm. The SCFG determines whether to merge two symbol candidates or not based on the class label and class confidence produced by the symbol classifier. The probability associated with the grammar rule is statistics data of the training data. The 2D-CYK parsing contains two steps. First, the initialization begins building the set of basic units from the set of segmentation hypotheses. Next, the parsing process continues calculating new subproblems of increasing size, where both spatial and syntactic constraints are taken into account for each new subproblem. In the 2D-CYK parsing, there is a level for each subproblem size, and these levels store a set of elements which contain their two-dimensional space information.

Awal et. al [9] present a math expression recognition system based on a set of two dimensional context-free grammars. The two dimensional math expression grammars are combined by two sets of one dimensional grammars on horizontal and vertical direction. Each production rule in the grammars is associated to a spatial relation which describes the layout between the elements of the rule. Each relation is associated to a cost function. Two features: normalized position and size difference are used to build the model. Gaussian models are constructed for each element of a relation. Given the input expression, the hypothesis generator produces all possible symbol candidates which will be passed to symbol classifier and structure analyzer. Maximum number of hypothesis, maximum number of strokes per symbol and maximum distances between strokes forming the same hypothesis are limited. For each symbol candidate, a score is assigned based on the recognition cost and structural cost. The output of the recognition system is a relation tree. Relation tree is similar to operator tree. But its node can be a subexpression and the root of the relation tree represent the expression itself. Each relation tree has its cost which is calculated recursively based on the classifier probability and relation probability. The decision maker will choose the recognition result with minimum cost by using the language model which defines the grammars.

Simistira et. al [70] proposed a CYK based algorithm to parse the math expression. They used 7 geometrical features and trained a probabilistic SVM classifier to recognized spatial relations between two symbols or sub-expressions. This paper assumes the symbols have been correctly recognized and symbols are divided into 3 categories: ascenders, centered and descenders. In addition, the chronological order of symbols is used during the parsing.

2.4.4 Methods based on baseline extraction

A baseline is a list of symbols and each symbol is right adjacent to the previous symbol. Math expression is represented as a hierarchical structure of nested baseline.

Zanibbi et. al [90] propose a layout analysis method based on baseline extraction. Baseline extraction parses symbol layout by locating symbols on the main baseline from left to right, and then repeating this process in regions around baseline symbols recursively. The method is deterministic, and requires symbol bounding boxes and their typographic/layout classes (e.g. ascender, descender, centered, root, nonscripted) as input. To deal with ambiguous spatial relationships, fuzzy logic methods have been used to produce multiple interpretations [92]. Tapia and Rojas [75] present a parsing method based on baseline extraction. When extracting the baseline, the method looks for the leftmost dominant symbol, then the next leftmost dominant symbol and continues, until there is no more symbols are found. A function is defined to get the dominant symbol. Minimum spanning tree is used to help assign symbols which do not belong to the main baseline to the symbols on the main baseline. Some pen-based math entry systems [7, 71] are also based on baseline extraction.

2.4.5 Summary

Graph grammar-based methods convert the layout analysis into graph rewriting. In the graph, the node could either be a basic unit (a stroke or a symbol) or the whole expression. This makes the structure of the graph can be arbitrary and the number of graph could be huge. It is impossible to design a complete graph grammar set to cover all the possible situations. Therefore, graph grammar based methods become less and less popular.

Most graph grammars based methods and stochastic context-free grammars based methods are from bottom to top. For all the above stochastic context-free grammars-based methods, a symbol candidate generator produces multiple symbol candidates. The grammars are used to check the validity of proposed interpretations. The proposed interpretations are sorted based on the scores produced by the predefined cost function.

The similarity of the above stochastic context-free grammars-based methods is that they all use the CYK parsing algorithm. The difference is that the basic unit of the parsing algorithm is different. The basic unit is pixel in [21], stroke [87] in and symbol in [2, 9, 17]. The stochastic context-free grammars are also different because they all are designed manually for different dataset. Different systems have different methods to assign the probabilities to the production rules. Stochastic context-free grammars based methods are widely used now and get some good performance. Because the stochastic context-free grammars can incorporate the people's prior knowledge and heuristics well. The quality of the grammars is the key to these systems. Therefore, stochastic context-free grammars based methods cannot generalize well to different datasets or different tasks. People have to re-design the grammars for different datasets.

The idea of baseline extraction based methods is that math expression can be represented as a hierarchical structure of nested baseline. The advantage of these methods is that they are simple and effective. The relationship between two symbols on the same baseline can only be horizontal adjacent. Therefore finding the next right adjacent symbol precisely is critical to these methods. After getting one baseline, how to get the nested baselines are difficult and important. Different systems have different rules for the nested baselines. For the systems [32, 75] which have a main baseline, how to find the beginning symbol of the main baseline is challenging and important.

2.5 Integration

Symbol segmentation, symbol classification and layout analysis are interdependent [89], and context is often needed for disambiguation. Integration of the three modules can solve local ambiguity and has attracted extensive attention in the last ten years [89]. A number of approaches have been proposed to integrate the three tasks. Figure 2.8b is an example of this kind of method.

2.5.1 Language Models

For graph-based and grammar-based integration methods, they apply some language constraints to reduce the computation time complexity.

Some systems [39, 40, 68, 85, 86] use time constraint and only deal with time consecutive stroke pair or symbol pair. Some systems specify user's writing order. The system [40] requires user writes the math expression from left to right and from top to bottom. Some systems [53] have space constraint: only connected subtrees in the Minimum Spanning Tree (MST) over strokes can form the symbol candidates. Some stochastic context-free grammar based systems [2, 9] only consider the horizontally and vertically adjacent strokes whey they apply CYK parsing. Some systems [9, 68, 72, 76, 86] have n-grams constraint. In those systems, a symbol at most has 4 or 5 strokes.

2.5.2 Graph probability maximization

A lot of methods [39,68,85,86] are based on graph probability maximization. In those methods, all possible symbol hypothesis will be generated and has a score produced by segmenter, classifier or structural analyzer. Those symbol hypothesis will form a graph. In the graph, each node represents a symbol candidate and each edge denotes the spatial relationship between the two symbols. Then the recognition of math recognition is converted to find the best symbol sequence among the graph which could maximize the probability.

Winkler et. al [39,85,86] propose a recognition system for online handwritten mathematical expressions based on symbol hypotheses net (SHN). For a given expression, an SHN is constructed. Figure 2.9 shows an example of SHN. The system has a strong but common assumption: a symbol at most has 4 strokes and the strokes have to be successive in time sequence. So each path in the SHN represents a symbol hypothesis sequence. The score for each symbol sequence is the product of the segmentation score and classification score. The segmentation probability for each symbol hypothesis is computed based on the stroke specific features and geometrical features between the strokes. Stroke specific features are used to classify each stroke to three complexity categories: primitive, standard and complex. Rules specify that only certain combinations of these categories are possible. Prerecognition is used to only detect dot, minus and fraction lines, and these classification results are used to improve the segmentation rate using rules for these specific symbols before segmenting the other symbols. The segmentation for these specific symbols is actually done by the prerecognition. After being found by prerecognition, the strokes for the specific symbols are separated from the other strokes and segmentation will only deal with the other strokes but not these specific strokes. The score of the symbol hypothesis sequence is the product of all the segmentation probabilities of the nodes along the sequence. The symbol classifier is based on HMM and the probabilities produced by HMM classifier are used to get the classification score for the symbol sequence in the same way as the segmentation score.

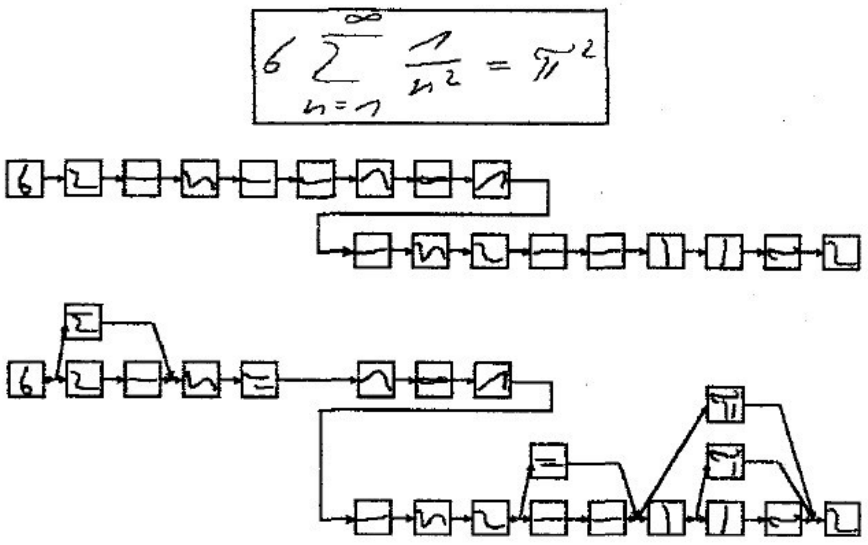


Figure 2.9: Handwritten expression, corresponding stroke sequence after preprocessing and the generated symbol hypotheses net [85].

2.5.3 Bayesian framework probability maximization

Probabilistic graphical model acquires success in image parsing and text recognition. Tu et. al [77, 78] present a Bayesian framework for parsing images which optimizes the posterior probability and produce a parsing graph. This method could segment the images, detect faces, and detect and read text at the same time. Wang et. al [83] present a Bayesian framework for the offline recognition of unconstrained handwritten Chinese texts. The method is based on character over-segmentation and candidate path search. It will construct a segmentation-recognition lattice for the text line image. The method evaluates the paths based on Bayesian inference by combining multiple contexts, including the character classification scores, geometric and linguistic contexts.

Shi et. al [68] present a unified Bayesian probabilistic framework for symbol segmentation and recognition of handwritten mathematical expressions. A symbol graph is generated for each expression. Figure 2.10 shows an example of symbol graph. The system assumes the user would write the symbol consecutively in time and a symbol at most has 4 strokes. So each path in the graph represent a possible symbol sequence. They define a function to calculate the grouping likelihood between two strokes by using three geometric features (horizontal distance, size difference and vertical offset). The parameters are learned by using training data. The probabilities used in probabilistic model are trained by simple Expectation Maximization algorithm or are the statistics data of the training data. Dynamic programming is used to find the best path in the graph.

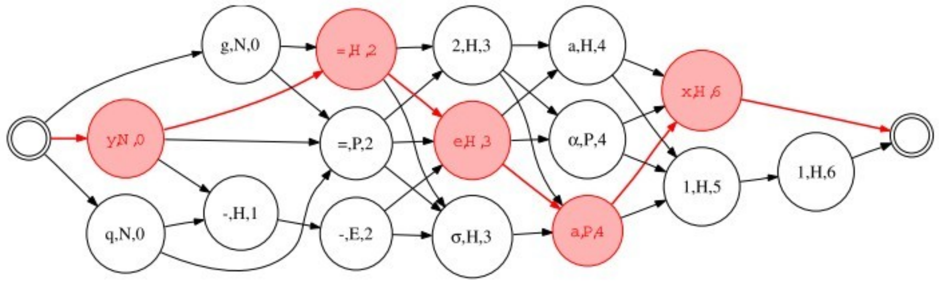


Figure 2.10: Example of symbol graph for expression $y = e^{ax}$ [68].

2.5.4 Graph penalty minimization

Graph penalty minimization method shares the same idea with the graph probability maximization method. The difference is that graph penalty minimization aims to minimize the penalty instead of maximizing the probability.

Suzuki et. al [25] present a math expression recognition system by using virtual link network. In the network, each node represents a symbol, and each edge is represented by four components: (parent candidate, child candidate, label, cost). The recognition is converted to find the spanning tree among the network with the minimum penalty. The penalty of the spanning tree contains two parts: local penalty and global penalty. The local penalty is calculated based on the distribution maps of relative sizes and positions for each relation types of parent-child links. The global penalty is based on the predefined rules. The total penalty is the sum of the link penalty of the edge and the global penalty. Although the parsing Eto and Suzuki proposed is based on MST, our method is quite different from theirs. We recognize handwritten math expressions, while they recognize printing ones. About the search algorithm, we use Edmonds' algorithm to find the MST in the directed LOS graph. They sort all the symbols from right to lest. Starting from the rightmost symbol, they use a local beam search to find an edge for the current symbol from the edges which end at the current symbol. We use LOS graph to construct the graph and the edge score is produced by the symbol pair classifier. How they construct the graph is missing, and they assign the score to the edge manually.

2.5.5 Methods based on grammars

The earliest math recognition system is base on grammar and it is proposed by Anderson [6] in 1967. In his system, each grammar production is equipped with predicates that determine how a set of input elements should be partitioned for parsing. The predicates include rules applying to the minimum, maximum, and central x- and y-coordinates of expression and symbol bounding boxes, as well as threshold-based rules to ensure proper alignment of neighboring subexpressions. The operator tree is constructed from top to bottom and a string which represents the tree structure is constructed from bottom to top.

MacLean et. al [48] present a new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. The approach reports all identifiable parses of the input expression. The parses are represented as a fuzzy set, in which the membership grade of a parse measures the similarity between it and the handwritten input. Parsing contains 2 steps: shared parse forest construction and parse tree extraction. To identify and report parses efficiently, they adapt and apply rectangular partitions and shared parse forests, and introduce relational classes and interchangeability. The parsing is incremental and can be real-time. The system has a correction mechanism that allows users to navigate parse results and choose the correct interpretation in case of recognition errors or ambiguity. Such corrections are incorporated into subsequent incremental recognition results. Three main contributions of this approach are fuzzy relational context-free grammars, relational classes and correction count metric. This system [48] gets the second place in

CROHME 2012 competition [57] after the company Vision Objects. In [48] , the notion of ambiguity is formalized in terms of fuzzy sets and portions of input are represented as fuzzy sets of math expressions.

In 2015, MacLean and Labahn [49] introduce a Bayesian scoring model for parsing trees. They construct a Bayesian network based on the input structure, and each joint variable assignment corresponds to a different parse tree. The distributions of stroke grouping, symbol identity and relation identity are based on the results of low-level classification systems; while distributions of expressions and symbol priors are derived from the grammar structure and training data respectively. This system [49] achieves better performance than their previous work [48]. It gets 55.75% expression recognition rate on CROHME 2011 dataset, and 55.18% on CROHME 2012 dataset.

2.5.6 Summary

These integration systems aim to optimize segmentation, symbol classification and layout analysis simultaneously. There are many interactions among the three tasks. This makes these systems have heavy computation task. All the integration systems generate a lot of symbol candidates for each expression and calculate a score produced by the segmentation classifier, symbol classifier or structural analyzer for each symbol candidate.

For graph-based methods, those symbol hypotheses will form a graph, in which each node represents a symbol candidate and each edge denotes the relationship between two symbol candidates. For different graph-based methods, each path or the sub-graph which obeys some rules form a possible interpretation for the given expression. All the possible interpretation are based on their scores. The interpretation with the optimal score is the recognition result. The score for an interpretation is usually the summation or production of all the scores of the symbol candidates within the interpretation. In order to reduce computation, all the graph-based methods have their constraints. Some systems have time constraint (e.g. only successive strokes in time series can belong to the same symbol) while some systems have space constraint (e.g. only connected subtrees in the Minimum Spanning Tree (MST) over strokes can form the symbol candidates or only strokes within certain window size can belong to the same symbol). Most systems constraint a symbol candidate at most has 4 or 5 strokes.

For grammar-based methods (including these stochastic context-free grammars based methods mentioned in Section 2.3), grammars are used to check the validity of proposed interpretations. Both stochastic context-free grammars based methods and fuzzy grammar based method assign a score to each production rule. These means the soft decision is useful. They explore all the possible interpretations and use the score of the production rule to measure the quality of the interpretation. Therefore, the quality of grammar is the key to these grammar-based methods. The time complexity is usually exponential if there are no constraints. The grammar-based methods also use some constraints mentioned for graph-based methods to reduce the computation.

For all current grammar-based systems, people have to manually design the grammars for different datasets. For the graph-based methods, once the graph type is defined, it can be applied to different datasets and do not need much change. Therefore the graph-based methods have better generalization ability than grammar-based methods. But through designing these grammars carefully, grammar-based methods can use people's prior knowledge and heuristics more and better than graph-based methods.

We think both graph-based methods and grammar-based methods have major shortcomings.

For the graph based methods, the time complexity will be exponential in the number of strokes if there is no constraints. That's why some strict constraints are introduced to reduce the computation. Another problem with the graph based methods is it is difficult to design a good score scheme to make the correct recognition result have the optimal score. In some situations, the top N recognition results produced by the graph based methods contain the correct recognition result, but the correct result does not get the highest score among the top N recognition results. The reason is that each of symbol segmentation, symbol classification, relation classification produces a score. Those scores are produced by a different classifier and usually are at different scale. It is hard to combine these different scores as one final score to evaluate the quality of recognition result.

For grammar-based methods, grammars and related parameters are defined manually for different tasks. Therefore, the generalization ability of grammar-based methods is poor. In addition, grammar-based methods are slow because the time complexity is also exponential in the number of strokes if there are no constraints.

2.6 Summary

We introduced the math expression representation, symbol segmentation, symbol classification and layout analysis in this section. We find many different feature sets are designed for different tasks. A feature set which can be calculated based on a simple rule and works well for symbol segmentation, symbol classification and symbol pair relation classification is highly desired.

For the graph based parsing, the time complexity will be exponential in the number of strokes if there are no constraints. That's why some strict constraints are introduced to reduce the computation. But a good graph representation with high recall and reasonable precision could help reduce the search space without adding constraints.

For the grammar-driven parsing, people have to manually design the grammars for different datasets. Is there any parsing could get competitive performance, but require much less human's intervention?

We are going to answer these questions in the following chapters. Chapter 3 is going to cover graph representation; Chapter 4 discusses visual features; while Chapter 5 and 6 are about how we get a simple but effective parsing.

Chapter 3

Graph Representation

The output of our math expression recognition is a symbol layout tree, in which the node represents symbol while the edge represents symbol pair relation. We can understand recognizing the structure of an expression as selecting a tree from a graph, and then classifying edges in the tree. It involves selecting edges, and labeling edges (relationships) and symbol nodes (stroke groups). Therefore, a good graph representation is essential to the math expression recognition. Ideally, a good graph representation should cover all the edges in the ground truth symbol layout tree (100% recall), and contain few invalid edges (high precision). Invalid edges are the edges which are in the graph but not in the ground truth symbol layout tree.

A high recall is required. Because if a ground truth edge is missing in the graph, the later parsing algorithm cannot select the missing edge and this would make the expression cannot be recognized correct even the later segmentation, symbol classification and parsing are perfect.

At the same time, a high precision is desired. Because even a graph representation has a high recall but a low precision, there would be a lot of invalid edges. This makes later parsing very difficult. For example, the complete graph can include all ground truth edges, but the precision is very low. For an expression with N strokes, there would be $N \times (N-1)$ edges, while there are only around N ground truth edge. The exact number of ground truth edges depends on the number of single-stroke symbols. If all symbols are single-stroke symbol in the expression, there are $N-1$ ground truth edge. For complete graph, the precision is around $\frac{1}{N}$. The average number of symbol in expression is larger than 10 in the datasets we use. In this case, the precision for complete graph is lower than 10%.

3.1 Graph Types

This section describes different graph representations we consider. Many previous work [33, 39, 40, 68, 72, 85, 86] use time series graph; while [25] use K nearest neighbor (KNN) as graph representation in the parsing. Minimum/Maximum Spanning Tree (MST) is used in [53].

Similar to KNN, Delaunay Triangulation creates a connected graph by adding an edge from each node to its nearby vertices (each node represents a stroke). But in Delaunay Triangulation, the number of neighbors for each node is different. Also, Delaunay Triangulation aims to divide the space uniformly and avoid skinny triangles. These characteristics make Delaunay Triangulation a good candidate for graph representation. [31] uses Delaunay Triangulation as graph representation.

For a given stroke, we find it usually can see the strokes which have a relation with it in the symbol layout tree, and almost all unseen strokes are not related to it. Stroke a can see Stroke b means the straight line connecting center of stroke a and center of stroke b would not have intersection with other different strokes. Therefore, we think line of sight (LOS) graph is worth trying.

Time Series For a given expression, for each stroke (except the last stroke), there is an edge from it to its subsequent stroke in time series and from its subsequent stroke to itself. Although there are some delayed strokes, many math expressions only have successive symbols in their symbol layout trees and most of symbols only contain successive strokes. Time series graph representation could capture the correct structure of the symbol layout tree for the math expressions which only contain successive symbols and successive strokes. Time series graph has a chain structure and it is a special tree (each parent node only contains a single child node).

MST In the MST each node represents a stroke. The edge in the MST is undirected. To compare it to the ground truth label graph, for each undirected edge [i,j], we use two directed edges (i,j) and (j,i) to represent it. So for an expression with N strokes, there are N-1 undirected edges in its MST and 2*(N-1) directed edges in the corresponding graph representation.

KNN Graph In KNN graph, for a given stroke, there is a directed edge from it to each of its K nearest neighbor strokes and from each of its K nearest neighbor strokes to itself.

Delaunay Triangulation Delaunay Triangulation (DT) for a set of points S in a plane is a triangulation DT(S) such that no point in P is inside the circumcircle of any triangle in DT(S). Delaunay triangulations maximize the minimum angle of all the angles of the triangles in the triangulation; they tend to avoid skinny triangles [14]. Delaunay Triangulation (DT) is the dual structure of the Voronoi diagram. Voronoi diagram is the partition of the plane into blocks of points with the same nearest site or sites [14]. 1-NN defines a Voronoi diagram, where all points closest to a point are in the Voronoi polygon containing the given point.

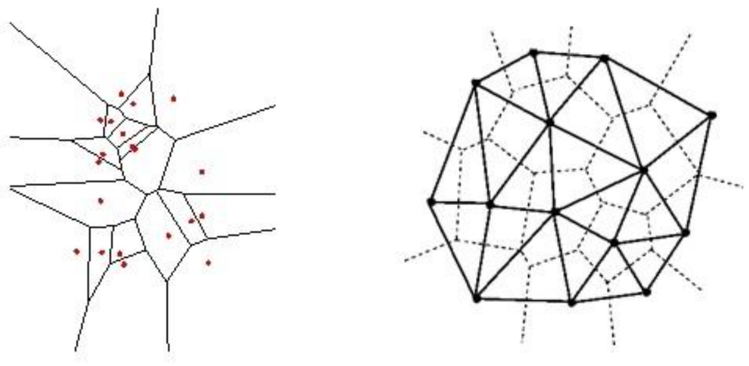


Figure 3.1: Delaunay Triangulation [14]

In the Delaunay Triangulation, each center of the stroke bounding box is a node. We get all edges of all the triangles in the Delaunay Triangulation. The edge in the Delaunay Triangulation is undirected. The same as MST, for each undirected edge [i,j], we use two directed edges (i,j) and (j,i) to represent it.

Line of Sight The center of each stroke is taken as an eye. There would be an edge from the current stroke to any stroke it could see. The distance between any two strokes could be CPP distance, BBC distance. Each stroke could be represented as a bounding box (BB) or a convex hull (CH). The convex hull CH(S) of a set of points S is the smallest convex set that contains S [14]. Line of Sight 360 Degree (LOS) is a visibility graph [14]. Algorithm 1 and 2 are used to calculated the line of sight graph as showed in Figure 3.2 and 3.3. For each stroke, its center is taken as the eye (the point P as showed in Figure 3.2). For the given eye stroke, all the other strokes are sorted based on the distance between it and the eye stroke. For each other stroke by increasing distance, we calculate the angle (BAR) its convex hull can block from the eye. If there is non-zero overlap between the BAR and unblocked angle range, we add an edge (visibility edge) between the eye stroke and the stroke. Then we would remove the overlap from the unblocked angle range and deal with the next stroke.

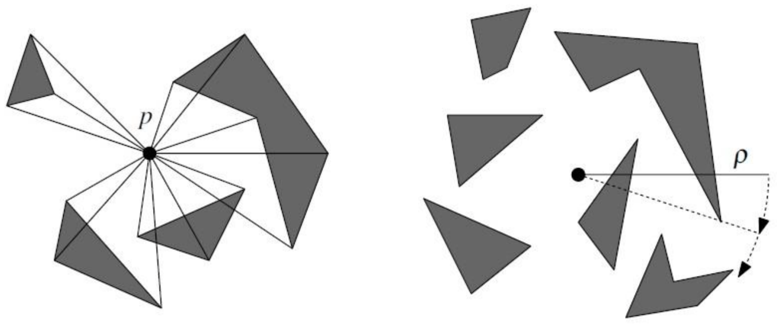


Figure 3.2: Line of sight graph [14]. The point ’P’ is the eye.

In Algorithm 2, the input is an eye and a convex hull and output is the angle range the convex hull can block from the eye.

Figure 3.3 shows a stroke level line of sight graph for a math expression.

Algorithm 1 Line of sight

```
for each stroke $S$, its bounding box center is eye and the unblocked angle range UAR is [0, 2pi]: do
    Sort the other strokes based on the closest point pair distance between two strokes.
    for each other stroke $OS$ by increasing distance from $S$: do
        (a) Calculate all the angle range BAR its convex hull can block.
        (b) Check the visibility visibility angle range (VAR) = overlap between BAR and UAR
            if the stroke can be seen from the eye (VAR != 0)
            then
                (1) add an edge from $S$ to $OS$
                (2) UAR = UAR - VAR
    end for
end for
```

Algorithm 2 Calculate block angle range (BAR)

```
for each nodes ($X_n$, $Y_n$) in the convex hull do
    Construct the vector (V1) from the eye ($X_0$, $Y_0$) to the node. V1 = ($X_n$-$X_0$, $Y_n$-$Y_0$)
    Calculate the angle $\theta$ between V1 and horizontal vector V2 = (1, 0).
```

$$\theta = \begin{cases} \arccos(\frac{V1 \cdot V2}{||V1||||V2||}) & \text{if } Y_n \geq Y_0 \\ 2 \times \pi - \arccos(\frac{V1 \cdot V2}{||V1||||V2||}) & \text{if } Y_n < Y_0 \end{cases}$$

```
end for
Return BAR = [min $\theta$, max $\theta$]
```

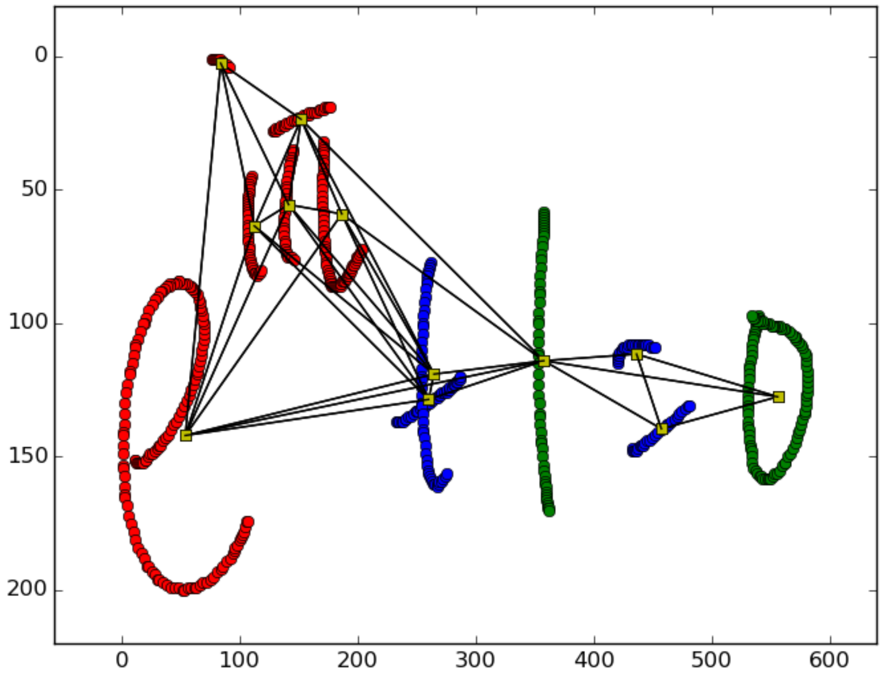


Figure 3.3: Line of sight graph for a math expression.

3.1.1 Distance Metrics

Both MST and KNN need to calculate the distance between two strokes. We test three different Euclidean distances.

AC distance Averaged Center (AC) distance represents averaged center distance. AC is the mean of x and y coordinates for the set of stroke points. It is also known as the center of mass.

BBC distance BBC represents bounding box center. BBC distance is the Euclidean distance between the two bounding box centers of the two strokes.

CPP distance CPP distance represents closest point pair distance. It is the minimal distance between the point pair in which each of the two strokes has a point.

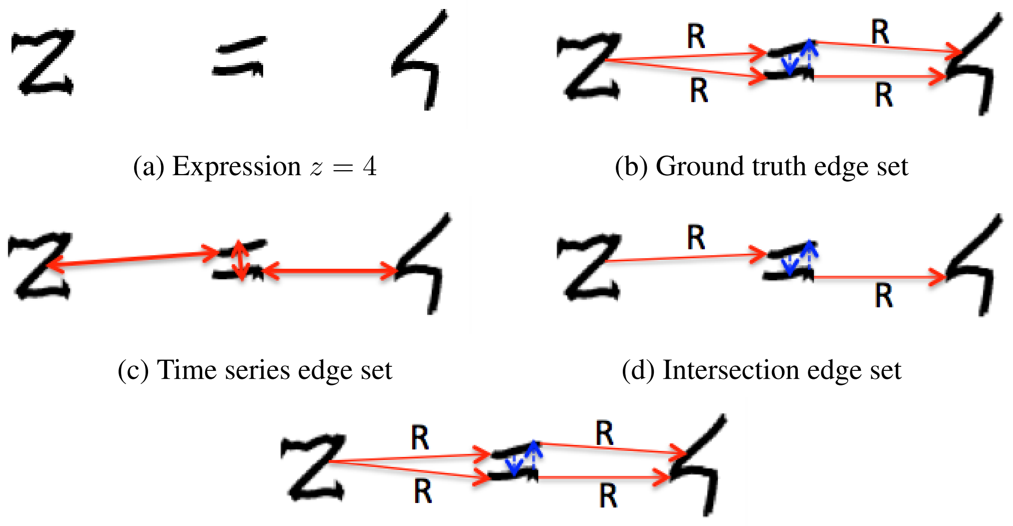
For Delaunay Triangulation, we do not need the distance between stroke pair, but need a point to represent the stroke. We use bounding box center (BBC) or center of mass to represent the stroke.

3.2 Recovering Symbol Layout Trees From Graphs

Algorithm 3 Recover symbol layout tree from graph representation. For edge label between stroke pair, ***** means merge, and **-** represents no relation.

```
for each expression $E$: do
    1. (Get graph representation) Use graph representation to get the edge set GR.
    2. (Get ground truth) Use ground truth to get the ground truth symbol layout tree edge set GT.
    3. (Intersection) Get the intersection edge set I of GR and GT.
    4. (Recover symbol layout tree) Recover symbol layout tree based on I to get the
    recovered edge set R.
        (a) (Create a clique for each symbol) Make sure there is an edge from stroke j to stroke i
        with label * if there is an edge from stroke i to stroke j with label *.
        (b) (Inherit Relationships) Incoming and outgoing edges between symbols are shared by
        all of their member strokes.
    5. (Return) R as final edge set.
end for
```

Algorithm 3 recovers the possible symbol layout tree from the graph representation edge set. Figure



(a) Expression $z = 4$

(b) Ground truth edge set

(c) Time series edge set

(d) Intersection edge set

(e) Recovered edge set

Figure 3.4: An example of getting symbol layout tree from graph representation. For expression $z = 4$, stroke 1 is ‘z‘; stroke 2 is the upper ’-’ of ’=’; stroke 3 is lower ’-’ of ’=’, stroke 4 is ’4’. **R** represents ’Right’ relation, while blue dashed line represents edge with ’merge’ relation.

3.4 shows on example about getting symbol layout tree from time series graph representation for expression $z = 4$. As showed in Figure 3.4 (b), strokes in the same symbol have the same edges. For example stroke 2 and stroke 3 belong to the same symbol ’=’, they share the same incoming edge from stroke 1 with label ’Right’, and share the same outgoing edge to stroke 4 with label ’Right’.

Therefore, **Algorithm 3** would add some edges if necessary to ensure strokes within the same symbol have the same relation with the strokes in different symbols. For example, if stroke i and stroke j belong to the same symbol while stroke k belongs to a different symbol, and there is an edge from i to k in the filtered edges with relation ’X’ (’X’ could be Right, Sub, Sup, Above, Below, Inside) but there is no edge from j to k, then an edge from j to k with relation ’X’ would be added.

Figure 3.4 (d) shows there are only 4 edges in the intersection edge set between the ground truth edge set and time series edge set; while there are 6 edges in the ground truth edge set as showed in Figure 3.4 (b). Step 4 (b) in Algorithm 3 would add edge ’1→3, Right’ to make the two strokes in symbol ’=’ have the same incoming edge; and add edge '2→4, Right' to make the two strokes in symbol '=' have the same outgoing edge.

The recovered edge set is subset of the ground truth edge set, and there are two cases for the recovered edge set. If the recovered edge set is the same as ground truth edge set, the ground truth symbol layout tree can be represented correctly by the graph representation. If the recovered edge set is only a proper subset of ground truth edge set, this means at least one edge in the ground truth edge set cannot be captured by the graph representation and then the graph representation cannot represent the ground truth symbol layout tree correct.

Figure 3.4 (e) shows that the recovered edge set is the same as ground truth edge set. Therefore, time series graph representation can capture the symbol layout tree of expression $z = 4$ correct. In another word, **Algorithm 3** is actually used to test whether there are sufficient edges to recover the ground truth symbol layout tree from a given graph.

3.2.1 Metrics

There are three metrics (correct expression number, recall, precision and F-score) we use in the graph representation experiment.

Correct expression number is at expression level. Given one graph representation, for each expression, we get three directed edge sets: ground truth edge set, graph representation edge set, and the recovered edge set. For a given expression, if the ground truth edge set and the recovered edge set are the same, the expression is correct.

The other three metrics (recall, precision and F-score) are at stroke level. They are for stroke pair edges.

$$\text{Recall} = \frac{\text{number of edges in recovered edge set}}{\text{number of edges in the ground truth edge set}} \tag{3.1}$$

$$\text{Precision} = \frac{\text{number of edges in recovered edge set}}{\text{number of edges in the graph representation edge set}} \tag{3.2}$$

$$\text{F measure} = 2 \times \frac{\text{precision} \times \text{recall}}{\text{precision} + \text{recall}} \tag{3.3}$$

3.3 Model Selection

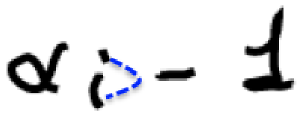
This section describes how we choose the model for different graph representation and shows expressions which cannot be represented correctly for each graph representation. We need to choose distance metric between stroke pair for MST. For KNN, the number of K and distance metric need to be selected. Regarding Delaunay Triangulation, we need to determine which stroke center should be used to represent the stroke. For LOS, we need to choose the eye and shape representation for each stroke.

The results reported in this section are based on 90% expressions (1221 expressions) of the CROHME 2012 training set.

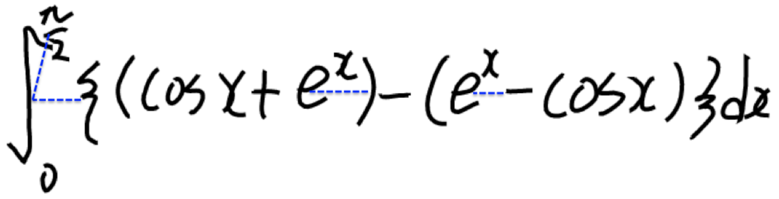
3.3.1 Time Series

Time series graph representation usually can achieve a very high precision (higher than 90%), and many previous work [33, 40, 72] use time series as graph representation.

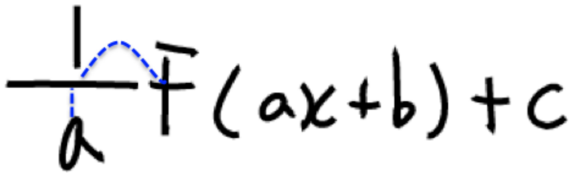
Because time series can only contain time consecutively stroke pair, it cannot capture the edge related to a delayed stroke and would miss the edge for the strokes with multiple children nodes. Figure 3.5 (a) shows one expression with delayed stroke, and time series graph representation cannot capture the edge related to the delayed stroke. Figure 3.5 (b) (c) show time series graph representation usually miss the edges related to a stroke with multiple children nodes in the ground truth symbol layout tree.



(a) The missing edge is the edge between the two strokes of 'i'. As the writer wrote '$-$', '1' and then the dot of 'i' after wring the bottom part of 'i'.



(b) The missing edges are the edge between '$\int$' and fraction bar '-', the edge between '$\int$' and '$\{$', the edge between the first 'e' and '$)$', and the edge between the second 'e' and the second minus '-'.



(c) The missing edges are the edge between the fraction bar '-' and the first 'a', and the edges between the fraction bar '-' and the 3 strokes of 'F'.

Figure 3.5: Missing edge for TS. The blue dashed line (curve) represents missing ground truth edge.

3.3.2 MST

Figure 3.6 shows the comparison for the three distance metrics (average center distance, bounding box center distance, and closest point pair distance) when the graph representation is MST. AC distance performs better than the other two distance metrics. Therefore, we choose Euclidean distance between the centers of mass as the distance metric for MST.

The precisions for MST with different distance metrics are all higher than 90%. This means most of the edges in the MST are in the ground truth symbol layout tree. Another reason for the high precision is the number of edges in the MST is smaller than the other graph representations (KNN, Delaunay Triangulation and LOS).

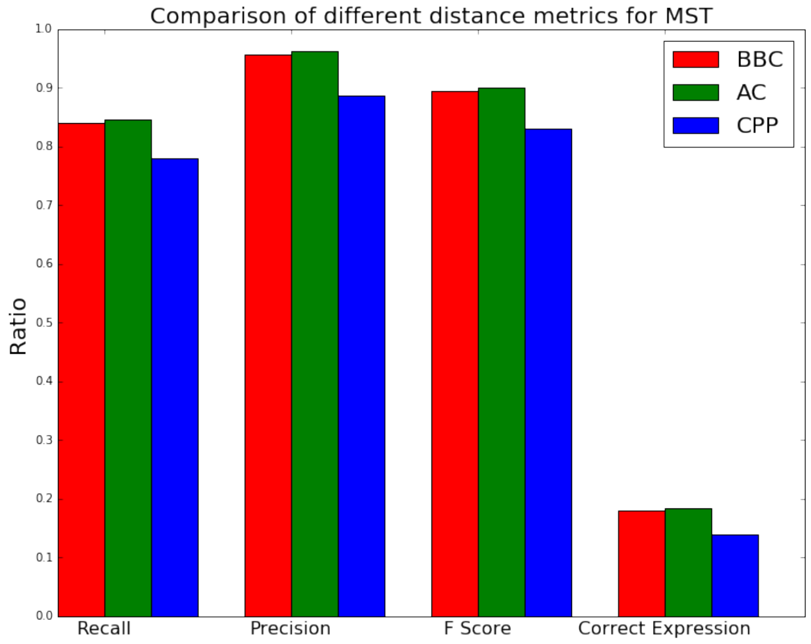
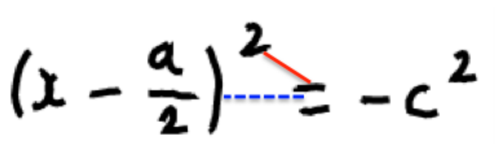
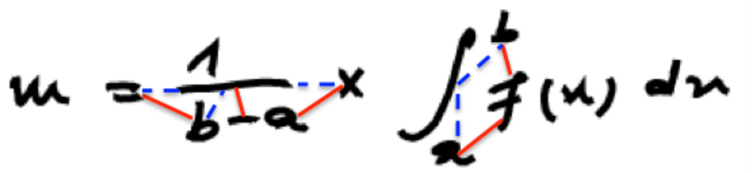


Figure 3.6: Comparison of different distance metrics (closest point pair, bounding box center and average center) for MST. Recall, precision and F-score are for stroke pair edges.

The MST cannot deal with the superscript and subscript very well. For example, for $a^b c$ where symbol 'b' is the superscript of symbol 'a', and symbol 'c' is right adjacent of symbol 'a', there would be an edge between symbol 'a' and symbol 'b', and an edge between symbol 'a' and symbol 'c' in the ground truth symbol layout tree. But the MST usually can capture the edge symbol 'a' and symbol 'b', but not the edge between symbol 'a' and symbol 'c'. Because symbol 'b' is usually closer than symbol 'a' to symbol 'c', and there usually be an edge between symbol 'b' and symbol 'c' in the MST. Figure 3.7 (a) shows one superscript failure example as mentioned above, and Figure 3.7 (b) shows MST would usually fail to capture some edges related to a stroke with multiple children nodes in the ground truth symbol layout tree.



(a) The missing edge is between ')' and '='. In the MST, there is an edge between ')' and the first '2', and an edge between the first '2' and '='.



(b) The missing edges are the edge between '=' and fraction line '-', the edge between fraction line '-' and the first 'b', the edge between fraction line '-' and 'x', the edge between '$\int$' and the second 'a', the edge between '$\int$' and the second 'b'

Figure 3.7: Missing edge for MST AC. The blue dashed line represents missing ground truth edge, while the red solid line is the corresponding wrong edge in the MST.

3.3.3 KNN

Figure 3.8, 3.9 shows the comparison of recall and correct expression number among the three distance metrics (average center distance, bounding box center distance, and closest point pair distance). CPP distance performs better than the other two distance metrics; while the performance for BBC distance and AC distance are very close. Therefore, we choose the Euclidean distance between closest point pair as the distance metric for KNN.

Figure 3.10 shows that the larger K is, the higher is the recall and the lower is the precision. When K is 2, the F-score is the highest. The recall is higher than 99% when K is 6, but it does not hit 100% until K is 20. This means in order to represent all expressions correctly, we need to add an edge for each stroke to its 20 nearest strokes. It is easy to know that there are a lot of invalid edges in the 20NN, and Figure 3.10 (b) shows the precision is around 10% for 20NN.

For KNN graph representation, we choose 2NN and 6NN. Because 2NN achieve the highest F-score, while 6NN achieve the highest precision for KNN with a recall higher than 99%.

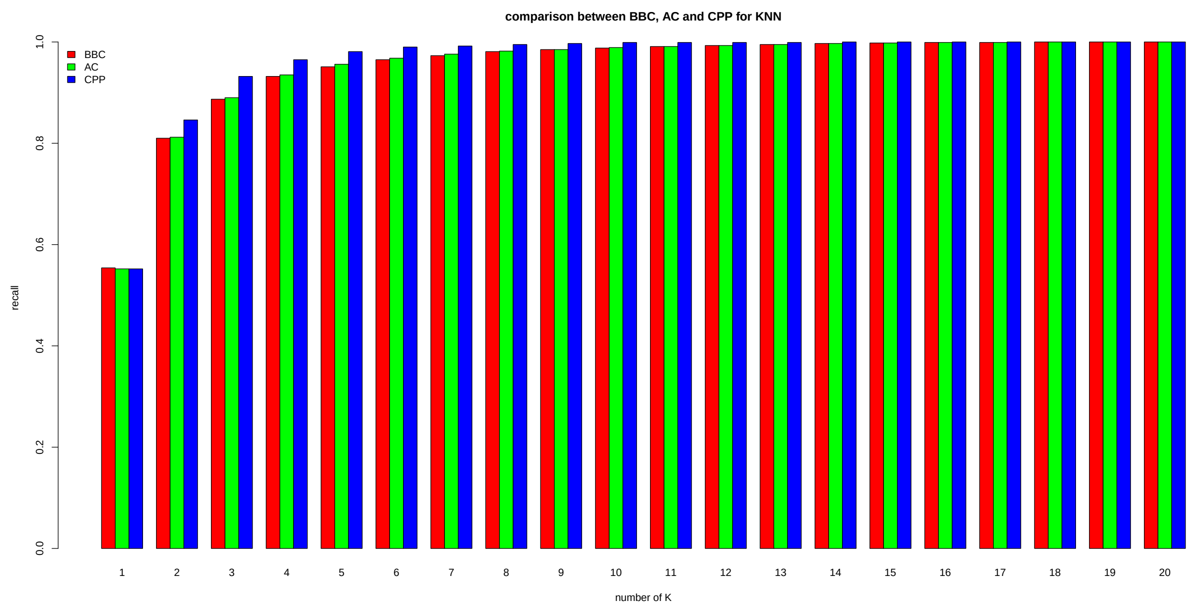


Figure 3.8: KNN recall comparison between closest point pair, bounding box center and average center. Recall is for stroke pair edges

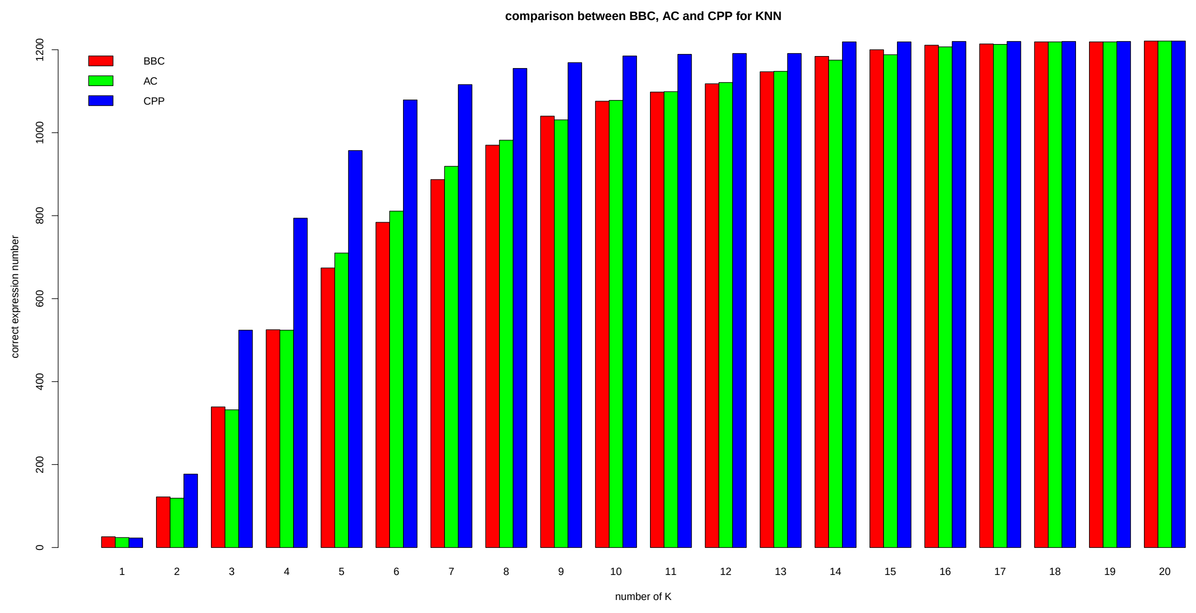
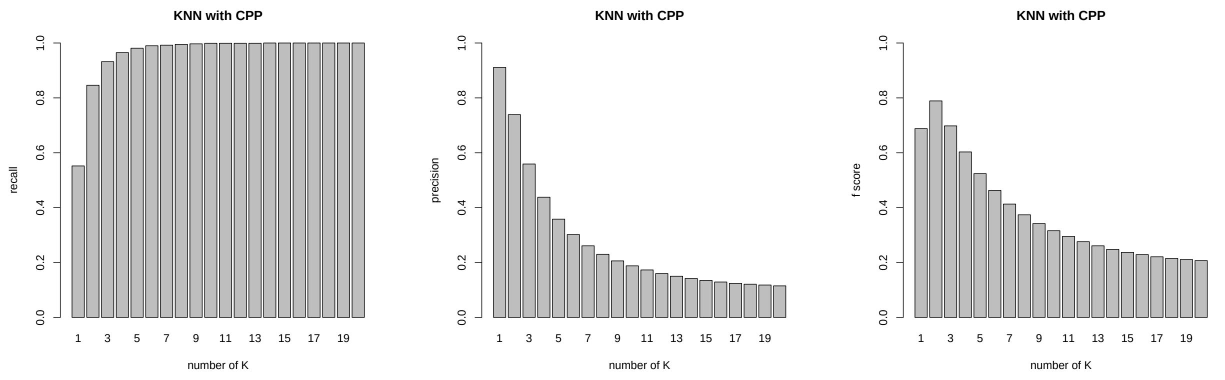


Figure 3.9: KNN correct expression number comparison between closest point pair, bounding box center and average center



(a) Recall (b) Precision (c) F-score

Figure 3.10: Recall, precision, F-score comparison for different number of K in KNN. The distance metric is closest point pair distance. Recall, precision and F-score are for stroke pair edges.

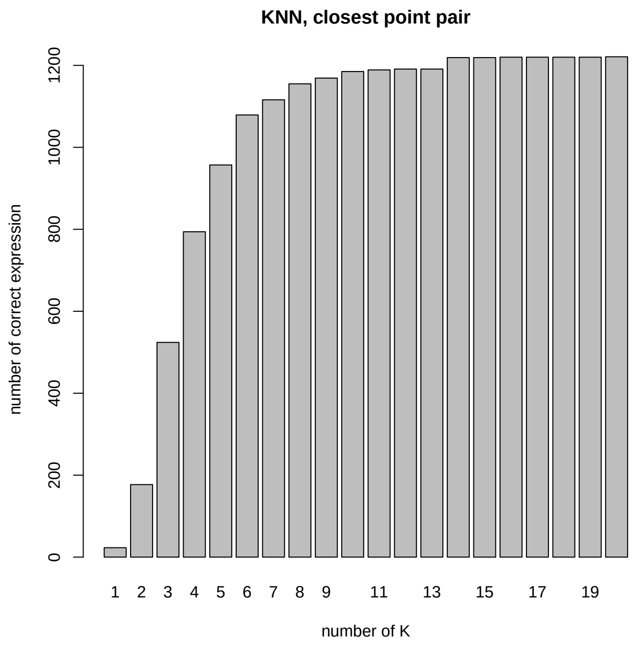
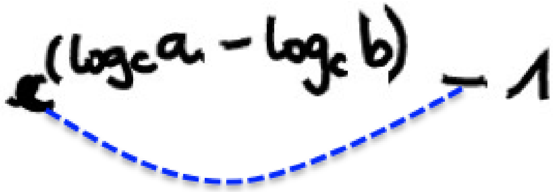
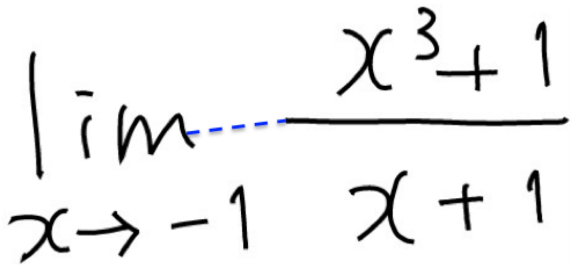


Figure 3.11: KNN closest point pair correct expression number.

Although 6NN can achieve a recall higher than 99%, it cannot handle big superscript and long horizontal bar correctly sometimes. Figure 3.12 (a) shows an example related to big superscript, while Figure 3.12 (b) shows an example related to long horizontal bar.



(a) The missing edge is between the leftmost 'c' and the second '$-$' because of the big superscript $(log_{c^a} - log_{c^b})$



(b) The missing edges are between each of the 4 strokes of '$\lim$' and the fraction line '-' between $x^3 + 1$ and $x + 1$

Figure 3.12: Missing edge for 6NN. The blue dashed line (curve) represents missing ground truth edge.

3.3.4 Delaunay Triangulation

Figure 3.13 shows the comparison for the two stroke centers (averaged center and bounding box center) when the graph representation is Delaunay Triangulation. We can find that averaged center performs slightly better than bounding box center. Therefore, we choose averaged center (center of mass) to represent the stroke for Delaunay Triangulation.

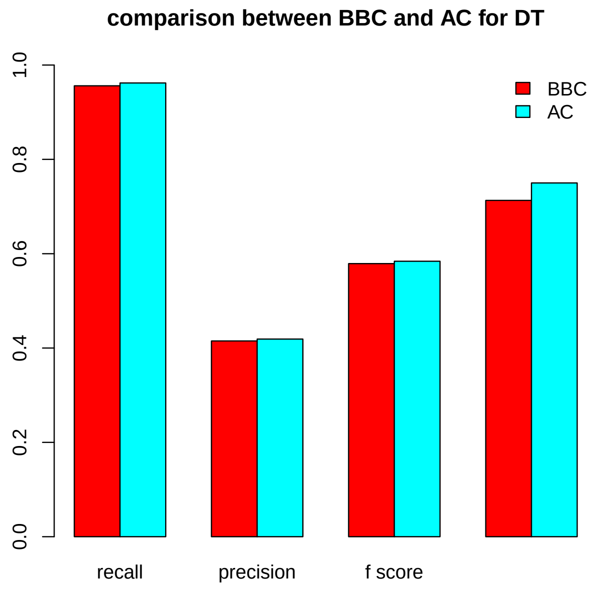


Figure 3.13: Delaunay Triangulation comparison between averaged center and bounding box center. Recall, precision and F-score are for stroke pair edges.

3.3.5 LOS

Table 3.1 shows that convex hull can capture more edges than bounding box when the distance metric is bounding box center distance or closest point pair distance. This supports that convex hull can represent stroke more precisely than bounding box. Table 3.1 also shows that closest point pair distance can capture more edges than bounding box center distance when the polygon type is bounding box or convex hull. Therefore, we choose convex hull to represent the polygon shape of stroke, and Euclidean distance between closest point pair as the distance metric for LOS.

Table 3.1: Missing edge for LOS related graph representation. Polygon type could be bounding box (BB) or convex hull (CH); while distance metric could be distance between bounding box centers (BBC) or closest point pair distance (CPP)

Graph Representation (Polygon Type, Distance Metric)	*****	**R**	**Sub**	**Sup**	**A**	**B**	**I**
LOS (BB, BBC)	0	51	0	3	0	0	40
LOS (BB, CPP)	0	17	0	8	0	0	12
LOS 3NN (BB, BBC)	0	15	0	0	0	0	2
LOS (CH, BBC)	0	27	0	0	0	0	16
LOS (CH, CPP)	0	14	0	0	0	0	8

Figure 3.14 shows LOS CH can capture more than 99.9% of edges in the ground truth symbol layout tree, and the precision (0.309) is acceptable. This means LOS CH contains almost all the ground truth edges, but there are less redundant edges than KNN when K is large. This makes LOS CH a good graph representation.

To improve the precision and F-score of LOS, we tried to restrict the angle range of the sight. This means for each stroke, we only add an edge from it to another stroke if the other stroke can be seen from the stroke within the restricted angle range (such as $[0.5\pi, 1.5\pi]$) instead of 360 degrees ($[0, 2\pi]$). Experiment results show restricting the angle range hurts the recall and F-score. Therefore, we do not restrict the angle range.

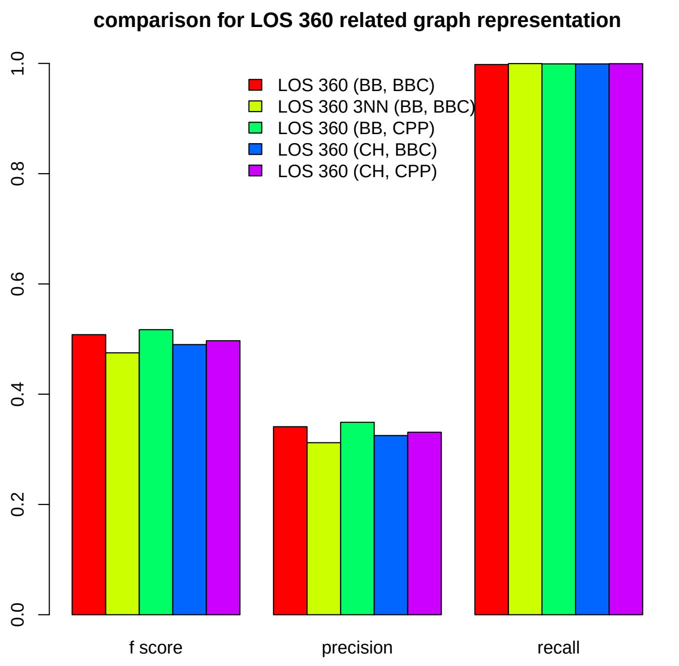
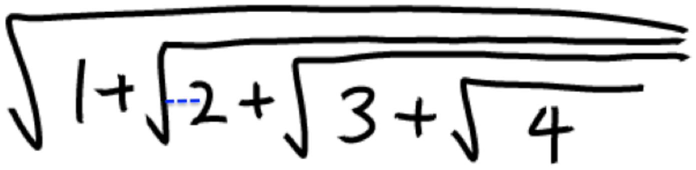


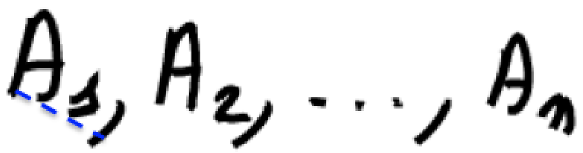
Figure 3.14: Comparison for LOS related graph representation. Recall, precision and F-score are for stroke pair edges.

We choose the center of the stroke as the eye of the stroke. It has some limits.

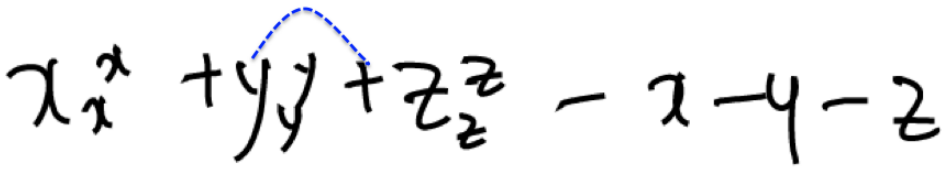
Figure 3.15 (a) (b) shows two examples in which stroke ’a’ can see stroke ’b’ but the center of stroke ’a’ cannot see stroke ’b’. The error in 3.15 (a) can be corrected by making LOS CH strictly symmetric. This means for each edge (i, j) in LOS CH, we would check whether edge (j, i) is in it or not. If not, we would add edge (j, i) into it. In this way, we can get LOS CH 2 (symmetric). For most of the edges in LOS CH, its symmetric edge is also in LOS CH (because when stroke i can see stroke j, stroke j usually can see stroke i). LOS CH 2 does not contain much more edges than LOS CH. The performance difference between LOS CH and LOS CH 2 (symmetric) is very small. But LOS CH 2 (symmetric) can achieve better recall, F-score and capture more correct expressions than LOS CH.



(a) Missing edge for LOS CH. The missing edge is between the second '$\surd$' and '2'. It is easy to find that the second '$\surd$' should see '2'. The reason the edge is missing is we use the center of the stroke as the eye. For the second '$\surd$', its eye is around the third '$+$', so its sight to the '2' is blocked by '$+$', '3' and the third '$\surd$'.



(b) Missing edge for LOS CH (symmetric). The missing edge is between the first 'A' and the first ','. The eyes (centers) of 'A' cannot see the first ',', and the eye (center) of the first ',' cannot see 'A'. But the bottom of the first 'A' can see the first ',', and the bottom of the first ',' can see the first 'A'.



(c) Missing edge for LOS CH (symmetric + multiple eyes). The missing edge is between the first 'y' and the second '$+$'. The sight from them to each other are totally blocked by the two 'y' between them.

Figure 3.15: Missing edge for LOS CH. The blue dashed line (curve) represents missing ground truth edge.

3.15 (b) shows an error cannot be corrected by making LOS CH strictly symmetric, but can be corrected by using multiple eyes. Therefore, we add topmost, bottommost, leftmost, rightmost points as eyes in LOS CH 3 (symmetric + multiple eyes). After using multiple eyes, many edges (most of them are invalid edges) are added. This makes LOS CH 3 (symmetric + multiple eyes) has a much lower precision and F-score than LOS CH 2 (symmetric). We also could use all the vertices of the convex hull as eyes. But the performance would be similar.

3.15 (c) shows an expression in which the sight between two strokes are totally blocked by other strokes. Line of sight graph representation cannot capture this kind of expression correct, and this is the limit of LOS.

3.4 Experiments

3.4.1 Datasets

Table 3.2: CROHME 2012 and 2014

dataset	**training expression**	**test expression**
CROHME 2012	1336	486
CROHME 2014	8834	986

Competition on Recognition of On-line Handwritten Mathematical Expressions (CROHME) is a well-established annual competition [56–60]. The results reported in this section are based on datasets CROHME2012 [57] and CROHME2014 [58]. For CROHME2012, we use all expressions on the list listeINKML_part3Test.txt and listeINKML_part3Train.txt. For CROHME 2014, we use all the expressions except two expressions in the training data. MfrDB3088.inkml is empty and MfrDB0104.inmkl is not well-formed. We discard these two and use the rest 8834 expressions as CROHME 2014 training data.

3.4.2 Results

Table 3.3, 3.4, 3.5, 3.6 shows the results on CROHME 2012 training set, CROHME 2012 test set, CROHME 2014 training set, CROHME 2014 test set.

It is easy to find that different graph representations have very stable performance for edges on the four different datasets. The precisions for TS and MST are both higher than 90%. Because most of expressions in the dataset are written in time sequence, and MST can capture the ground truth symbol layout tree well. Another reason for the high precision is the number of edges in the TS and MST is smaller than the other graph representations (KNN, Delaunay Triangulation and LOS). TS or MST is a good choose if high precision is required.

LOS CH 2 is the graph representation which has the highest F-score among the graph representations which achieve a recall higher than 99%. It performs better than 6NN in terms of all metrics (recall, precision, F-score correct expression number). Especially, its recall is higher than 99.9% on all the four different datasets. This means only less than 0.1% edges are missing. The precision is around 30% and there are not many invalid edges. We choose LOS CH 2, as we need a graph representation with a high recall and a reasonable precision.

Table 3.3: Graph representation comparison for CROHME 2012 training (1336 expressions). LOS CH 2 is LOS CH symmetric; LOS CH 3 is LOS CH symmetric + multiple eyes

Graph Representation	**Recall**	**Precision**	**F-score**	**Correct Expression**	**Correct Expression %**
2NN CPP	0.847	0.739	0.789	196	14.67%
MST AC	0.847	0.965	0.902	251	18.79%
Time Series	0.853	0.970	0.908	267	19.99%
Delaunay AC	0.963	0.419	0.584	827	61.90%
6NN CPP	0.990	0.302	0.463	1184	88.85%
LOS CH	0.9996	0.329	0.495	1321	98.88%
LOS CH 2	0.9997	0.330	0.497	1329	99.48%
LOS CH 3	0.9999	0.255	0.406	1333	99.78%
complete graph	1.000	0.105	0.190	1336	100%

Table 3.4: Graph representation comparison for CROHME 2012 testing (486 expressions). LOS CH 2 is LOS CH symmetric; LOS CH 3 is LOS CH symmetric + multiple eyes

Graph Representation	Recall	Precision	F-score	Correct Expression	Correct Expression %
2NN CPP	0.879	0.708	0.784	121	24.90%
Time Series	0.878	0.917	0.897	152	31.28%
MST AC	0.882	0.921	0.901	176	36.21%
Delaunay AC	0.977	0.388	0.555	373	76.75%
6NN CPP	0.994	0.286	0.444	437	89.92%
LOS CH	0.999	0.309	0.472	479	98.56%
LOS CH 2	0.999	0.309	0.472	479	98.56%
LOS CH 3	0.9998	0.233	0.378	484	99.59%
complete graph	1.000	0.087	0.159	486	100%

Table 3.5: Graph representation comparison for CROHME 2014 training (8834 expressions). LOS CH 2 is LOS CH symmetric; LOS CH 3 is LOS CH symmetric + multiple eyes

Graph Representation	Recall	Precision	F-score	Correct Expression	Correct Expression %
Time Series	0.864	0.932	0.896	3697	41.85%
2NN CPP	0.864	0.703	0.775	3701	41.89%
MST AC	0.869	0.937	0.902	3851	43.59%
Delaunay AC	0.967	0.407	0.573	6683	75.65%
6NN CPP	0.994	0.294	0.454	8330	94.29%
LOS CH	0.998	0.325	0.490	8663	98.06%
LOS CH 2	0.999	0.324	0.489	8683	98.29%
LOS CH 3	0.9992	0.255	0.406	8763	99.20%
complete graph	1.000	0.100	0.181	8834	100%

Table 3.6: Graph representation comparison for CROHME 2014 testing (986 expressions). LOS CH 2 is LOS CH symmetric; LOS CH 3 is LOS CH symmetric + multiple eyes

Graph Representation	**Recall**	**Precision**	**F-score**	**Correct Expression**	**Correct Expression %**
Time Series	0.868	0.891	0.879	402	40.77%
MST AC	0.875	0.899	0.887	418	42.39%
2NN CPP	0.885	0.685	0.773	443	44.93%
Delaunay AC	0.973	0.391	0.558	780	79.11%
6NN CPP	0.994	0.283	0.441	945	95.81%
LOS CH	0.997	0.296	0.457	966	97.97%
LOS CH 2	0.9992	0.297	0.458	976	98.99%
LOS CH 3	0.9997	0.233	0.378	983	99.70%
complete graph	1.000	0.091	0.167	986	100%

As both MST and TS can get high precision but a relatively low recall. We combine TS and MST to expect a higher recall than any one of the two graph representations, but a still high precision. Table 3.7 shows the result of combination of TS and MST. The recall of the combination is only 2% higher than the recall of either TS or MST. This means there is a big overlap of the edges in TS and the edges in MST.

Table 3.7: Missing Edge for TS and MST combination

recall	*	R	Sub	Sup	A	B	I
0.889	2	4516	8	287	343	224	19

Table 3.8 shows that the majority of missing edges in LOS CH are caused by the limit of using stroke center as the only eye, while the other missing edges are caused by the totally blocked sight. For the merge (*) relationship, the strokes are close to each other and then can see each other. Therefore, almost all edge with merge label can be covered by the line of sight graph. It lays a good foundation for the segmentation

based on the line of sight graph. The majority missing edge has 'Right' relation. The reason is the distance between stroke pair with 'Right' relation could be large, and this may cause the sight blocked by the other strokes.

Table 3.8: Missing edge for different datasets with different LOS CH graph representations. For each cell, a,b,c are for LOS CH, LOS CH 2 (symmetric), and LOS CH 3 (symmetric + multiple eyes)

Dataset	*****	**R**	**Sub**	**Sup**	**A**	**B**	**I**
CROHME 2012 testing	0,0,0	10,10,3	0,0,0	0,0,0	0,0,0	3,3,1	0,0,0
CROHME 2012 training	0,0,0	14,14,7	0,0,0	0,0,0	0,0,0	0,0,0	8,0,0
CROHME 2014 testing	54,0,0	24,22,8	0,0,0	0,0,0	0,0,0	0,0,0	0,0,0
CROHME 2014 training	18,2,0	237,235,105	65,65,28	24,24,13	17,16,15	16,16,12	9,0,0

3.5 Summary

As mention in Chapter 1, math expression recognition could be taken as selecting a tree from a graph in which the node represents stroke while edge represents stroke pair relation. A good graph representation is important to math expression recognition. There are two requirements for good graph representation: (1) it should almost cover all the edges in the ground truth symbol layout tree (high recall is necessary); (2) it should contain as few as noisy edges (high precision is desired).

Most of the prior work use MST [53], time series [33, 39, 40, 68, 72, 85, 86] or KNN [25] as the graph representations in the parsing. But neither of them meets the requirement of good graph representation. We find that MST and time series have a high precision but a low recall. This makes many expressions cannot be recognized correct even before the recognition. We also find that KNN with a large K contains too many invalid edges. This makes later parsing very difficult.

For stochastic context-free grammar based parsing [2, 9], they start constructing the graph from the most top-left stroke, and then search over the horizontally and vertically adjacent strokes. In order to reduce the search space, they use the writing order information and limit the number of strokes in a symbol. The graph they search [2, 9] is a combination of time series and KNN in which the distance is horizontal distance or vertical distance.

We propose the line of sight graph representation with convex hull representing the polygon shape of the stroke. Its performance is stable on the four different datesets (CROHME 2012 training/test, CHROHME 2014 training/test). The edge recall is higher than 99.9% edges while precision higher than 30%. This means more then 30% of the edges in the line of sight graph are in the ground truth edge set, while only less than 0.1% in the ground truth edge set cannot be captured. Also, line of sight graph achieves better recall, precision, F-score and capture more expressions than 6NN.

We think line of sight graph representation with convex hull is the best graph representation among the six different graph representations discussed in this chapter, and we use LOS CH 2 (symmetric) as the graph representation for this thesis.

In Chapter 4, we are going to talk about visual features and its use in symbol segmentation, symbol classification and parsing.

Chapter 4

Visual Features for Symbol Segmentation, Symbol Classification and Layout Analysis

Good features are important to any pattern recognition system. As mentioned in Chapter 2, many different features are manually designed for the three tasks (symbol segmentation, symbol classification, symbol pair relation classification) in math expression recognition. For most of those features, each one is calculated by a specific rule.

We are going to explore whether visual features are enough for math expression recognition? Are there any features works well for all the three tasks.

We propose shape context feature which was used for shape matching [13] at the beginning. We use Parzen window density estimation to modify the shape context feature. All the shape context features are calculated based on the image data rendered from stroke data.

We find that (1) the modified shape context feature performs better than the original shape context feature for stroke pair relation classification; (2) shape context feature achieves very competitive performance for symbol segmentation and symbol pair relation classification; (3) shape context feature can be used as complementary information for different feature sets in the three tasks to improve performance; (4) syntax context feature and MST context feature could improve the stroke pair and symbol pair relation classification if we can get the correct syntax label for the symbol and correct edge label.

4.1 Shape Context Feature

Belongie et al. [13] propose shape context for shape matching and object recognition. The shape context [13, 55] at a given point captures the distribution of the other points relative to it, and therefore provides a globally discriminative characterization. Corresponding points on two similar shapes will have similar shape contexts. Shape context achieves success in shape matching and object recognition [13, 55].

Figure 4.1 shows one example of shape context feature. For a given point, we draw a circle whose center is at the point. The circle is divided into 60 bins, containing 12 equal angle bins and 5 distance bins. The ratios of the radii of the five distance bins to radius of the outmost circle are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$ and 1. For each bin, the number of points within that bin divided by the number of points within the circle is the shape context feature.

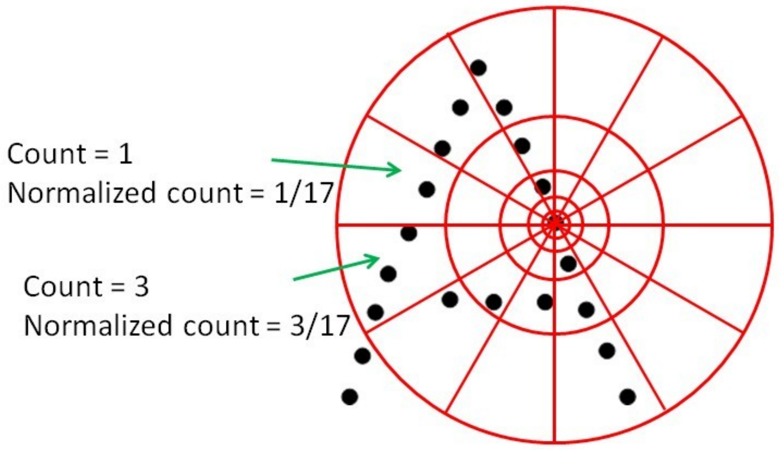


Figure 4.1: One example of shape context feature. This is for stroke data.

Shape context feature is used in many different math expression recognition related tasks. Ouyang and

Zanibbi [65, 66] presented a symbol layout classification method by using shape context feature to classify printed math symbols into seven layout classes [90]: ascender, descender, centered, open bracket, non-scripted, variable range and root. The whole context region is a circle. They divide the whole context region into 60 bins with 5 distance bins and 12 equal angle bins. The ratio of the five distance bins radii moving out from the center are $\frac{1}{16}$, $\frac{1}{8}$, $\frac{1}{4}$, $\frac{1}{2}$, 1, where 1 means the radius of the circle neighborhood.

Hirata and Honda [31] used shape context feature in their graph matching method to automatically label symbols in handwritten math expressions. They use two diagonals passing through the center of shape context to divide the shape context into four sections. The number of points falling inside each section is taken as shape context feature. Therefore, each point captures information of symbols present respectively to its right, up, left, and down sides

Marinai et al. [51, 52] used shape context feature for math symbol retrieval. Their bins are uniform in log-polar space. But the number of bins is not disclosed. They count all the points belonging to each bin as shape context features.

Alvaro and Zanibbi [3] used shape context feature for classifying spatial relationships between symbols or subexpressions in handwritten math expressions. They divide the shape context uniformly in angle and distance. For each bin, the shape context feature value could be -1 (more points from parent symbol/subexpression than child symbol/subexpression), 0 (empty bin), 1 (tie, or more points from child symbol/subexpression than parent symbol/subexpression).

Shape context feature achieved success in math symbol classification [31, 65, 66], math symbol retrieval [51, 52] and spatial relationship classification [3]. These shape context features are all about counting the number of points within the bin. Therefore, all those features are discrete. We design a continuous shape context feature based on Parzen Window for stroke pair relation classification, symbol classification and symbol pair relation classification.

4.1.1 Modification using Parzen Windows

The shape context feature of each bin is the number of points within it divided by the total number of points within the circle. Therefore, each point only affects the bin which it belongs to, and has no effect for the other bins (even for the adjacent bins). We apply Parzen Window density estimation to shape context feature to make the point have a contribution to every bin. The distance between the point and the bin determines the contribution. The point has a large contribution to the bin which it falls within and the nearby bins, but a small contribution to the bins far away.

Parzen-window density estimation is essentially a data-interpolation technique [10,42]. Given a random sample, $\mathbf{x}$, Parzen window density estimation calculates the probability density function (PDF) $p(X)$ from which the sample was derived. It essentially superposes kernel functions placed at each observation. In this way, each observation x_i contributes to the PDF estimate.

Figure 4.2 shows an example of Parzen Window density estimation with one dimensional Gaussian Kernel. Each sample x_i (green point) is taken as the center of the Kernel function (Gaussian distribution). When we want to estimate the $p(x)$ for a given point x, we sum the total of contributions from all observation samples. The Parzen-window estimate is defined as

$$P(x) = \frac{1}{n}\sum_{i=1}^{n} K(\frac{x - x_i}{h}) \tag{4.1}$$

where $K()$ is the kernel function, and $h > 0$ is a smoothing parameter called Parzen Window width.

For Parzen window shape context feature, we use two dimensional Gaussian distribution as the Kernel function. Formula (4.2) shows how to calculate the probability density for each bin. (x_i, y_i) are the coordinates of the points in Figure 4.1, while (x, y) represents the center of a given bin. The Parzen window shape context feature performs better than original shape context feature for stroke pair relation classification as showed in 4.5.2.

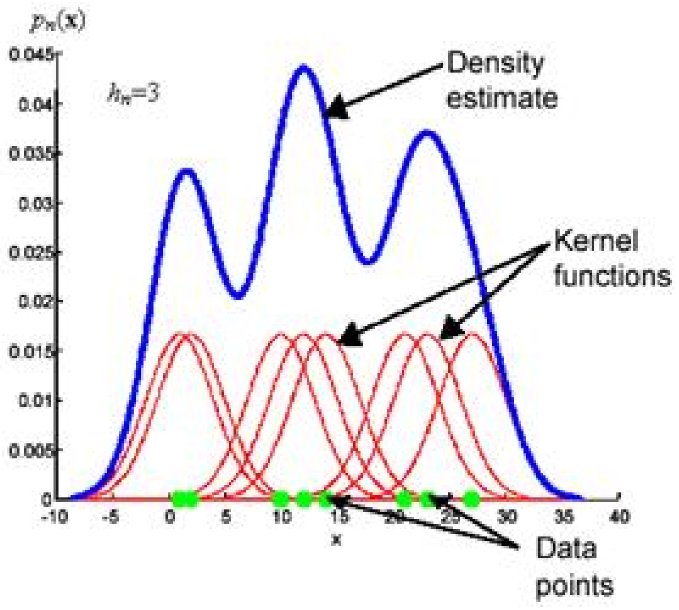


Figure 4.2: Parzen Window density estimation with one dimensional Gaussian Kernel [10].

$$P(x,y) = \frac{1}{n}\sum_{i=1}^{n}\frac{1}{\sqrt{2\pi}\sigma}exp\left(-\frac{(x_i-x)^2}{2\sigma^2}\right)\frac{1}{\sqrt{2\pi}\sigma}exp\left(-\frac{(y_i-y)^2}{2\sigma^2}\right) \quad (4.2)$$

4.2 Preprocessing

To reduce noise and resolution variance between different expressions, we apply preprocessing to render the expression to an image. Our procedure contains four steps: duplicate point filtering, size normalization, smoothing and resampling.

Duplicate point filtering deletes duplicate point which has the same (x,y) coordinates as the previous point because they cannot provide any useful information. In order to remove the influence of writing velocity and devices' difference in coordinate range and resolution, we transform the y coordinate's range to be the same while preserving the width-height aspect ratio. To reduce the noise caused by the stylus' jogging, we smooth the whole expression. Except the first point and the last point of each stroke, we replace the other points' coordinates by the average of the coordinates of current point, the previous point and the next point. Finally we use linear interpolation to resample the expression and render it to an image. The height of the image is 200 pixels, and we preserve the width height aspect ratio. We interpolate 10 points between the consecutive point pair. The duplicate point in the interpolated points would be removed.

4.3 Features and Experiments for Symbol Classification

For symbol classification, we only consider using Parzen window shape context features. For Parzen Window shape context feature in symbol classification, the center of shape context is the center of the symbol's bounding box. The radius is the distance between the center and the furthermost point of the symbol from the center. The points within the context are divided into two categories: points from the current symbol and points from other symbols. For each bin, there are two Parzen Window probabilities; one is based on points of the current symbol or the other is based on points from the other symbols.

The classifier we use for symbol classification is random forest. There are three different experiments based on the what kind symbol label we classify. The symbol label could be original symbol label, symbol layout label, symbol syntax label.

We randomly select 90% of CROHME 2012 training set as training data, and use the other 10% as validation set.

4.3.1 Original Symbol Label Classification

There are 75 different original symbol labels, such as 'a', 'b', '1' and so on.

We run the grid search to find the optimal number of tree and max depth. Grid Search is over number of trees = [10, 20, 30, 40, 50], max depth = [10, 20, 30, 40, 50, 60]. Figure 4.3 shows the error rate with different number of tree and max depth when we classify the original symbol label. When tree num = 50, max depth = 30, the error rate on validation set is minimal (17.36%). Using 75 binary classifiers instead of

1 multiple-class classifier could reduce the error rate to 16.16%. Through analyzing the confusion matrix, we find many classification errors are caused by symbols are classified to the classes with the similar shape, such as the symbol ’times’ $\times$, the capital letter ’X’ and the small letter ’x’.

While the error rates of classifiers [1, 23] with hybrid features and are around 10%. We use Adaboost with SVM [23] as symbol classifier in later Chapters.

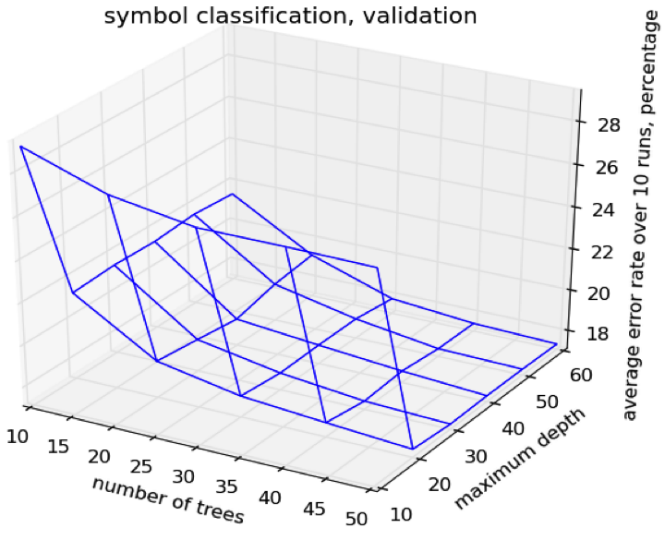


Figure 4.3: Original symbol label classification. Error rates on CROHME 2012 validation set with different number of trees and different max depth. Angle Number = 18, Distance Number = 15.

4.3.2 Symbol Layout Label Classification

We map the 75 symbol classes to 7 symbol layout classes: Non-Scripted, Open Bracket, Root, Variable Range, Ascender, Descender, Centered based on the rule in [90]. The layout label is actually the typographic class. Successfully identifying the layout class of symbols may make symbol layout be recognized without recognizing the specific label of symbols [65, 66].

Figure 4.4 shows the mapping from the original symbol label to the symbol layout label.

Symbol Layout	Symbols
Non-Scripted	$\geq \rightarrow \pm \leq \neq \div \times$ + - / =
Open Bracket	})] { ([
Root	$\surd$
Variable Range	$\int \sum$
Ascender	0-9 A B C F Y X b d f i k t θ
Center	cos $\forall \alpha \phi \infty \beta$ sin log . . . $\exists \in$ tan lim , . ! r a c e j m n x z
Descender	$\gamma \pi$ g p y

Figure 4.4: Mapping between original symbol label and symbol layout label

Figure 4.5 shows the error rate with different number of tree and max depth when we classify the symbol layout label. When tree num = 50, max depth = 50, the error rate on validation set is minimal (11.67%).

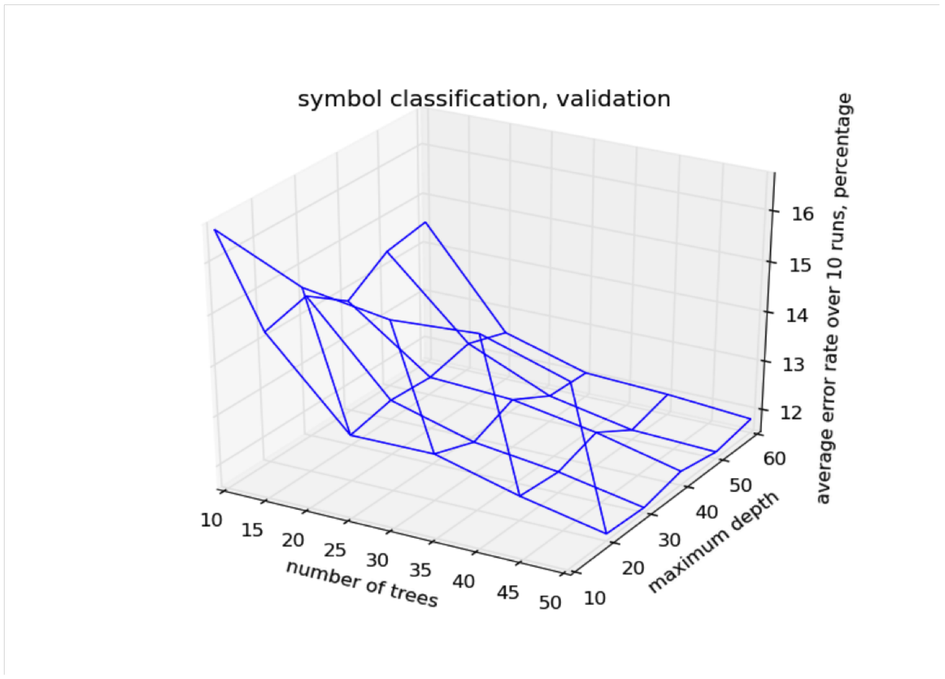


Figure 4.5: Symbol layout label classification. Error rates on CROHME 2012 validation set with different number of trees and different max depth. Angle Number = 18, Distance Number = 15.

4.3.3 Symbol Syntax Label Classification

We map the 75 symbol classes to 3 symbol syntax classes: letter, digit, other. Different symbols with the same syntax label (such as digit or letter) usually have similar behavior. When tree num = 50, max depth =

40, the error rate on validation set is minimal (10.23%). Using 3 binary classifiers instead of 1 multiple-class classifier could reduce the error rate to 8.99%.

4.3.4 Summary

The performance of Random Forest based on shape context features is worse than Random Forest with different features (global features of the symbols, crossing features, 2D fuzzy histograms of points, and fuzzy histograms of orientations) [23]. This shows Parzen window shape context feature is useful for symbol classification. But it is better to combine shape context feature with other features than using shape context feature alone.

We use Adaboost with SVM and the corresponding features in [23] as symbol classifier in later Chapters.

4.4 Features for Stroke Pair Relationship Classification

This section covers all features we consider for stroke pair relation classification. All the features mentioned in this section are calculated based on the preprocessed image data as mentioned in Section 4.2, but not the stroke data (raw input data).

4.4.1 Parzen Window Shape Context Feature for Stroke Pair

Given the stroke pair i, j, the center of the two bounding box centers is the shape context center. The longest distance for all the points of the stroke pair to the center is the radius of the circle. The points following in the shape contexts are divided into 3 categories: points from parent stroke, points from child stroke and points from other strokes. The number of bins is the product of the number of angle (M) and the number of distance (N). The shape context is divided into bins equally in angle and distance, the same as [3].

For each bin, three sets of points are used to calculate three probability densities. For a shape context with M angles and N distances, we will have M*N*3 probability densities as the features. For each point sets, each point is taken as the center of a Gaussian distribution, and the current window width (standard deviation) is $\frac{1}{5}$ of the shape context radius.

Plot of Parzen Window Shape Context Feature

Figure 4.6 shows an example of shape context feature based on Parzen window. Different color represents different points. Red represents points from parent stroke; green represents points from child stroke; blue represents points from other strokes. The parent stroke (red) is vertical stroke of '+' and child stroke (green) is '1', and other strokes are blue. Intensity or brightness indicates the feature value in different bins.

The points from the parent stroke are in the left half of the circle, and they are from top to bottom. Therefore, the red bins stay at the left part, and their shape is kind of vertical. The points from the child stroke are in the right half of the circle, and they are from top to bottom too. Therefore, the green bins stay at the right part, and their shape is kind of vertical too. The points from the other stroke are in the left half of the circle, and they are from left to right. Therefore, the blue bins stay at the left part, and their shape is kind of horizontal.

4.4.2 MST Context Feature and Syntax Context Feature

The motivation of MST Context Feature and Syntax Context Feature is we want to use the results of symbol classification and layout analysis to improve symbol segmentation. We start from perfect symbol classification and layout analysis.

For a given expression, we use its ground truth symbol layout tree. For each symbol pair in the ground truth symbol layout tree, we interpolate 100 points linearly between the bounding box centers of the two symbols. The label of those interpolated edge points depends on the label of the edge. MST context feature is calculated based on these interpolated edges points, just like shape context feature is calculated based on

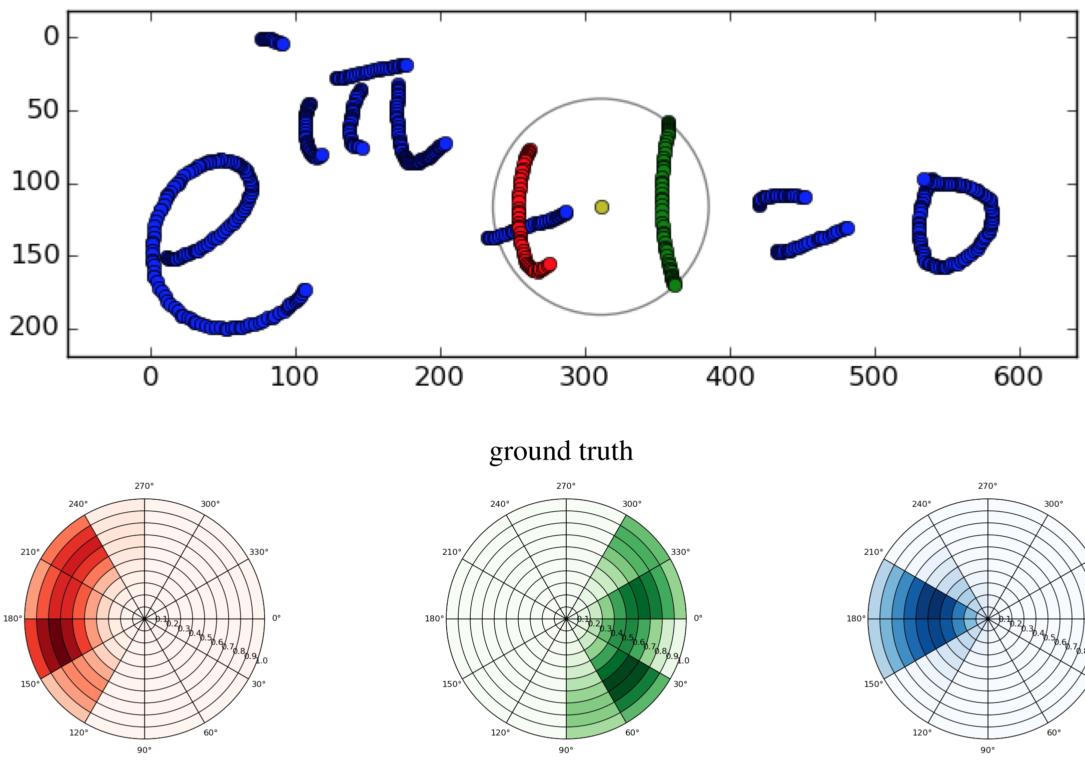


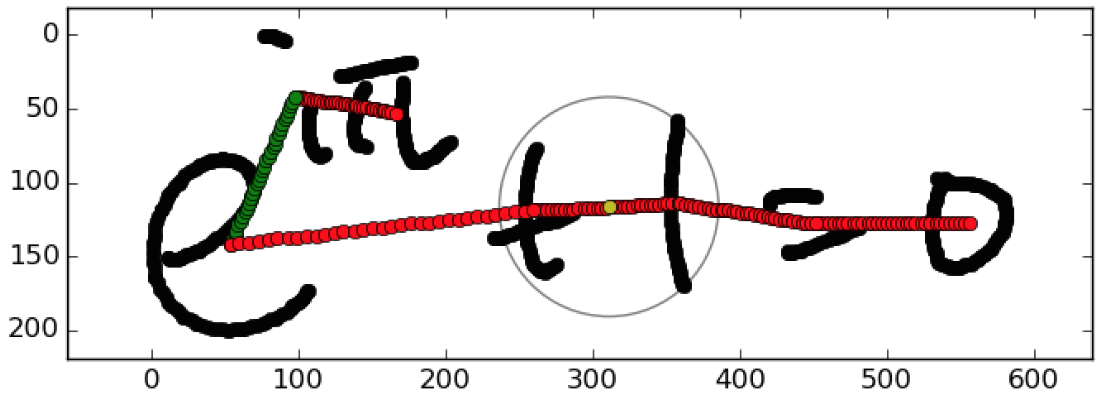
Figure 4.6: An example of shape context feature based on Parzen window.

the interpolated points on the stroke.

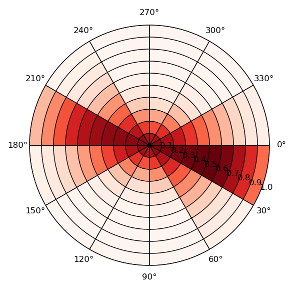
Figure 4.7 shows an example of Parzen Window MST context feature. Different color represents points from different edges. Red represents points from edge with Right adjacent label (Right or Inside); green represents points from edge with script label (superscript or subscript); blue represents points from edge with vertical label (above or below).

The edges used for calculating Parzen Window MST context feature are between symbol pair instead of stroke pair. The edges points which fall within the shape context are used to calculate three different Parzen Window MST context features.

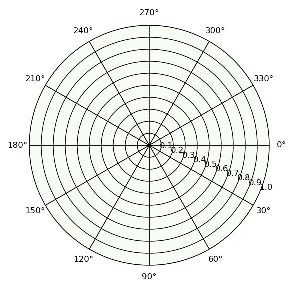
For the shape context in Figure 4.7, there are only edges points which are from edges with 'Right' label, but no edges points from 'script' edge or 'vertical' edge. Therefore Figure 4.7 (c) (d) are empty.



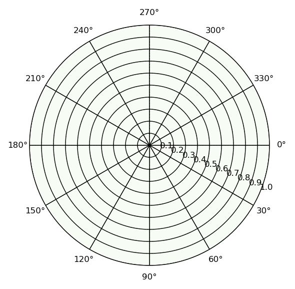
(a) ground truth



(b) Points from Right adjacent edge



(c) Points from script edge



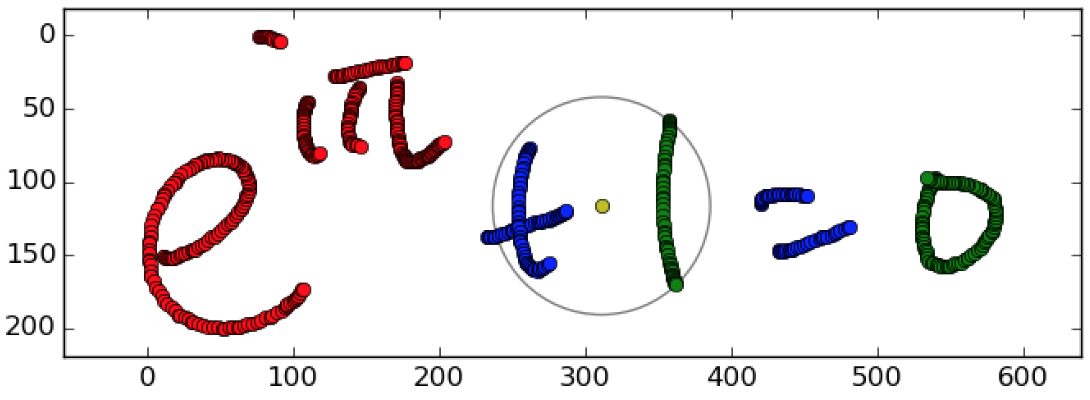
(d) Points from vertical

Figure 4.7: An example of MST context feature based on Parzen window. Stroke pair is vertical stroke of '+' and symbol '1'.

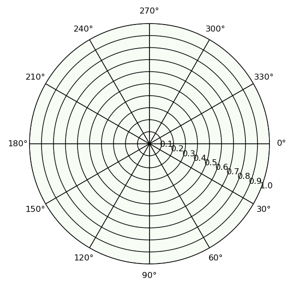
For shape context feature, points are divided into three categories based on which stroke the point is from: points from parent stroke, points from child stroke, and points from other strokes. Each stroke belongs to a symbol and has the symbol's label as its symbol label. It is natural to think of a new context feature which divides points into different categories based on its symbol label. The number of symbol class is big. If we divide points into different categories based on its original symbol label, the context features would have large dimensions and most of the features are 0.

We map different symbol label to its corresponding syntax class label (digit, letter, other) and propose syntax context feature. One benefit of using syntax class label is that the number of syntax class label is small. The other benefit is different symbols with the same syntax label (such as digit or letter) usually has similar behavior.

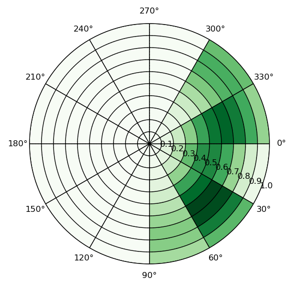
Figure 4.8 shows an example of Parzen Window syntax context feature. Different color represents different points. Red represents points from stroke with letter label; green represents points from stroke with digit label; blue represents points from stroke with other label.



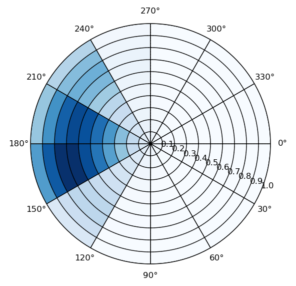
ground truth



Points from letter stroke



Points from digit stroke



Points from other stroke

Figure 4.8: An example of syntax context feature based on Parzen window. Stroke pair is vertical stroke of '+' and symbol '1'.

4.4.3 Geometric Feature

All the stroke pair features used in previous stroke pair relation methods are geometric features. Winkler et al. [46] use minimum distance, horizontal overlapping of the bounding box, distance and offset between the beginning points and end points, backward movement and parallelity of two successive strokes. Parallelity is actually the angle between two vectors. Each vector represents one stroke. The vector is from the first point of the stroke to the last point of the stroke. Shi et al. [68] use horizontal distance, size difference and vertical offsite. Toyozumi et al. [76] use the minimum point distance. MacLean et al. [48] use the overlapped area. Alvaro et al. [3] use some geometric features about the bounding box.

We use all the geometric features mentioned above and design some other geometric features: distance between bounding box centers, distance between averaged centers (the coordinates of averaged center are the average of the coordinates of all points of the stroke), maximal point pair distance (two points are from different strokes of the stroke pair), horizontal offset between the ending point of the first stroke and the beginning point of the second stroke, vertical distance between bounding box centers, writing slope (slope is the angle between the horizontal line and the line connecting the last point of the current stroke and the first point of the next stroke) and writing curvature (curvature is angle between the line connecting the first point and last point of the current stroke and the line connecting the first point and last point of the next stroke). We normalize all the geometric features except parallelity, writing slope and writing curvature to make them between [0, 1].

4.4.4 Time Gap Feature

For stroke pair (i, j), time gap feature is $j - i$. The time gap distributions for different relations are quite different from each other. For the stroke pair with relation merge, more than 80% of them have time gap 1 or -1, which means the stroke pair is consecutive in time series. Stroke pairs with relation Right or Sub have a larger time gap than stroke pair with relation merge or inside.

4.4.5 Symbol Classification Scores

For the given stroke pair, each stroke is taken as a symbol and classified by the symbol classifier. The symbol classification scores are considered as features for the stroke pair relation classification.

4.5 Experiments for Stroke Pair Relation Classification

Stroke pair relation classification can be binary stroke pair relation classification or multiclass stroke pair relation classification. Binary stroke pair relation classification only classifies stroke pair relation as merge or split; while multiclass stroke pair relation classification classifies the stroke pair relation as one of the eight classes: merge (*), no-relation (_), Right (R), Subscript (Sub), Superscript (Sup), Above (A), Below (B), Inside (I).

The stroke pair data could be different too, depending on what graph representation. The graph representation could be time series graph, 3 nearest neighbor (3NN) graph, and line of sight graph.

The classifier we consider for stroke pair relation classification could be AdaBoost and Random Forest.

4.5.1 Binary Stroke Pair Relation Classification with AdaBoost

This subsection presents a binary stroke pair relation classification method based on AdaBoost with confidence weighted predictions.

We use Part3 training data of CROHME 2012 as training data. The Part3 training data contains 16970 symbols but only 20 symbols contain unsuccessive strokes (the rate is 0.12%). So for this dataset, the time successive assumption is a very good assumption which could make the segmentation much easier and faster but only decrease the segmentation rate a little.

Like many previous segmentation methods, we use time series graph as graph representation, which assuming a symbol can only contain successive strokes. But our method does not specify the number of strokes a symbol can have and does not use any language model. Given an expression with n strokes, our segmentation method just considers merging or splitting the $n-1$ stroke pairs $(S_1, S_2), (S_2, S_3), ..., (S_{n-1}, S_n)$ in time sequence and only provides one segmentation interpretation. The time complexity of our segmentation method is $O(N^2)$. Therefore our segmentation method is efficient, in terms of computation.

Expressions are preprocessed to reduce noise and resolution variance first. Then for each stroke pair, we extract 351 features, including 21 geometric features, three types of shape context features (each shape context feature has 60 dimensions) and two sets of classification scores (each set of classification scores has 75 dimensions). After the feature extraction, we apply PCA to the 351 features and choose the first 100 components. The first 100 components are used to train the classifier by using AdaBoost with confidence weighted predictions [67].

Multi-scale Shape Context Feature

We design three shape context features: stroke pair, local neighborhood and global. Figure 4.9 shows one example of multi-scale shape contexts. For the three kinds of shape contexts, the circle center is the center of the bounding box of the current stroke, but their radii will be different. **Stroke pair shape context** only considers the current stroke and the following stroke in time sequence. The radius for stroke pair shape context is the maximal distance between the points in the two strokes and the circle center. **Local neighborhood shape context** considers the current stroke and its three nearest neighbor strokes. The distance between two strokes is the minimal point distance between them. The radius for neighbor shape context is the maximal distance between the points in the four strokes and the circle center. **Global shape context** covers the whole expression. The radius is the maximal distance between the points in the whole expression and the circle center. Each one of the three circles will be divided into 60 bins with 5 distances and 12 angles. The MSSCF capture the successive stroke pair, local neighborhood and the whole expression separately and include information at different level. Compared to a single shape context feature, the MSSCF contains much more information which is helpful for deciding to merge or split the successive stroke pair.

For different symbol classes, the strokes' layout and neighborhoods are different. We add the classification scores as additional features to use the symbol class information. The classifier we use is similar to the HMM classifier in [34], but with additional angular feature and global features [32]. We design a continuous

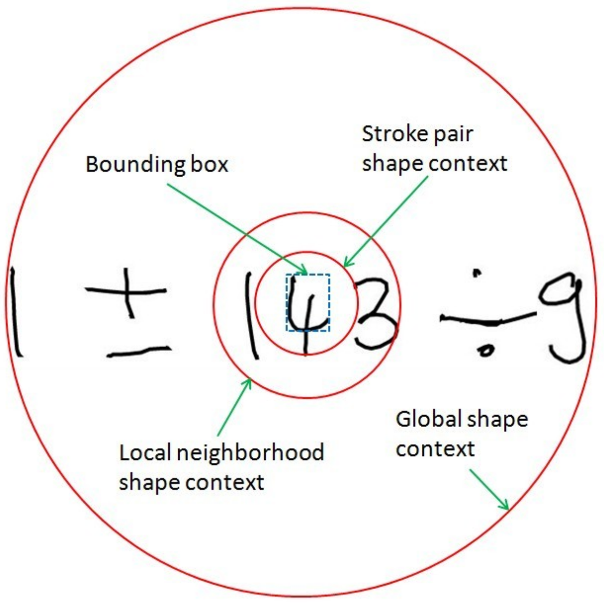


Figure 4.9: Example of multi-scale shape contexts (current stroke is the top stroke of the symbol 4).

left to right HMM for each symbol class. A variant of segmental K-means is used to get initialization of the Gaussian Mixture Models' parameters which represent the observation probability distribution of the HMMs. We take the current stroke (the first stroke of the stroke pair) as one symbol candidate and also take the stroke pair as the other symbol candidate. For each symbol candidate, the classification scores with a fixed order over all the classes are used as features.

Dimensionality Reduction

For each stroke pair, we have 351 features which are based on the stroke pair, such as geometric features, stroke pair shape context features and classification scores, or are centered around the stroke pair, such as local neighborhood shape context features and global shape context features. Among these features, some are redundant. For example, the bins which are far away from the circle center vertically in global shape context usually contain no points, and those shape context features are always 0. We use PCA to reduce noise and redundancy. Figure 4.10 the first 100 components account for the 99.86% variance of all the features. The ratio of the variance of the first five components are 45.24%, 25.75%, 18.96%, 3.90% and

1.11%. The ratio of the variance of the other components are all less than 1.00% and form a long tail. The shape of the plot about the ratio of the variance of each component in descending order is close to an 'L'. The first 100 components are used as new features to train the classifier.

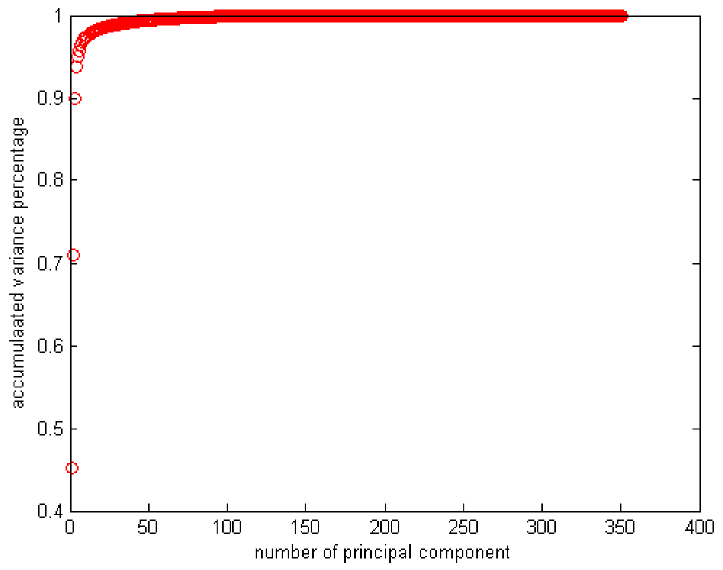


Figure 4.10: Accumulated variance percentage and the number of component.

AdaBoost

AdaBoost is an ensemble classifier. It produces a strong classifier by combing many different base learners [67]. Algorithm 4 shows the pseudocode of AdaBoost. The basic idea behind AdaBoost is to choose training datasets for the base learner in such a way to force the base learner to infer something new about the data each time [67]. AdaBoost proceeds in rounds. On each round, AdaBoost maintains a weight distribution D_t over the training samples to choose the training datasets for the base learner. The weight of a sample measures the importance of the sample on the current round. All the weights are set equally at the beginning. But the weights of wrongly classified samples increase on each round. This means difficult samples get successively higher weight which forces the base learner to focus its attention on them.

The quality of a base classifier is measured by its summed weighted error ϵ_t. The weight or confidence

Algorithm 4 AdaBoost [67]

```
1: Given: $(x_1, y_1), \cdots, (x_m, y_m)$ where $x_i \in X, y_i \in \{-1, +1\}$.
2: Initialize: $D_1(i) = 1/m$ for $i = 1, \cdots, m$.
3: for $t = 1, \cdots T$: do
4:     Train weak learner using distribution $D_t$.
5:     Get weak hypothesis $h_t : X \to \{-1, +1\}$.
6:     Aim: select $h_t$ to minimalize the weighted error: $\epsilon_t \doteq Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
7:     Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$.
8:     Update, for $i = 1, \cdots, m$:
9:         $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$
10:    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
11: end for
12: Output the final hypothesis:
13:     $H(x) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

of the base classifier α_t is calculated based on ϵ_t. Intuitively, α_t measures the importance that is assigned to the base learner h_t. The lower the summed weighted error ϵ_t, the higher the weight α_t of base learner. Finally, AdaBoost combines the many base classifiers by a simple weighted vote of the base classifiers.

We use AdaBoost with confidence-rated predictions, and the weak learner is a decision stump. Decision stump is a base learner which chooses a threshold as cut off point. It classifies all samples to the left of the threshold as positive or negative, and all samples to the right as negative or positive. On each iteration of AdaBoost, we try to minimize the normalization factor

$$Z_t \doteq \sum_{i=1}^{m} D_t(i) exp(-\alpha_t y_i h_t(x_i)), \tag{4.3}$$

where D_t is the weight distribution over the all training samples; α_t is a parameter which is used to update $D_t(i)$; $y_i \in \{-1, +1\}$ ($+1$ represents merge while -1 represents split); h_t is the weak hypothesis.

There are 23424 training samples for our classifier, therefore it is hard to calculate h_t and a_t in general method. By assuming the weak learner can scale any weak hypothesis h by any constant factor $\alpha \in R$ freely without loss of generality, expression (1) can be simplified by folding a_t into h_t [67]. We choose α to be 1. Then our goal becomes minimizing

$$Z_t = \sum_{i=1}^{m} D_t(i) exp(-y_i h_t(x_i)). \tag{4.4}$$

Our classifier only contains two classes: merge and split. The weak learner decision stump finds a threshold to divide the whole training samples into two parts. In each iteration, for all samples within each block X_j, weak hypothesis h is equal to some fixed value c_j. The normalization factor in equation (2) is minimized when

$$c_j = \frac{1}{2} ln \left(\frac{W^j_{\text{merge}} + \epsilon}{W^j_{\text{split}} + \epsilon} \right), \tag{4.5}$$

where W^j_{merge} is the summed weight of merge samples which fall in block j while W^j_{split} is the summed weight of split samples which fall in block j. ϵ in expression (3) is typically chosen to be on the order of $\frac{1}{m}$ and m is the number of training samples [67]. We choose ϵ to be $\frac{1}{23424}$.

Dataset and experiment results

The dataset we use here is the Part-III dataset in CROHME 2012 [57]. We use the training set of Part III data to train the classifier and the test set for test. The training set has 1338 expressions and 23424 successive stroke pairs, in which 7773 stroke pairs should be merged while 15651 should be split. The test set has 488 expressions and 8379 successive stroke pairs, in which 2459 stroke pairs should be merged while 5920 should be split. training set and test set have the same 75 symbol classes.

We train the classifier 10000 iterations, and Table 4.1 shows the minimal classification error rates on the successive stroke pairs of training set and test set with different features among the 10000 rounds.

From Table 4.1, we can find that MSSCF performs better than G (geometric features) on training set but

Table 4.1: Minimal error rates (corresponding number of iteration) on the successive stroke pairs of CROHME 2012 training set and test set with different features among the 10000 rounds (G: geometric features, MSSCF: multi-scale shape context features, SPSCF: stroke pair shape context features, LNSCF: local neighborhood shape context features, GSCF: global shape context features, C: classification scores). The features are the first 100 components.

features	training set		test set	
G	7.60%	(9995)	12.85%	(390)
MSSCF	1.50%	(9999)	14.49%	(822)
SPSCF	5.72%	(9995)	15.85%	(5366)
LNSCF	14.48%	(9909)	25.27%	(542)
GSCF	15.48%	(9965)	29.26%	(12)
G + MSSCF	0.00%	(9946)	10.36%	(6604)
G + MSSCF + C	0.00%	(7147)	8.80%	(4111)

worse on test set. This shows MSSCF is easy to overfit. MSSCF performs much better than either one of three shape context features (SPSCF, LNSCF and GSCF) on both training and test set. If we just use one of the three shape context features, the larger the radius is, the worse the performance. The reason is that the shape context features will have less variance and become less discriminative when the scope become larger. Using G and MSSCF can get better performance on training set and test set than using either one of them alone. Adding classification scores to G and MSSCF can improve the performance further.

The 'corresponding number of iteration' in Table 4.1 means the number of iterations when error rate is minimal. When we use multi-scale shape context features (MSSCF), the error rate for the classifier is minimal (1.50%) after 9999 iterations on training set; and achieves the lowest (14.49%) after 822 iterations on test set. For the classification error rate curve on training set and test set over the 10000 iterations by using G + MSSCF + C, the error rates on training set and test set drop fast in the first 2000 iterations. After 2000 iterations, the training error rates continue to drop and are almost 0 after 6000 iterations, while the testing error rates just change a little. The error rate curves by using the other sets of features (such as G or MSSCF or G+MSSCF) have the similar shape. This shows that AdaBoost with confidence-rated predictions will not overfit the training set. Figure 4.11 shows error rates on training set and test set over 10000 iterations of AdaBoost.

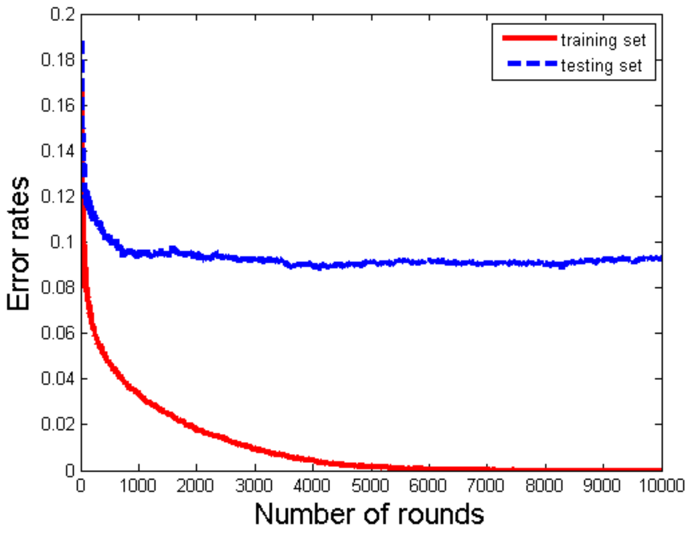


Figure 4.11: Error rates on CROHME 2012 training set and test set over 10000 iterations with G + MSSCF + C.

Figure 4.12 shows the precision and recall on test set with different features. Recall is the same as symbol segmentation rate in [57]. We can find that the precision and recall for different features on training set or test set is very close. It is easy to find that using G + MSSCF + C gets the best performance. By using all these three sets of features, our segmentation method achieves 99.88% recall, 99.76% precision on training data and 84.95% recall, 84.79% precision on testing data.

Through analyzing the experiment results, we find many segmentation errors happen to multi-stroke symbols, such as cos, sin, log, lim, ellipsis . . . and so on. Those multi-stroke symbols are easy to be over-segmented. Symbol segmentation rates for cos, sin, log, lim, ellipsis . . . are 38.37%, 48.00%, 44.44%, 34.69% and 0. For the single-stroke symbols, which have intersections with nearby symbols or are very close to nearby symbols, they are easy to be under-segmented. Therefore the symbol segmentation rate can be improved by adding prerecognition. Prerecognition means trying to find symbol candidate which is highly possible to be a symbol and this symbol is not a part of other symbol. The strokes of the symbol candidate will be isolated from the other strokes.

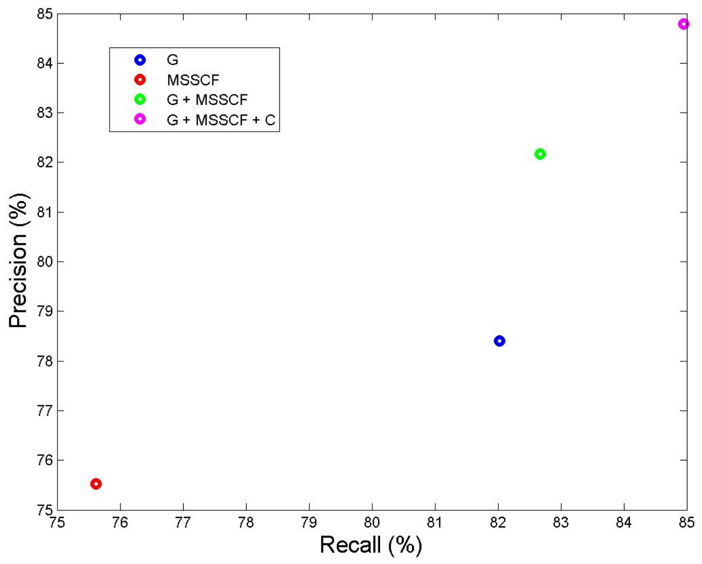


Figure 4.12: Precision and recall on CROHME 2012 test set with different features (G: geometric features, MSSCF: multi-scale shape context features, C: classification scores).

4.5.2 Multiclass Stroke Pair Relation Classification with AdaBoost.MH and Random Forest

This section is about multiclass stroke pair relation classification. The relation between the stroke pair is among the 8 classes: merge (*), no-relation (_), Right (R), Subscript (Sub), Superscript (Sup), Above (A), Below (B), Inside (I). The classifier used in this section is AdaBoost.MH. The AdaBoost.MH has 200 iterations. At the beginning, we only use Parzen window shape context feature. Then we try add new features to the shape context feature to see whether adding new feature could help. The same as previous section, we apply PCA to the features and use the first 100 PCA components.

Dataset

The training data is the part 3 training data of CROHME 2012. It has 1338 expressions. In order to tune the parameters, such as the number of angle, the number of distance and the window width. The whole training set is divided into two parts (training set and validation set) randomly. The training set has 1221 expressions (90%), while the validation set has 117 expressions (10%).

Table 4.2: Distribution of the stroke pair relation in 3 nearest neighbor graph for CROHME 2012 dataset.

Dataset	expression	stroke pair	_	*	R	Sub	Sup	A	B	I
Training	1221	67821	32442	16523	13689	1764	1528	734	707	434
Validation	117	6441	3086	1553	1323	147	158	79	82	13
Testing	488	26573	13078	5835	6089	262	596	324	336	53

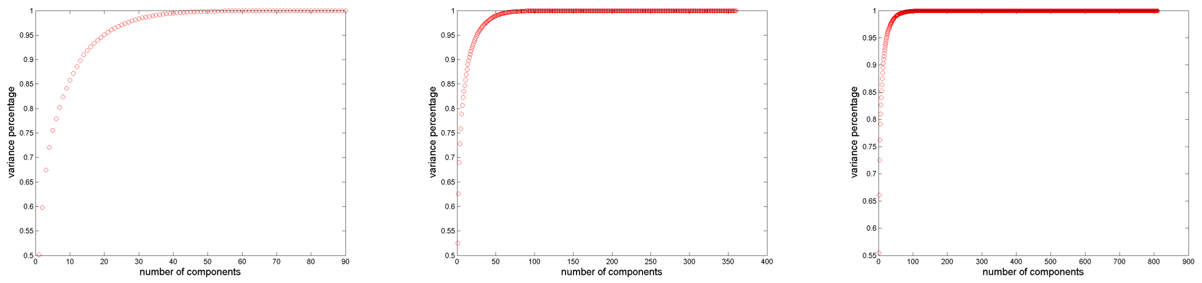
Table 4.2 shows the distribution of the stroke pair relation for the training set and validation set in 3NN graph.

Angle Number and Distance Number

The angle number (M) and distance number (N) determines the number of bins in the shape context. The number of bins in the shape context equals to $M \times N$.

We set M = 6, 12, 18 and N = 5, 10, 15. Then we apply PCA to the features to reduce the dimensionality.

Figure 4.13 shows the plot of variance percentage against the number of components with different combination. For each (angle number, distance number) combination, we use the first 100 components as the features. For all combinations, it is easy to find that the first 100 components have more than 95% variance.



$(M = 6, N = 5)$ $(M = 12, N = 10)$ $(M = 18, N = 15)$

Figure 4.13: Variance percentage against the number of components with different (angle number, distance number) (M, N) combinations.

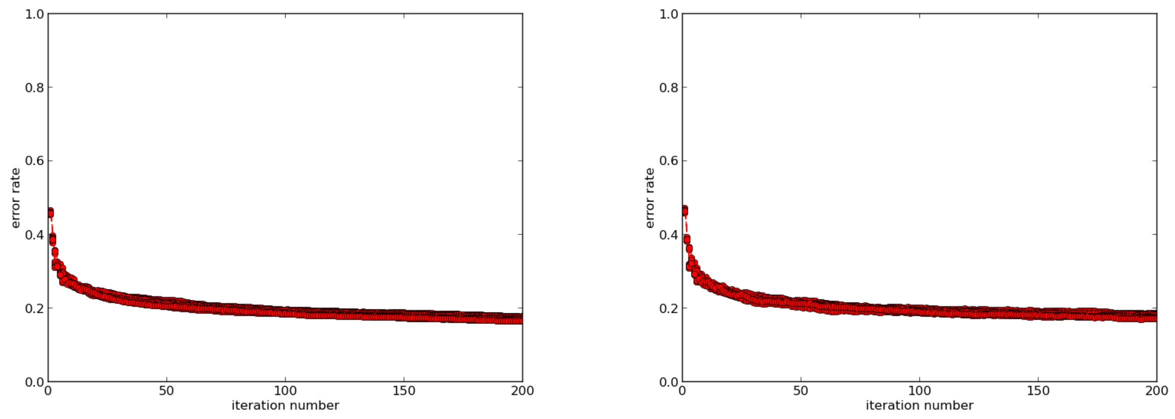
Table 4.3 shows the minimal error rates on both training set and validation set with different (angle number, distance number) (M, N) combinations. We can find the more bins, the lower the minimal error rate. For the same angle number (M) and distance number (N), the minimal error rates between training set and validation set are small, this shows there is no overfitting for AdaBoost.

Table 4.3: Minimal error rates of multiclass stroke pair relation classification on CROHME 2012 training set and validation set with different (angle number, distance number) (M, N) combinations.

	training set	validation set
M = 6, N = 5	17.60%	18.41%
M = 6, N = 10	17.59%	18.18%
M = 6, N = 15	17.46%	18.06%
M = 12, N = 5	16.92%	17.45%
M = 12, N = 10	16.66%	17.44%
M = 12, N = 15	16.47%	17.14%
M = 18, N = 5	16.82%	17.34%
M = 18, N = 10	16.51%	17.19%
M = 18, N = 15	16.39%	16.89%

Figure 4.14 shows the error rate for the classifier after each iteration of AdaBoost on training set and validation set when (angle number, distance number) is (18, 15). The AdaBoost.MH classifier has 200 iterations. We can find that the error rate plots on training set and validation set have the similar shape.

Figure 4.15 shows the confusion matrix on training set and validation set when (angle number, distance number) is (18, 15). The color of each matrix element reflects the corresponding rate and the color bar on the right side indicate the relations between color and rate. Suppose $M_{i,j}$ represents the number at the i-th row and j-th column in the confusion matrix whose size is $K * K$, the rate at the same location $R_{i,j} = \frac{M_{i,j}}{\sum_{j=1}^{K} M_{i,j}}$. Therefore, the rate $R_{i,j}$ along the main diagonal is the classification rate for each class. It is easy to find that *, Sub, B and I have lower classification rate than the rest 4 classes (-, R, Sup, A).



(a) training dataset.

(b) Validation dataset.

Figure 4.14: Error rates for the classifier after each iteration of AdaBoost on CROHME 2012 training set and validation set with angle number = 18, distance number = 15, and first 100 PCA components as features.

Table 4.4 shows detailed classification rate for each class on training set and validation dataset. For the errors about class *, most of them are taken as no relation (_) or R. For the errors about class Sub, most of them are taken as R. This is easy to understand because the differences between R and Sub is subtle. For the errors about class B, most of them are taken as _. The class I has the lowest classification rate, and most of the errors are taken as _.

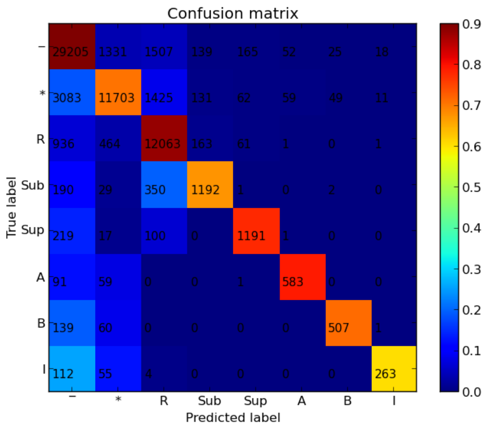
Table 4.4: Classification rate for each class on CROHME 2012 training set and validation set (M = 18, N = 15).

Dataset	_	*	R	Sub	Sup	A	B	I
Training	90.02%	70.83%	88.12%	67.57%	77.95%	79.43%	71.71%	60.60%
Validation	89.34%	69.03%	88.89%	61.90%	73.42%	77.22%	74.39%	46.15%

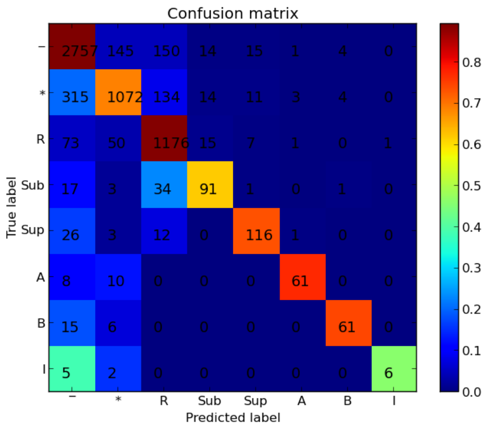
Parzen Window Shape Context Feature VS Original Shape Context Feature

Parzen window shape context feature is calculated as showed in Formula (4.2), while original shape context feature is calculated as showed in Figure 4.1.

Table 4.5 shows the error rate of Parzen window shape context feature is more than 1% lower than the



(a) training dataset.



(b) Validation dataset.

Figure 4.15: Confusion matrixes of AdaBoost.MH with 200 iterations on CROHME 2012 training set and validation dataset. Angle number = 18, distance number = 15, and features are the first 100 PCA components.

error rate of original shape context feature on validation set for the stroke pair relation classification. We use Parzen window shape context feature instead of original shape context feature, and shape context feature means Parzen window shape context feature in the rest of this thesis.

Table 4.5: Error rates comparison between Parzen window shape context feature and original shape context feature on CROHME 2012 training set and validation dataset for multiclass stroke pair relation classification. Angle Num is 18 and Distance Num is 15. Classifier is AdaBoost.MH with 200 iterations.

features	Training	Validation
Original shape context feature	16.39%	16.89%
Parzen window shape context feature	12.70%	15.56%

Parzen Window Width

To choose the Parzen window width, we set Angle Number = 18, Distance Number = 15, and use the AdaBoost.MH to classify the stroke pair relation with different Parzen window width. The top 100 PCA components are used as features, and AdaBoost.MH has 200 iterations. The top 100 PCA components are used as features, and AdaBoost.MH has 200 iterations.

Table 4.6: Error rates comparison with different Parzen window width on CROHME 2012 training set and validation dataset for multiclass stroke pair relation classification. Angle Number is 18 and Distance Number is 15. N means the Parzen window width is $\frac{1}{N}$ of the shape context's radius. Classifier is AdaBoost.MH with 200 iterations.

N	training set	validation dataset
0.5	13.42%	15.86%
1	13.13%	15.78%
5	12.70%	15.56%
10	13.71%	20.38%
20	14.96%	36.59%
50	17.48%	60.66%
100	19.89%	49.62%

Table 4.6 shows when $N = 5$, the AdaBoost stroke pair classifier achieves the lowest error rate on both training set and validation dataset. Therefore, we choose Parzen window to be $\frac{1}{5}$ of the shape context's radius.

Adding Geometric Feature and Time Gap Feature to Parzen Window Shape Context Feature

After determining the angle number, distance number and Parzen window width, we compare different feature sets.

Table 4.7 shows that geometric feature can reduce the error rate on validation set by almost 2%; and time gap feature can reduce another 1%.

Table 4.7: Error rates comparison among different features on CROHME 2012 training set and validation dataset for multiclass stroke pair relation classification. Angle Number is 18 and Distance Number is 15. Shape context feature is Parzen window shape context feature. Classifier is AdaBoost.MH with 200 iterations.

features	training set	validation dataset
Shape context feature	12.70%	15.56%
Shape context feature + geometric feature	11.36%	13.64%
Shape context feature + geometric feature + time gap	10.86%	12.75%

Classification Confidence

Our previous work [33] shows that symbol classification scores can help symbol segmentation. We use AdaBoost with C4.5 decision trees [23] as symbol classifier. For the given stroke pair, each stroke is taken as a symbol and classified by the symbol classifier. The symbol classification scores are added as features for the stroke pair relation classification.

Table 4.8: Error rates comparison between with and without symbol classification scores on CROHME 2012 validation dataset for multiclass stroke pair relation classification. Angle Number is 18 and Distance Number is 15. Classifier is AdaBoost.MH with 200 iterations.

features	validation dataset
Shape context feature	15.56%
Shape context feature + classification scores	44.86%

The reason why adding classification confidence makes the performance worse is that the classification confidence is much larger than the Parzen window shape context features. The summation of the 75 confidence scores for a symbol is 1. In addition, the classification confidence produced by the symbol classifier (AdaBoost with C4.5 decision trees) is quite skewed. The top 5 classification results have very high classification confidence scores, while the classification confidence scores for other classification results are nearly

0. For the summation of the 810 Parzen window shape context features of each stroke pair, the average of the summation is around 0.02. Therefore, the first N (100) PCA components mainly capture the variance of the classification confidence.

In our previous work [33], the summation of the 180 shape context features is 1. This makes the classification scores are at the same scale as the shape context features, and are useful as auxiliary features for stroke pair relation classification.

The other reason is that there are 8 classes for the stroke pair relation classification while there are only 2 classes in the symbol segmentation [33].

In order to make all the features have the same scale, we apply two normalizations to the feature values. One is standard score, which uses $\frac{x-\mu}{\sigma}$ instead of x. The other one is linearly scale which convert the range of feature values to [-1,1] by using $\frac{2\times x-(x_{max}+x_{min})}{x_{max}-x_{min}}$ instead of x. But neither these two normalization helps.

Therefore, we use Parzen window shape context feature, geometric feature and time gap feature as features for stroke pair relation classification.

Different Classifiers

With the first 100 PCA components of Parzen Window shape context feature + geometric feature, we compare AdaBoost.MH and Random Forest. AdaBoost.MH has 200 iterations and decision stump is weak learner. For Random Forest, there are 10 trees in the Random Forest. The number of features to consider when looking for the best split is sqrt(n), where n is the total feature number. The function to measure the quality of a split is Gini impurity. For max depth, nodes are expanded until all leaves are pure or contain less than 2 samples. Bootstrap samples are used when building trees. The sample size is the same as the number of samples in the training dataset.

Table 4.9: Error rates comparison between Adaboost.MH and Random Forest on CROHME 2012 training set and validation dataset for multiclass stroke pair relation classification.

classifier	training set	validation set
AdaBoost.MH	11.36%	13.64%
Random Forest	0.34%	8.54%

From Table 4.9, we can find Random Forest performs much better than AdaBoost.MH. Therefore, we choose Random Forest as the stroke pair relation classifier.

Summary

For multiclass stroke pair relation classification, random forest performs much better than AdaBoost.MH for stroke pair relation classification, and therefore we choose Random Forest as stroke pair relation classifier. Geometric features and time gap feature can help improve the performance, but symbol classification confidence scores can not. We use Parzen window shape context feature, geometric feature and time gap feature for stroke pair relation classification.

4.5.3 Binary Stroke Pair Relation Classification with Random Forest

This section is about binary stroke pair classification for stroke pair in line of sight graph. The relation between the stroke pair is only merge or split. We use Parzen window shape context feature, geometric feature and time gap feature as features. The classifier is random forest. Random forest selects features randomly at each node during constructing each tree. We do not apply PCA but use all the raw features for random forest.

Adding MST Context Feature and Syntax Context Feature

We use the ground truth symbol layout tree to calculate the MST Context Feature and Syntax Context Feature in Table 4.10. This means the edge label and syntax class label are 100% accurate.

Table 4.10 shows adding MST Context Feature and Syntax Context Feature could reduce the error rate for binary stroke pair relation classification. It is easy to find that the minimal error rate on validation set is 2.1% with the original feature (Parzen window shape context feature + geometric feature + time gap feature). Adding Parzen window MST context feature reduces the min error rate on validation set to 0.94%. Adding Parzen window MST context feature and Parzen window syntax context feature reduces the min error rate on validation set to 0.37%. These experiment results show Parzen window MST context feature and Parzen window syntax context feature can provide useful information for stroke pair relation classification.

Table 4.10: Error rates of binary stroke pair relation classification on CROHME 2012 training set and validation dataset. Original feature means Parzen window shape context feature + geometric feature + time gap feature. Data is the stroke pair in the line of sight graph. Angle number = 18, Distance number = 15. Classifier is Random Forest.

features	training set	validation set
Original feature	0%	2.11%
Original feature + MST context feature	0%	0.94%
Original feature + MST context feature + syntax context feature	0%	0.37%

If the edge label and syntax class label are not 100% accurate, are MST Context Feature and Syntax Context Feature still useful for binary stroke pair relation classification? We use our symbol classifier and symbol pair relation classifier to produce the syntax class label and edge label for the symbol layout tree. The symbol layout tree uses the ground truth symbol candidate which means the segmentation is perfect, and we only produce the syntax class label and edge label.

With the Parzen window MST context feature and Parzen window syntax context feature which are calculated based on the syntax class label and edge label produced by our classifier, the error rate is higher than without those MST context feature and syntax context feature. The errors of the symbol classification, and symbol pair relation classification bring noisy information to the binary stroke pair relation classification.

We cannot come to the conclusion that only knowing symbol's syntax label and edge's label with 100% accuracy could help symbol segmentation (binary stroke pair relation classification). But our symbol classifier and symbol pair relation classifier are not accurate enough to make MST Context Feature and Syntax Context Feature useful for binary stroke pair relation classification. We would not add Parzen window MST context feature and Parzen window syntax context feature for the stroke pair relation classification.

Radius of Shape Context

The basic radius unit of shape context is the largest distance between the point of the stroke pair and the shape context center. It is long enough and only long enough to cover all the point on the stroke pair. But not only the stroke pair but also the neighbor strokes could help symbol segmentation [33]. We tested longer radius for shape context to cover more points from neighbor strokes.

Figure 4.16 shows an example of shape context radius comparison. With a longer radius, the shape context would cover more points from 'other' strokes, while the points from parent stroke and child stroke would fall into the bins which are close to the shape context center.

Table 4.11 shows that when the shape context radius is 1.5 times of the basic radius unit, the error rate on validation set is lowest. This supports that neighbor strokes could provide useful information for stroke pair relation classification in addition to the parent stroke and child stroke. Therefore, we use 1.5 times of the basic radius unit for the binary stroke pair relation classification.

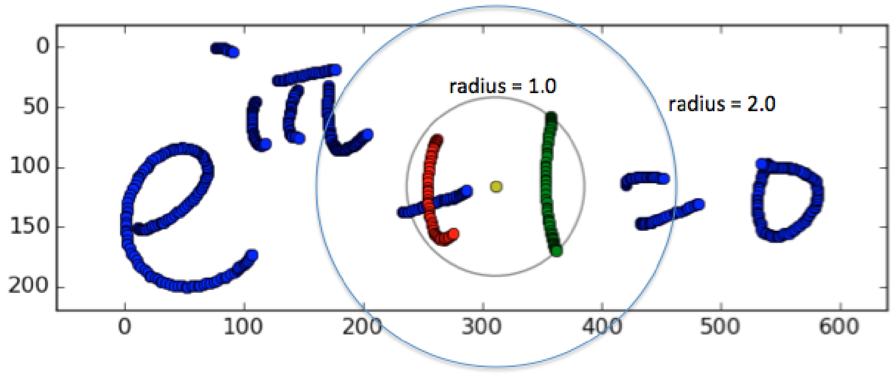


Figure 4.16: An example of shape context radius comparison. The radius of inner circle the same as the basic radius unit, while the radius of outer circle is 2.0 times of the basic radius unit.

Table 4.11: Error rates of binary stroke pair relation classification on CROHME 2012 training set and validation set with different shape context radii. When radius is 2.0, the shape context radius is 2 times of the basic radius unit. Features are Parzen window shape context feature + geometric feature + time gap feature. Data is the stroke pair in the line of sight graph. Angle number = 18, Distance number = 15. Classifier is Random Forest.

radius	training set	validation set
1.0	0%	2.11%
1.5	0%	1.89%
2.0	0%	1.90%
2.5	0%	1.93%
3.0	0%	1.97%

Angle Number and Distance Number

After fixing the radius = 1.5, we compare different (angle number, distance number) combinations. We set Angle Number = [6, 12, 18] and Distance Number = [5, 10, 15].

Table 4.12 shows (angle number, distance number) (6, 5) achieves the lowest error rate on Validation dataset. But the difference among different (Angle number, Distance number) combinations is very small. Therefore, we use Angle number = 6, Distance number = 5 for the binary stroke pair relation classification.

Table 4.12: Error rates of binary stroke pair relation classification on CROHME 2012 training set and validation set with (angle number, distance number) combinations. Features are Parzen window shape context feature + geometric feature + time gap feature. Data is the stroke pair in the line of sight graph. Radius = 1.5. Classifier is Random Forest.

(angle number, distance number)	training set	validation dataset
(6, 5)	0%	1.88%
(6, 10)	0%	1.93%
(6, 15)	0%	1.92%
(12, 5)	0%	1.89%
(12, 10)	0%	1.90%
(12, 15)	0%	1.88%
(18, 5)	0%	1.88%
(18, 10)	0%	1.88%
(18, 15)	0%	1.89%

Random Forest Parameter

After determining the parameters for the shape context, we need to set the parameter for the classifier (random forest). For random forest, there are two important parameters: the number of trees and the max depth of tree. Figure 4.17 shows when we increase the number of tree from 10 to 50, the error rate decreases for different max depth. When tree number is 50, the error rate is minimal. We did experiments which show using more than 50 trees (such as 100 trees or 200 trees) cannot decrease the error rate further. When the number of tree is 50, we find max depth = 40 gets the lowest error rate (1.88%). Therefore, we choose number of trees = 50, and max depth = 40.

We use different split functions (Gini impurity and entropy), the performance is almost the same and we use Gini impurity for the rest of thesis.

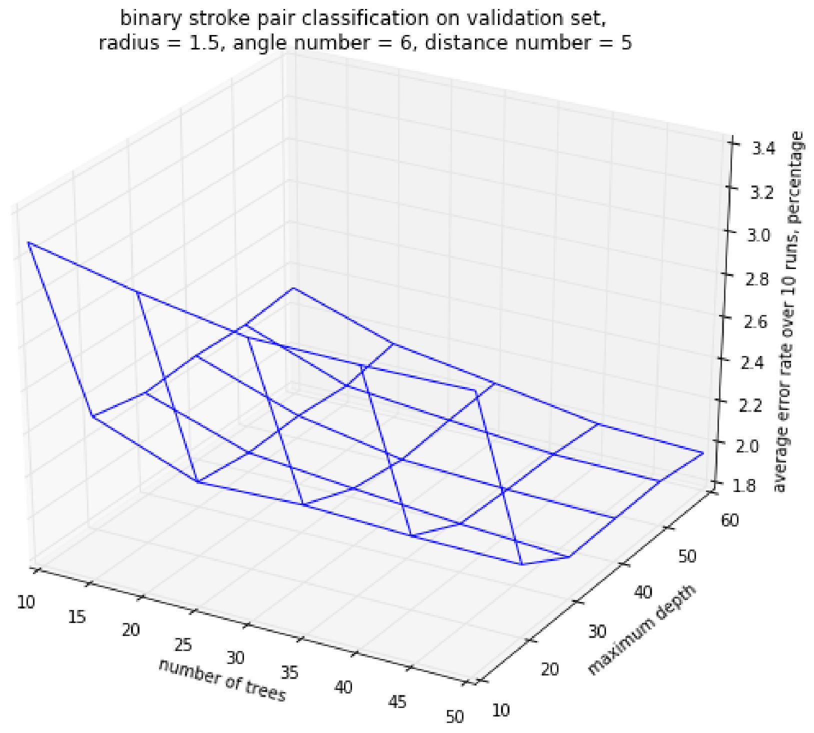


Figure 4.17: Binary stroke pair relation classification. Error rates on validation set with different number of trees and different max depth.

Binary Stroke Pair Relation Classification and Segmentation Experiment Results

Table 4.13: Error rates of binary stroke pair relation classification on line of sight graph for different datasets. Features are Parzen window shape context feature + geometric feature + time gap feature. Angle number = 6, distance number = 5, Radius = 1.5. Classifier is Random Forest.

dataset	training set	testing dataset
CROHME 2012	0%	1.74%
CROHME 2014	0.03%	2.12%

We apply the stroke pair relation classifier to the edges in stroke level line of sight graph. Each edge would get a label: either merge or split. We take each connected component as a symbol candidate.

Table 4.14: Symbol segmentation results on different datasets.

dataset	recall	precision	F-score
CROHME 2012 training	100%	100%	100%
CROHME 2012 testing	94.87%	94.56%	94.72%
CROHME 2014 training	99.98%	99.79%	99.89%
CROHME 2014 testing	92.41%	92.45%	92.43%

Summary

MST Context Feature and Syntax Context Feature can help binary stroke pair relation classification if we can get the correct edge label for the symbol pair and correct symbol syntax label for the symbol candidate in the symbol layout tree. It is good to use a longer radius for shape context to cover more points from other strokes, because the neighbor strokes could help stroke pair relation classification. For different angle number and distance number combinations, the performance difference is very small. For this stroke pair relation classification, 50 trees are enough. Adding more trees cannot improve the performance further.

4.6 Features and Experiments Symbol Pair Relation Classification

Symbol pair relation classification is very similar to stroke pair relation classification. We use Parzen window shape context feature, geometric feature and time gap feature for stroke pair relation classification. There is no time gap feature for stroke pair. Therefore we use symbol level Parzen window shape context feature and geometric feature for symbol pair relation classification. We also consider adding syntax context feature. But it does not help. There is no merge relation for symbol pair. Therefore symbol pair relation classifier classifies the symbol pair relation as one of the seven classes: no-relation (_), Right (R), Subscript (Sub), Superscript (Sup), Above (A), Below (B), Inside (I).

We extend the line of sight graph from stroke level to symbol level. To construct symbol level line of sight graph, we only need to change stroke to symbol in Algorithm 1 in Chapter 3. The symbol pair data are symbol pairs in the symbol level line of sight graph.

Table 4.15 shows stroke pair relation and symbol pair relation distribution in line of sight graph.

Table 4.15: Distribution of the stroke pair relation and symbol pair relation in line of sight graph on CROHME 2012 training dataset.

	expression	_	*	R	Sub	Sup	A	B	I
stroke pair	1221	104855	16906	18622	1596	1806	1178	1595	534
symbol pair	1221	68233	0	10013	953	1105	844	925	418

4.6.1 Symbol Pair Relation Classification with Random Forest

This section is about multiclass symbol pair classification with Random Forest on line of sight graph.

Adding Syntax Context Feature

We use the syntax context feature mentioned in Section 4.4.1. For each symbol, we use its syntax class label (digit, letter, other). All the points are divided into three categories: points from digit, points from letter, points from other. Each syntax category of points produces a syntax Parzen window probability for each bin.

Table 4.16 shows that the minimal error rate on validation set is 1.32% with the original feature (Parzen window shape context feature + geometric feature). Adding Parzen window syntax context feature calculated based on the ground truth syntax class label reduces the min error rate on validation set to 0.77%. The min error rate on validation set is 1.35% with the Parzen window syntax context feature calculated based on the syntax class label produced by the symbol classifier in section 4.3. These experiment results show Parzen window syntax context feature can provide useful information when the syntax label is accurate for symbol pair relation classification.

Table 4.16: Error rates of multiple class symbol pair relation classification on line of sight graph on CROHME 2012 training set and validation dataset. The symbol pair relation classifier is Random Forest. Original feature means Parzen window shape context feature + geometric feature. Syntax context feature uses symbol's ground truth syntax label. Syntax context feature 2 means the syntax label is given by the symbol classifier. Symbol Classifier is Adaboost with SVM in [23].

features	training set	validation dataset
Original feature	0%	1.32%
Original feature + syntax context feature	0%	0.77%
Original feature + syntax context feature 2	0%	1.35%

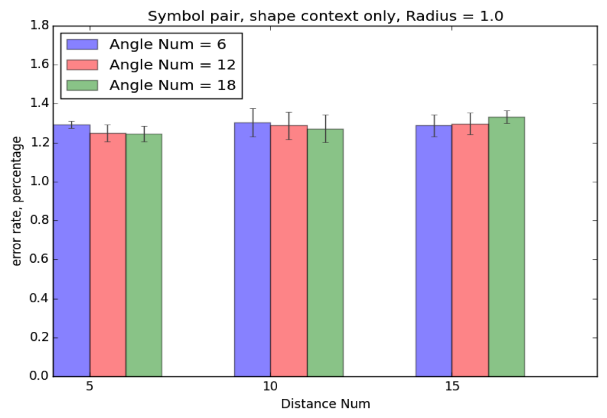
We cannot come to the conclusion that only knowing symbol's syntax label with 100% accuracy could help symbol pair relation classification. But our symbol classifier is not accurate enough to make the syntax context feature improve symbol pair relation classification. We would not use Parzen window syntax context feature for the symbol pair relation classification.

Radius of Shape Context, Angle Number and Distance Number

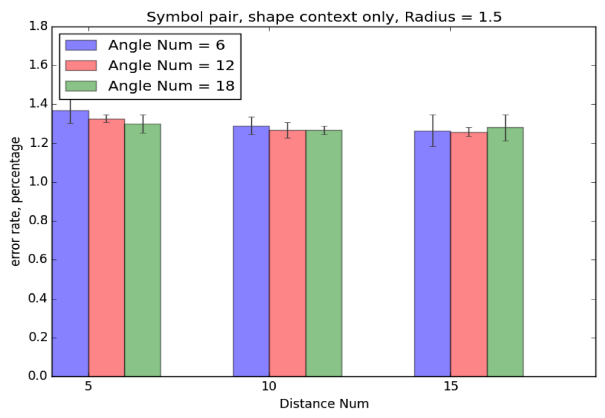
Figure 4.18 shows when the shape context radius is the basic radius unit, Angle number = 6, Distance number = 5, the error rate on the validation set is minimal. But the difference among different (Angle number, Distance number) combinations is very small. Therefore, we use shape context radius = basic radius unit, Angle number = 6, Distance number = 5 for the symbol pair relation classification.

Random Forest Parameter

We run the grid search to find the optimal number of tree and max depth. Figure 4.19 shows that when number of tree = 50, max depth =50, the error rate on validation set is minimal. Therefore, we choose number of trees = 50, and max depth = 50.



(a) Basic radius unit.



(b) 1.5 times of the basic radius unit.

Figure 4.18: Error rates of multiple class symbol pair relation classification on line of sight graph on CROHME 2012 training set and validation set with different radius of shape context, angle number and distance number.

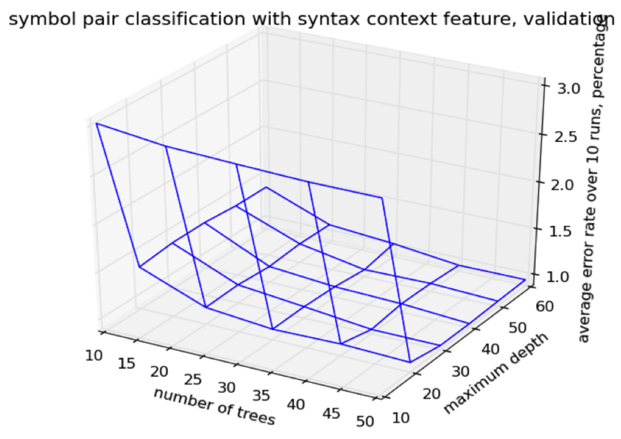


Figure 4.19: Symbol pair relation classification. Error rates on CROHME 2012 validation set with different number of trees and different max depth.

Experiment Results

Table 4.17 shows the multiclass symbol pair relation classification results on line of sight graph for CROHME 2012/2014 datasets. The accuracy on CROHME 2012 test set is 97.64%; while the accuracy on CROHME 2014 test set is 96.55%.

Table 4.17: Error rates of multiple class symbol pair relation classification on line of sight graph for different datasets. Features are Parzen window shape context feature + geometric feature. Classifier is Random Forest.

dataset	training set	testing dataset
CROHME 2012	0.01%	2.36%
CROHME 2014	0.01%	3.45%

4.7 Summary

A feature set which can be calculated based on a simple rule and works well for symbol segmentation, symbol classification and symbol pair relation classification is highly desired. In this section, we propose Parzen window shape context feature for the three tasks.

Stroke pair relation and symbol pair relation classification are very similar. The features used for the two tasks are very similar. We use shape context feature (angle number = 6, distance number = 5, radius = 1.5), geometric feature and time gap for stroke pair relation classification. While there is no time gap between two symbols, we only use shape context feature (angle number = 6, distance number = 5, radius = 1.0) and geometric feature for symbol pair relation classification. As the geometric features are calculated for two objects, we only use shape context feature (angle number = 6, distance number = 5, radius = 1.0) for symbol classification.

We found shape context feature could get very competitive results in symbol segmentation and symbol pair relation classification. It achieves 94.71% symbol segmentation recall on CROHME 2012 test set, and 92.41% on CROHME 2014 test set. Symbol pair relation classification accuracy is around 97% for CROHME 2012 and 2014 test set. The performance of shape context feature on symbol classification is not as good as other features [23]. Therefore, shape context feature can be used as complementary information for different feature sets in the three tasks to improve performance. In addition, our Parzen window shape context feature performs better than original shape context feature for stroke pair relation classification. Using syntax context feature and MST context feature could improve performance for symbol segmentation and symbol pair relation classification if we know the symbol's syntax label and edge label.

As the shape context feature works well for the stroke pair relation classification, symbol classification and symbol pair relation classification of handwritten math, it is interesting to research whether it works well for images of typeset math in the future. It is also interesting to study the effect of imbalance between the 'no relation' and other stroke pair/symbol pair classes on the stroke pair/symbol pair relation classification.

We are going to cover learning parser ensembles for handwritten math in Chapter 5.

Chapter 5

Learning Parser Ensembles for Handwritten Math

Structured learning is the subfield of machine learning and has attracted extensive attention recently in machine learning and computer vision [11, 62].

Math expression is structured data which consists of several parts, and not only the parts themselves (symbol labels) contain information, but also the way in which the parts belong together (spatial relations). The output of math expression recognition system is symbol layout tree. So math expression recognition is a perfect vehicle for studying structured learning. To the best of our knowledge, nobody has used structured learning for math expression recognition before.

We are going to explore whether structured learning is a good direction for handwritten math expression recognition. This chapter is mainly about structured Boosting for stroke-level parser ensemble and holistic recognition for symbol-level parser ensemble.

5.1 Introduction to Structured Learning

Structured data consists of several parts, and not only the parts themselves contain information, but also the way in which the parts belong together. In contrast with conventional supervised learning such as classification, where input data are mapped to atomic labels and regression, where inputs are mapped to scalar numbers or vectors, structured learning is concerned with computer programs that learn to map input data to structured outputs (graphs). Figure 5.1 shows examples of structured output prediction tasks.

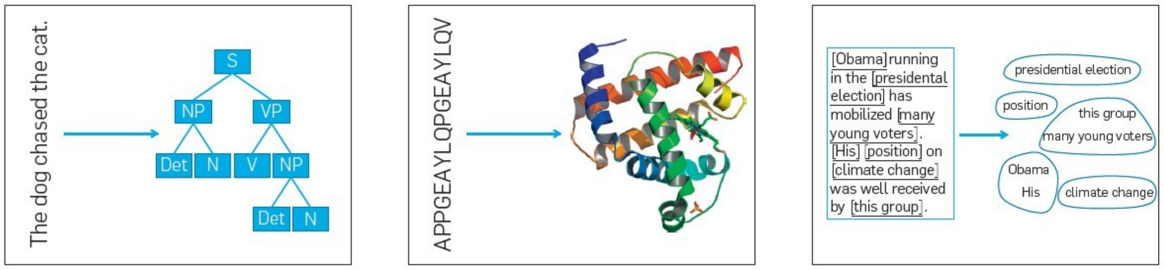

The dog chased the cat.
S
NP
VP
Det
N
V
NP
Det
N
APPGEAYLQPGEAYLQV
[Obama] running in the [presidental election] has mobilized [many young voters]. [His] [position] on [climate change] was well received by [this group].
presidential election
position
this group
many young voters
Obama
His
climate change

Figure 5.1: Examples of structured output prediction tasks: predicting trees in natural language parsing (left), predicting the structure of proteins (middle), and predicting an equivalence relation over noun phrases (right) [35].

In structured learning, there is a prediction function $f : X \to Y$ from an input domain X to a structured output domain Y. The prediction function is defined in such a way that the actual prediction $f(x)$ for a given instance $x \in X$ is obtained by maximizing an auxiliary evaluation function $g : X \times Y \to R$ over all possible elements in y [62], such that

$$y^* = f(x) := \operatorname*{argmax}_{y \in Y} g(x, y). \tag{5.1}$$

This is a generalization of the maximum a posteriori probability (MAP) inference task in graphical models. For example, when the model of interest is the probabilistic model $p(y \mid x)$, we can view the problem of finding the element $y^* \in Y$ that maximizes $p(y \mid x)$ as an instance of Equation 5.1 by defining $g(x, y) = p(y \mid x)$.

5.1.1 An Example of Structured Learning

The foreground-background image segmentation is used as an example to illustrate the idea of structured learning [62]. In foreground background image segmentation we are given a natural image and need to determine for each pixel whether it represents the foreground object or the background. To this end we define one binary output variable $y_i \in \{0, 1\}$ for each pixel i, taking $y_i = 1$ if i belongs to the foreground, $y_i = 0$ otherwise. A single observation variable $x \in X$ will represent the entire observed image. The output is a binary image which has the same size as the input image.

In usual classification, a local model $g_i(y_i, x)$ is formed to classify each pixel independently. If x is a foreground object, $g_i(1, x)$ should be high. While if x is a background object, $g_i(1, x)$ should be low. The idea of structured learning takes the context into account. For a pixel x, if its neighbors are foreground, it is more possible to be foreground. Therefore we introduce an interaction aimed at making locally consistent decisions about the output variables: for each pair (i, j) of pixels that are close to each other in the image plane (say within the 4-neighborhood J, we introduce a pairwise interaction term $g_{i,j}(y_i, y_j)$ that takes a large value if $y_i = y_j$ and a small value otherwise.

We can now pose segmentation as a maximization problem over all possible segmentations on n pixels,

$$y^* = \underset{y \in \{0,1\}^n}{\operatorname{argmax}} [\sum_{i=1}^{n} g_i(y_i, x) + \sum_{(i,j) \in J} g_{i,j}(y_i, y_j)]. \tag{5.2}$$

The auxiliary evaluation function $g(x, y)$ contains two parts: $\sum_{i=1}^{n} g_i(y_i, x)$ and $\sum_{(i,j)\in J} g_{i,j}(y_i, y_j)$. $\sum_{i=1}^{n} g_i(y_i, x)$ is determined by the local classifier, and $\sum_{(i,j)\in J} g_{i,j}(y_i, y_j)$ is determined by the global context and structure information.

The local classifier $g_i(y_i, x)$ can be a simple binary classifier which produce a score for a pixel based on its feature vector. The pairwise interaction $g_{i,j}(y_i, y_j)$ can be represented by a 2×2 table which has a score for $g_{i,j}(0, 0)$, $g_{i,j}(0, 1)$, $g_{i,j}(1, 0)$ and $g_{i,j}(1, 1)$ for all adjacent pixel pair $(i, j) \in J$

The optimal prediction y^* will trade off the quality of the local model g_i with making decisions that are spatially consistent according to $g_{i,j}$. Spatially consistency indicates each pixel tends to have the same label as its neighbors. This is shown in Figure 5.2.

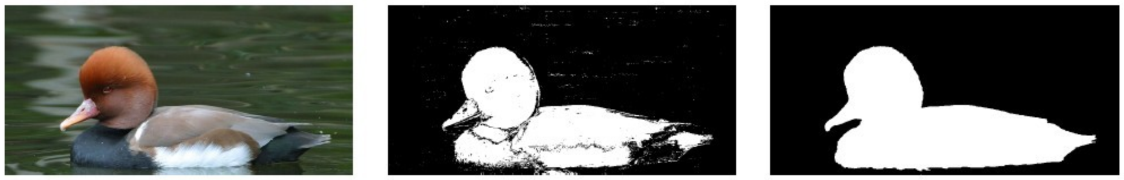


Figure 5.2: Input image (left); pixelwise separate classification by g_i only: image noisy, locally inconsistent decisions (middle); joint optimum y^* with spatially consistent decisions (right) [62].

5.1.2 Conditional Random Field and Structured Support Vector Machine

Conditional random field (CRF) and structured support vector machine (SSVM) are the two state-of-the-art structured learning methods [62]. They both can represent the interaction among output variables very well and incorporate prior heuristics into the structured learning methods easily.

In discriminative models, conditional distribution $p(y \mid x)$ is needed for classification.

(Conditional random field) [38] A conditional random field is an undirected graph H whose nodes correspond to $X \cup Y$; the network is annotated with a set of factors $\phi_1(D_1), \cdots, \phi_m(D_m)$ such that each $D_i \not\supset X$. The network encodes a conditional distribution as follows: $P(Y \mid X) = \frac{1}{Z(X)}\tilde{P}(Y, X)$, $\tilde{P}(Y, X) = \prod_{i=1}^{m} \phi_m(D_m)$, $Z(X) = \sum_Y \tilde{P}(Y, X)$. Figure 3.3 shows a chain-structured case of CRFs for sequences. X is the random variable and it represents the observation sequences. X is not generated by the model. Y is a random variable and it represents the label sequences [43]. For example, X could be a natural language sentence, while Y is part-of-speech tagging of the sentence. In theory the structure of graph can be arbitrary as long as it obeys the conditional independencies in the label sequences being modeled [43]. In Figure 5.3, the structure of Y forms a simple first-order chain.

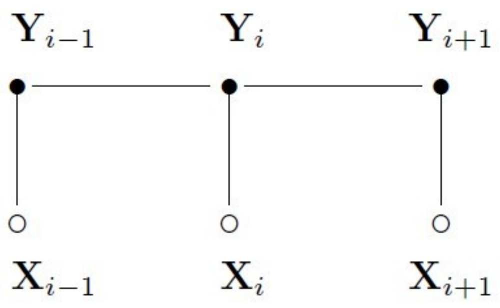


Figure 5.3: Chain-structured case of CRFs [43].

CRFs are essentially a way of combining the advantages of classification and graphical modeling, combining the ability to compactly model multivariate data with the ability to leverage a large number of input features for prediction [73]. The advantage to a conditional model is that dependencies that involve only variables in x play no role in the conditional model, so that an accurate conditional model can have much simpler structure than a joint model. Although early applications of CRFs used linear chains, recent applications of CRFs have also used more general graphical structures. General graphical structures are useful for predicting complex structures, such as graphs and trees, and for relaxing the independent and identically distribution (i.i.d.) assumption among entities.

CRFs provide one method for extending the ideas behind classification to structured prediction. The auxiliary evaluation function $g(x, y)$ can factorize according to a set of local factors, just as in graphical models. For example, the auxiliary evaluation function $g(x, y)$ over labels can be written as a sum of local functions $g(x, y) = \sum_i f_i(y_i, x, w)$. The task is to estimate the real-valued parameter vector w given a training set $D = \{x^i, y^i\}(1 \leq i \leq N)$. The parameter vector w can be used for any kind of model.

In probabilistic graphical model, probabilistic learning aims at identifying a weight vector w that makes $p(y \mid x, w)$ as close to the true conditional label distribution $d(y \mid x)$ as possible. The most common principle for probabilistic training is (regularized) conditional likelihood maximization. training a graphical model in this way is generally called CRF training.

Besides probabilistically trained CRFs, margin-based parameter learning is also widely used in structured prediction. CRF training is probabilistic parameter learning, while margin-based parameter learning is loss-minimizing parameter learning.

(Probabilistic Parameter Learning) [62] Let $d(y \mid x)$ be the (unknown) conditional distribution of labels for a problem to be solved. For a parameterized conditional distribution $p(y \mid x, w)$ with parameters $w \in W^D$, probabilistic parameter learning is the task of finding a point estimate of the parameter w^* that makes $p(y \mid x, w^*)$ closest to $d(y \mid x)$ for every $x \in X$.

(Loss-Minimizing Parameter Learning) [62] Let $d(x, y)$ be the unknown distribution of data in labels, and let $\Delta : Y \times Y \rightarrow$ be a be a loss function. Loss minimizing parameter learning is the task of finding a parameter value w^* such that the expected prediction risk $E_{(x,y)\ d(x,y)}[\Delta(y, f_p(x))]$ is as small as possible, where $f_p(x) = \text{argmax}_{y \in Y}\, p(y \mid x, w^*)$.

Loss-minimizing parameter learning aims at finding a prediction function $f : X \rightarrow Y$ which minimizes the Bayes risk, i.e. the expected Δ-loss $E_{(x,y)\ d(x,y)}\Delta(y, f(x))$. As in structured prediction, f is assumed to have the form $f(x) = \underset{y}{\text{argmax}}\, g(x, y, w)$ for an auxiliary evaluation function $g : X \times Y \rightarrow R$, which is parameterized by $w \in R^D$. Because $d(x, y)$ is unknown, minimizing the Bayes risk directly is not possible, but structural risk minimization [79] offers an indirect way to identify a function with good predictive qualities. It chooses a prediction function f that minimizes the regularized empirical risk functional

$$R(f) + \frac{C}{N} \sum_{n=1}^{N} \Delta(y_n, f(x_n)), \tag{5.3}$$

where the second term is an empirical estimate of the expected risk, and the first term is chosen as a regularizer that prevents overfitting by penalizing functions depending on how complex they are [62]. $\Delta(y^n, f(x^n))$ is piece-wise constant. This means the gradient is always 0 and gradient-based methods do not work here. Therefore minimizing the Equation 5.3 with respect to w is impracticable for structured prediction functions $f(x) = \underset{y}{\operatorname{argmax}}\, g(x, y, w)$. However, results from statistical learning theory show that it can be sufficient to minimize a convex upper bound to Equation 5.3 and still achieve an optimal prediction accuracy in the limit of infinite data. This forms the basis of SSVM training.

(Structured Support Vector Machine) [62] Let $g(x, y, w) =< w, \varphi(x, y) >$ be a compatibility function parameterized by $w \in R^D$. For any $C > 0$, structured support vector machine (SSVM) training chooses the parameter

$$w^* = \underset{w \in R^n}{\operatorname{argmin}} \frac{1}{2} \mid\mid w \mid\mid^2 + \frac{C}{N} \sum_{n=1}^{N} l(x^n, y^n, w), \tag{5.4}$$

with

$$l(x^n, y^n, w) = \max_{y \in Y} \Delta(y^n, y) - g(x^n, y^n, w) + g(x^n, y, w). \tag{5.5}$$

Equation 5.4 is derived from Equation 5.3 by choosing $R(f) = \frac{1}{2} \mid\mid w \mid\mid^2$ as a regularizer and replacing the Δ-loss by its convex upper bound l.

For an intended output $t = \pm 1$ and a classifier score y (y is a raw output of SVM), the hinge loss of y is defined as $l(y) = max(0, 1 - t \cdot y)$. Equation 5.5 generalizes the Hinge loss to multiple outputs labels. As a result, Equation 5.4 can be interpreted as a maximum margin training procedure that extends the popular support vector machine classifiers to structured output spaces.

CRF training converts global training for structured prediction into a convex optimization problem. SSVM has the similar idea but employs different loss functions and optimization methods. As for CRFs, training SSVM is a computationally expensive task, and it often only becomes feasible by a careful analysis of the problem components and the exploitation of domain knowledge [62]. The differences between the various structured prediction methods are not well understood. To date, there has been little careful comparison of these, especially CRFs and max-margin approaches (SSVM), across different structures and domains [73]. The similarities between various structured prediction methods are more important than the differences [73]. Careful selection of features has more effect on performance than the choice of structured prediction algorithm.

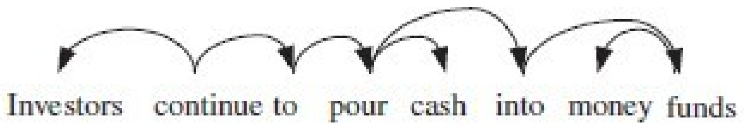
Conditional random field (CRF) and structured support vector machine (SSVM) are the two state-of-the-art structured learning methods [62]. They both can represent the interaction among output variables well and integrate prior heuristics given a problem of interest. CRF training converts global training for structured prediction into a convex optimization problem. SSVM has the similar idea but employs different loss functions and optimization methods. Both CRF and SSVM are complex to implement, expensive to run and based on specific training algorithm.

5.1.3 Structured Boosting

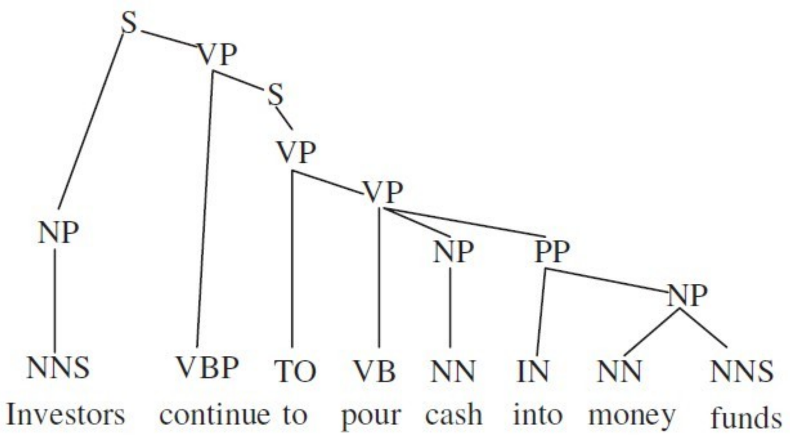
Wang et. al [81, 82] propose a structured Boosting to train a dependency parser for natural language processing. Dependency syntax focuses on relationships among words. For the input sentence, the dependency parser outputs a dependency tree, as showed in Figure 5.4 (a). A dependency tree indicates which words in a sentence are directly related. A dependency tree is much easier to understand and annotate than constituency tree (as showed in Figure 5.4 (b)) which contains speech and phrase labels.

The idea of the structured Boosting [81, 82] is to train the local classifier on the local labeled samples. Then the global structured classifier is used to classify the data by using the classification results of the current weak local classifier. The ensemble weight for the current weak local classifier is calculated and the local sample weights are updated based on the classification results of global structured classifier according to any boosting algorithm.

The structured Boosting is used to augment the training of local classifier and does not change the training of the local classifier. The idea is to train the local classifier on the local labeled samples. Then



(a) dependency tree

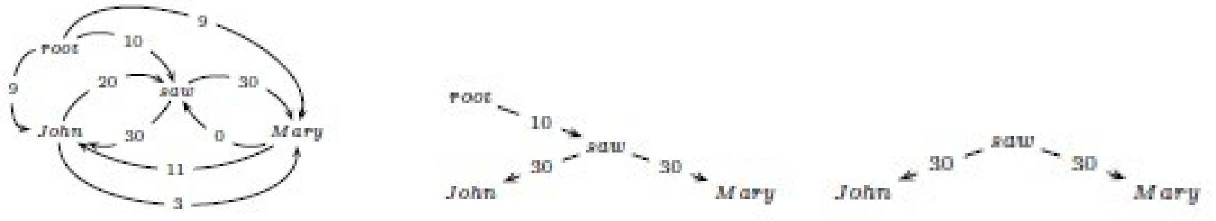


(b) constituency tree

Figure 5.4: Dependency tree and constituency tree [82].

the classifier with global context is used to classify the data by using the output of the current weak local classifier. The ensemble weight for the current weak local classifier is calculated and the local sample weights are updated based on the classification results of global structured classifier.

Wang et. al [82] uses a general framework for parsing dependency trees which is similar to [54]. The dependency parsing algorithm is a dynamic programming algorithm which aims to produce a directed maximum weight spanning tree subject to constraints in directed graphs. Figure 5.5 shows an example of parsing the sentence John saw Mary. For each sentence, a corresponding directed graph is constructed. There is a directed edge between every pair of distinct words and from the dummy root symbol to every word. Therefore the dummy node is guaranteed to be the root node of the MST and there is only one edge going out from the dummy node in the MST. In the MST, the child node of the dummy node in the MST is the root node of the dependency tree. Adding the dummy node can help people not have to choose the root node for the dependency tree.



(a) full directed graph (b) MST with the dummy node (c) MST without the dummy node

Figure 5.5: An example of getting dependency tree by using MST. The sentence is John saw Mary.

The weak learner used in [82] is logistic regression. Here the logistic regression is a binary classifier. It classifies the relation between a word pair to be left or right. Because the relationship between a word pair can only be left, right or no relation and there is a relation (left or right) or no relation is determined by the MST produced by the Edmonds' algorithm.

For each iteration of the boosting, a logistic regression classifier is trained. For each edge in the directed graph, the logistic regression classifier can calculate the probability for the two possible classes (left, right). Chu-Liu-Edmonds algorithm is used to get the MST in the directed graph. The output of the local classifier (base learner) is used as a numerical weight to score a potential link. The classification is determined by the MST but not the base learner. If there is no edge for a word pair in the MST, then the class for the word pair relationship is no relation, no matter the base learner (logistic regression) classifies the word pair relationship to be left or right. If there is an edge for the word pair, then the class for the word pair relationship is the label associated with the edge produced by the base learner. In this sense, the role of a local classifier is to supply a learned scoring function to a pre-existing algorithm. The weights of each mis-parsed local example were simply increased by an additive constant, with other weights kept the same. Instead of using all the weak learner at each iteration, only the last hypothesis is kept. The parser used in the paper is standard CYK parsing [88].

Wang et. al [82] mention a distance feature which measures how far apart the two words are in a sentence is highly predictive of link existence. The dynamic features are similar to the time gap feature or bigram or trigram statistics used for math symbol relationship classification. The larger the dynamic feature is, the more possible there is no relationship between the word pair.

For Structured Boosting, any classification technique which can produce a score for the classification result can be used as base learner. Structured Boosting does not change the training of the base learner, but the sample weight is updated based on structured classification result but not the base learner.

5.1.4 Application to Parsing Handwritten Math

Math expression is structured data which consists of several parts, and not only the parts themselves (symbol labels) contain information, but also the way in which the parts belong together (spatial relations). In contrast with conventional supervised learning such as classification, where input data (instances) are mapped to atomic labels and regression, where inputs are mapped to scalar numbers or vectors of scalars, structured learning is concerned with computer programs that learn to map input data to structured outputs (graphs) [11, 62]. The output of math expression recognition system is symbol layout tree. So math expression recognition is a perfect vehicle for studying structured learning. To the best of our knowledge, nobody has used structured learning for math expression recognition before.

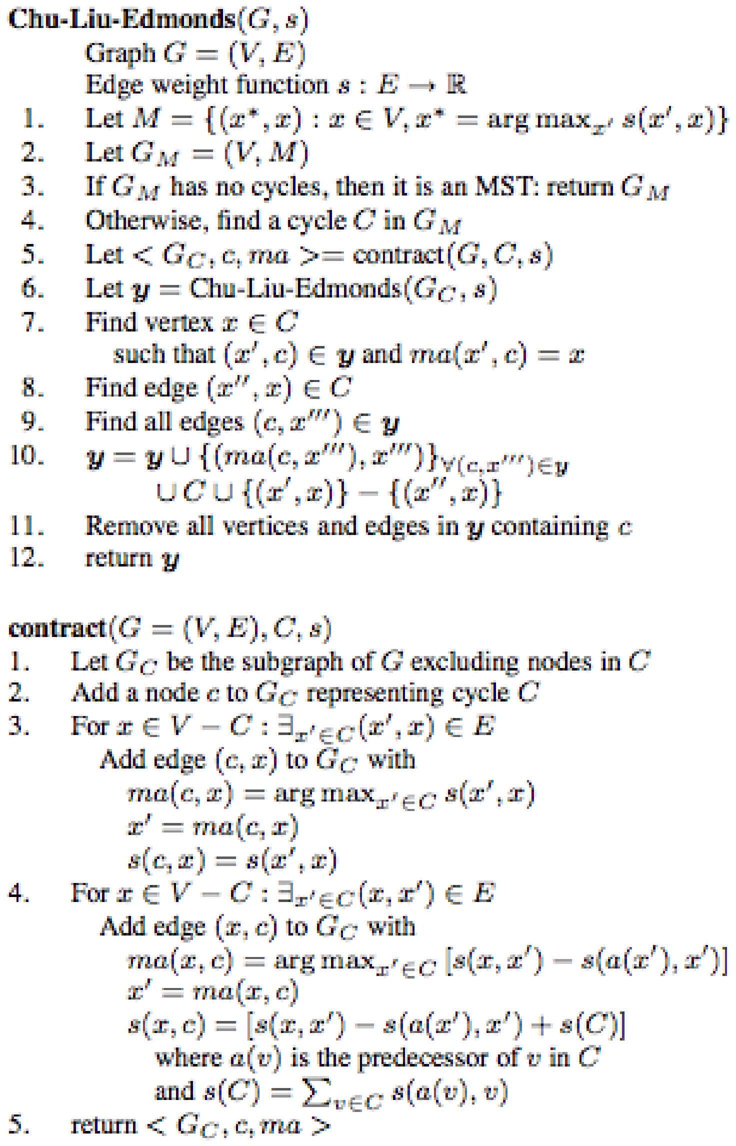
As structured Boosting achieves success in parsing natural language [81, 82], and parsing math expression is similar to parsing natural language. We adopt structured Boosting as structure learning schema for handwritten math expression recognition.

5.2 Edmonds' Algorithm based Parser

We use Edmonds' algorithm to parse the math expression. This section is going to cover Edmonds' algorithm and how we apply it to parse math expression at stroke level and symbol level.

5.2.1 Edmonds' Algorithm

Edmonds' algorithm or Chu-Liu/Edmonds' algorithm [22,24] is an algorithm for finding minimum/maximum spanning tree (MST) in directed graph. Figure 5.6 shows the Pseudocode of Edmonds' Algorithm. The input is a directed graph $G = (V, E)$, where V is a set of nodes and E is a set of directed edges. Each edge has a real-valued weight.



```
**Chu-Liu-Edmonds**$(G, s)$
        Graph $G = (V, E)$
        Edge weight function $s : E \rightarrow \mathbb{R}$
 1.     Let $M = \{(x^*, x) : x \in V, x^* = \arg\max_{x'} s(x', x)\}$
 2.     Let $G_M = (V, M)$
 3.     If $G_M$ has no cycles, then it is an MST: return $G_M$
 4.     Otherwise, find a cycle $C$ in $G_M$
 5.     Let $< G_C, c, ma >=$ contract$(G, C, s)$
 6.     Let $\boldsymbol{y} =$ Chu-Liu-Edmonds$(G_C, s)$
 7.     Find vertex $x \in C$
           such that $(x', c) \in \boldsymbol{y}$ and $ma(x', c) = x$
 8.     Find edge $(x'', x) \in C$
 9.     Find all edges $(c, x''') \in \boldsymbol{y}$
10.     $\boldsymbol{y} = \boldsymbol{y} \cup \{(ma(c, x'''), x''')\}_{\forall (c, x''') \in \boldsymbol{y}}$
              $\cup\, C \cup \{(x', x)\} - \{(x'', x)\}$
11.     Remove all vertices and edges in $\boldsymbol{y}$ containing $c$
12.     return $\boldsymbol{y}$

**contract**$(G = (V, E), C, s)$
1.    Let $G_C$ be the subgraph of $G$ excluding nodes in $C$
2.    Add a node $c$ to $G_C$ representing cycle $C$
3.    For $x \in V - C : \exists_{x' \in C}(x', x) \in E$
         Add edge $(c, x)$ to $G_C$ with
            $ma(c, x) = \arg\max_{x' \in C} s(x', x)$
            $x' = ma(c, x)$
            $s(c, x) = s(x', x)$
4.    For $x \in V - C : \exists_{x' \in C}(x, x') \in E$
         Add edge $(x, c)$ to $G_C$ with
            $ma(x, c) = \arg\max_{x' \in C} [s(x, x') - s(a(x'), x')]$
            $x' = ma(x, c)$
            $s(x, c) = [s(x, x') - s(a(x'), x') + s(C)]$
               where $a(v)$ is the predecessor of $v$ in $C$
               and $s(C) = \sum_{v \in C} s(a(v), v)$
5.    return $< G_C, c, ma >$
```

Figure 5.6: Pseudocode of Edmonds' Algorithm. [19]

The algorithm greedily selects the incoming edge with the highest weight for each node. If there is no cycle, then all the selected edges form an MST. If there is a cycle, the algorithm would contract the cycle into a single node and recalculate edge weights going into and out the cycle. The running time of Edmonds' algorithm is $O(|E| \times |V|)$.

5.2.2 Stroke-Level Parser

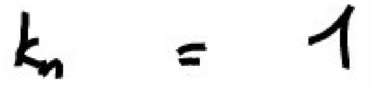
Figure 5.7 shows an example of stroke-level parser based on multiclass stroke pair relation classifier and Edmonds' algorithm. The input expression is $k_n = 1$ and it contains 6 strokes. Symbol k has two strokes $|$ and $<$. Symbol $=$ consists of two $-$. Other symbols have one stroke. Stroke level line of sight graph is constructed then, and we add a dummy node to the line of sight graph. There is an edge from dummy node to each stroke node, but the dummy node does not have incoming edge. For each edge (stroke pair) in the line of sight graph, the stroke pair relation classifier produces a score S_i for each of the classes C_i. Each edge has a score-class list $(S_1, C_1), \cdots (S_M, C_M)$ and M is the number of stroke pair relationship classes. Edmonds' algorithm produces the MST from the directed graph (as showed in Figure 5.7 (b)) and the MST at stroke level would be converted to a symbol layout tree (as showed in Figure 5.7 (e)).

Comparison Between Multiclass Stroke Pair Relation Classifier and Stroke-Level Parser

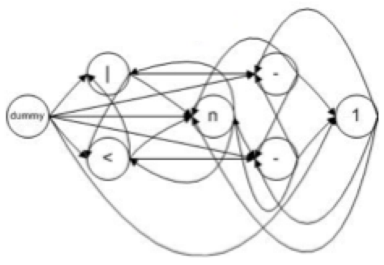
We compare multiclass stroke pair relation classifier and multiclass stroke pair relation classifier + Edmonds' algorithm to check whether Edmond' algorithm could improve stroke pair relation classification accuracy.

We train a multiclass stroke pair relation classifier (AdaBoost.MH). For each edge in the stroke level line of sight graph, the AdaBoost.MH classify its label to be the eight possible classes (merge, no-relation, right, subscript, superscript, below, above, inside).

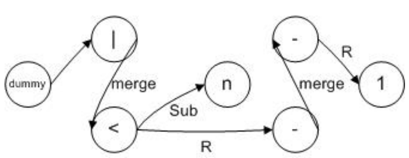
In contrast, we do not use the label produced by the AdaBoost.MH classifier. The AdaBoost.MH produces a score list for each edge. Edmonds' algorithm extracts the stroke level MST from the line of sight



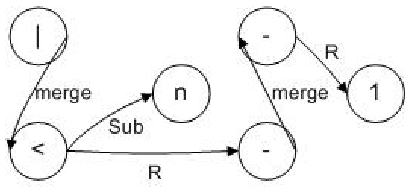
(a) Input expression



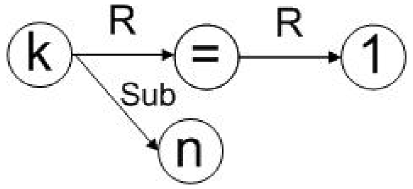
(b) Stroke level line of sight graph, each edge has a class-score list $(S_1, C_1) \cdots (S_M, C_M)$



(c) Apply Edmonds' algorithm to get MST with dummy node



(d) Remove the dummy node



(e) Recover symbol layout tree from stroke level MST

Figure 5.7: Stroke-level parser based on multiclass stroke pair relation classifier and Edmonds' algorithm. Score-class list $(S_1, C_1), \cdots (S_M, C_M)$ is produced by multiclass stroke pair relation classifier for each stroke pair.

graph based on the score list. The MST is converted into the symbol layout tree. The label of each stroke pair is determined by the symbol layout tree but not stroke pair relation classifier (AdaBoost.MH).

Table 5.1 shows that stroke-level parser has a lower error rate than AdaBoost.MH for different angle number and distance number combinations. It supports stroke level parser could improve the stroke pair relation classification accuracy.

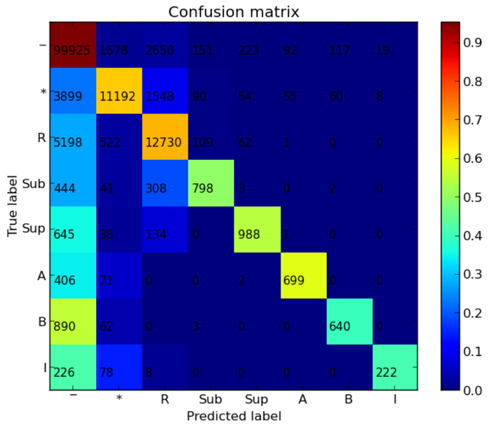
Figure 5.8 shows the confusion matrixes for AdaBoost.MH and stroke-level parser when angle number is 18 and distance number is 15. For no-relation (_), stroke-level parser performs worse than AdaBoost.MH. But it performs better for all the other relations. For Below (B) and Inside (I) relations, stroke-level parser can reduce almost 50% of the errors.

The reason is because the distribution of stroke pair label is very unbalanced in stroke level line of sight graph. More than 71% of training stroke pair samples have no-relation (_) as the label. The multiclass stroke

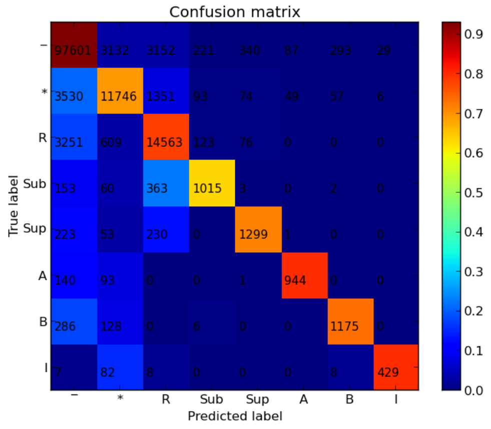
Table 5.1: Error rate comparison between AdaBoost.MH (classifier 1) and stroke-level parser (classifier 2) with different angle number (M) and distance number (N)

M, N	classifier 1 (training)	classifier 2 (training)	classifier 1 (validation)	classifier 2 (validation)
6, 5	13.97%	13.38%	15.37%	16.40%
6, 10	13.83%	12.95%	15.14%	14.08%
6, 15	14.00%	12.42%	15.03%	13.52%
12, 5	13.98%	12.25%	15.35%	14.13%
12, 10	13.74%	12.32%	15.12%	14.58%
12, 15	13.68%	12.06%	15.50%	16.28%
18, 5	13.71%	12.04%	14.65%	16.16%
18, 10	13.65%	12.34%	15.21%	14.80%
18, 15	13.53%	12.45%	14.68%	14.86%

pair relation classifier (AdaBoost.MH) tends to classify other labels as no-relation (_). But the stroke-level parser produces a stroke level MST. All the edges in the stroke level MST has a label instead of no-relation (_). This makes stroke-level parser classify less stroke pair as no-relation (_) than the multiclass stroke pair relation classifier (AdaBoost.MH).



(a) AdaBoost.MH

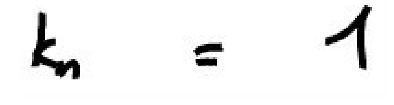


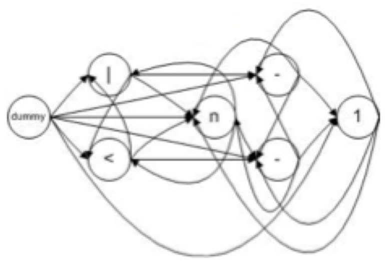
(b) stroke-level parser

Figure 5.8: Confusion matrixes comparison.

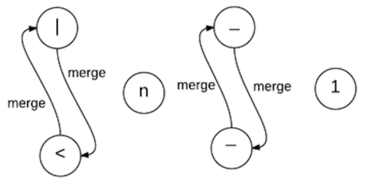
5.2.3 Symbol-Level Parser

Figure 5.9 shows an example of symbol-level parser based on binary stroke pair relation classifier, multiclass symbol pair relation classifier and Edmonds' algorithm. Edmonds' After constructing the stroke level line of sight graph, the binary stroke pair relation classifier would classify each edge as merge or split (as showed in Figure 5.9 (c)). Each connected component is taken as a symbol candidate, and symbol level line of sight graph is constructed based on those symbol candidates (as showed in Figure 5.9 (d)). For each edge (symbol pair) in the symbol level line of sight graph, the multiclass symbol pair relation classifier produces a score list (a score S_i for each of the classes C_i). After Edmonds' algorithm extracting the symbol level maximum spanning tree form the LOS graph (as showed in Figure 5.9 (e)), the MST without the dummy node is taken as the recognition result.

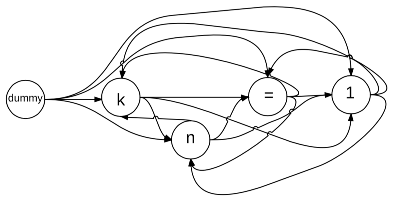

(a) Input expression



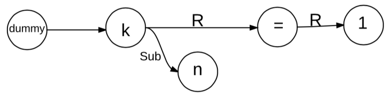
(b) Stroke level line of sight graph



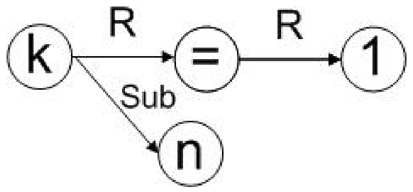
(c) Apply binary stroke pair relation classifier to each edge. Each connected component is taken as a symbol candidate and classified



(d) Symbol level line of sight graph, each edge has a class-score list $(S_1, C_1) \cdots (S_M, C_M)$



(e) Apply Edmonds' algorithm to get MST with dummy node



(f) Remove dummy node to get symbol layout tree

Figure 5.9: Stroke-level parser based on multiclass stroke pair relation classifier and Edmonds' algorithm. Score-class list $(S_1, C_1), \cdots (S_M, C_M)$ is produced by multiclass stroke pair relation classifier for each stroke pair.

5.3 Parser ensembles

5.3.1 Stroke-Level Parser Ensemble

We use Structured Boosting to get stroke level parser ensemble. Figure 5.10 shows the training of stroke-level parser ensemble with structured boosting. During the training, a multiclass stroke pair relation classifier (such as AdaBoost.MH or random forest) is trained at each iteration of Boosting. For each edge in the line of sight graph at stroke level, the stroke pair relation classifier can calculate the probability for the eight possible classes (merge, no-relation, right, subscript, superscript, below, above, inside). Edmonds' algorithm is used to get the stroke level MST in the directed graph. The MST is converted into the symbol layout tree. The classification is determined by the symbol layout tree but not the multiclass stroke pair relation classifier. If there is no edge for a stroke pair in the symbol layout tree, then the class for the stroke pair relationship is no relation, no matter the multiclass stroke pair relation classifier (such as AdaBoost.MH or random forest) classifies the stroke pair relationship to be one of seven relations (merge, right, subscript, superscript, below, above, inside). If there is an edge for the stroke pair in the symbol layout tree, then the class for the stroke pair relationship is the label associated with the edge selected by the Edmonds' algorithm. The ensemble weight for the current base learner (such as AdaBoost.MH or random forest) is calculated and the weight of each sample (stroke pair) is updated based on its label in the symbol layout tree. The role of a base learner is to supply a learned scoring function to Edmonds' algorithm.

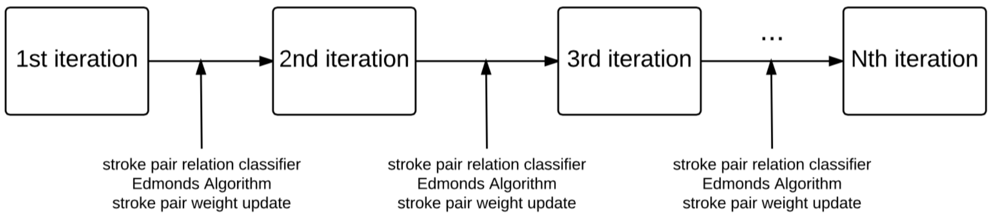

1st iteration
2nd iteration
3rd iteration
...
Nth iteration
stroke pair relation classifier
Edmonds Algorithm
stroke pair weight update
stroke pair relation classifier
Edmonds Algorithm
stroke pair weight update
stroke pair relation classifier
Edmonds Algorithm
stroke pair weight update

Figure 5.10: training of Structured Boosting for Handwritten Math Expression Recognition

After the training is finished, for a given math expression, we would use the stroke pair relation classifier trained at each iteration to produce a score list for each edge, and sum over all the score list to get the final score list. Then Edmonds' algorithm is applied to the line of sight graph with summed scores to get the MST at stroke level. The MST is converted into the symbol layout tree and the symbol layout tree is the recognition result for the given expression.

We use AdaBoost.M1 as the boosting schema for structured Boosting. The base learner (multiclass stroke pair relation classifier) of AdaBoost.M1 is AdaBoost.MH or Random Forest.

AdaBoost.M1 is a multiclass extension of AdaBoost (AdaBoost deals with binary classification) [67]. Algorithm 5 describes AdaBoost.M1. At each iteration of AdaBoost.M1, it aims to minimize the summed weighted error ϵ_t (at line 6 of Algorithm 5) by training the weak learner with the weighted samples. AdaBoost.M1 updates the weight of each sample based on the classification result (line 12 of Algorithm 5). If the sample is classified correctly, the weight goes down and vice versa. The final classification is the majority vote of all the base learners from each iteration.

Algorithm 6 shows the pseudo code of stroke-level parser ensemble . At the 1st iteration of AdaBoost.M1, the initial weight for each sample (stroke pair) is $\frac{1}{N}$, while N is the number of training sample. The multiclass pair relation classifier (AdaBoost.MH or Random Forest) will be trained based on the initial sample weight. Once the training of stroke pair relation classifier is finished, the stroke pair relation classifier can calculate the score list for each edge. There is one score in the score list for each possible relationship. Edmonds' algorithm is used to get the MST in the directed graph. The MST is converted into the symbol layout tree. The classification is determined by the symbol layout tree but not the base learner (multiclass stroke pair relation classifier). In the Algorithm 6, the multiclass stroke pair relation does not give the label for the given stroke pair, but just provide a score list for Edmonds' algorithm. The weight of each sample is updated based on the classification result.

Algorithm 5 AdaBoost.M1: multiclass extension of AdaBoost [67]

```
1: Given: $(x_1, y_1), \cdots, (x_m, y_m)$ where $x_i \in X, y_i \in Y$
2: Initialize: $D_1(i) = 1/m$ for $i = 1, \cdots, m$.
3: for $t = 1, \cdots T$: do
4:     Train weak learner using distribution $D_t$.
5:     Get weak hypothesis $h_t : X \to Y$.
6:     Aim: select $h_t$ to minimalize the weighted error: $\epsilon_t \doteq Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
7:     if $\epsilon_t \geq \frac{1}{2}$ then
8:         set $T = t - 1$ and exit loop.
9:     end if
10:    Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$.
11:    Update, for $i = 1, \cdots, m$:
12:        $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$
13:    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution).
14: end for
15: Output the final hypothesis:
16:    $$H(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} \alpha_t \{h_t(x) = y\}.$$
```

Then, at the 2nd iteration of AdaBoostM1, the sample weight is the updated sample weight. Multiclass stroke pair relation classifier (AdaBoost.MH or Random Forest) will be trained based on the updated sample weight. The rest of 2nd iteration is the same as the 1st iteration.

The rest of each iteration is the same as the 2nd iteration.

Random Forest as Multiclass Stroke Pair Relation Classifier

When multiclass stroke pair relation classifier is Random Forest, the training of stroke-level parser ensemble terminates after eight iterations. The stroke pair feature we use here is parzen window shape context feature (angle number = 18, distance number = 15), geometric feature and the time gap feature.

Table 5.2 shows the stroke pair error rates of stroke-level parser ensemble when multiclass stroke pair relation classifier is Random Forest; while Table 5.3 shows the correct expression rates.

Algorithm 6 Stroke-level parser ensemble. Boosting schema in structured Boosting is AdaBoost.M1 [67].

```
1: Given: $(x_1, y_1), \cdots, (x_m, y_m)$ where $x_i \in X, y_i \in Y$
2: Initialize: $D_1(i) = 1/m$ for $i = 1, \cdots, m$.
3: for $t = 1, \cdots T$: do
4:     Train weak learner (multiclass stroke pair relation classifier) using distribution $D_t$.
5:     Get weak hypothesis $h_t : X \rightarrow Y$.
6:     Aim: select $h_t$ to minimalize the weighted error: $\epsilon_t \doteq Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.
7:     if $\epsilon_t \geq \frac{1}{2}$ then
8:         set $T = t - 1$ and exit loop.
9:     end if
10:    Choose $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$.
11:    Update, for $i = 1, \cdots, m$:
12:
       $$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } h_t(x_i) = y_i \\ e^{\alpha_t} & \text{if } h_t(x_i) \neq y_i \end{cases}$$
13:    where $Z_t$ is a normalization factor (chosen so that $D_{t+1}$ will be a distribution);
14:    $h_t(x_i)$ is produced by multiclass stroke pair relation classifier and Edmonds' algorithm.
15: end for
16: Output the final hypothesis:
17:
       $$H(x) = \underset{y \in Y}{\text{argmax}} \sum_{t=1}^{T} \alpha_t \{h_t(x) = y\}.$$
```

Table 5.2: Stroke pair error rates of stroke-level parser ensemble when multiclass stroke pair relation classifier is Random Forest.

iteration	training set	validation set
1	1.58%	4.34%
2	1.38%	4.32%
3	1.33%	4.44%
4	1.33%	4.54%
5	1.33%	4.69%
6	1.30%	4.70%
7	1.31%	4.80%
8	1.30%	4.89%

Table 5.3: Correct expression rate of stroke-level parser ensemble when multiclass stroke pair relation classifier is Random Forest. Correct Expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

iteration	training set	validation set
1	67.32%	36.75%
2	72.56%	34.19%
3	73.63%	35.04%
4	74.61%	32.48%
5	74.28%	30.77%
6	75.27%	33.33%
7	75.02%	32.48%
8	74.86%	31.62%

Summary

For stroke-level parser ensemble, we find performance could be improved as the iteration increases sometimes during the Structured Boosting. But the final performance of Structured Boosting is worse than the performance without Structured Boosting. The reason could be that math expression recognition is more complex than the natural language processing (NLP). There are 8 relations for symbol pair in math recognition while there are only 2 relations for word pair in NLP. For math expression, the stroke-level parser ensemble needs to deal with both segmentation and parsing; while structured Boosting only needs to solve parsing in NLP as the words to parse are already given.

5.3.2 Symbol-Level Parser Ensemble

Motivated by the high accuracy of segmenting symbols using binary stroke pair relation classifier, and improved symbol pair relation classification accuracy from using random forests, we tried using a random forest ensemble of symbol-level parsers, and we call the symbol-level parser ensemble parselets. Each parselet is a symbol-level parser (as showed in Figure 5.9). It comprises of two decision trees that (1) segment symbols (as showed in Figure 5.9 (c)), and then (2) classify symbol pair relation (as showed in Figure 5.9 (d)). Each parselet produces a symbol layout tree independently for a given expression. There is no symbol classifier in the parselet, therefore we use the symbol's ground truth label as the label of node in the symbol layout tree.

Figure 5.11 shows the training of parselets. In training, different parselets are trained independently. For each parselet, a binary stroke pair classifier (merge or split) is trained for all the stroke pairs in the stroke level line of sight graph. The binary stroke pair classifier is a decision tree. Each connected component is taken as a symbol candidate, and symbol level line of sight graph is constructed. The symbol pairs in the symbol level line of sight graph are taken as training sample for the multiclass symbol pair relation classifier which is also a decision tree. We extract all the stroke pairs in the symbol pair and get the ground truth labels for the stroke pairs. The label of the symbol pair is determined by the majority vote of these stroke pair labels. The tie is broken randomly.

In testing, for a given expression, we would get the stroke level line of sight graph first. Then for each stroke pair, the binary stroke pair relation classifier of each parselet produces a merge score and split score. We sum over all the merge score and split score to get the final scores. Stroke pair whose merge score is higher than split score would be merged. After each connected is taken as a symbol candidate we construct the symbol level LOS graph, and the score list of symbol pair is the sum of score list produced by the multiclass symbol pair relation classifier of each parselet. Finally, we apply Edmonds' algorithm to the symbol level LOS graph to get the symbol layout tree.

Figure 5.12 shows the experiment results with different number of parselets. Segmentation means segmenting symbols in Figure 5.12. Segmentation recall is the ratio of number of detected correct symbols

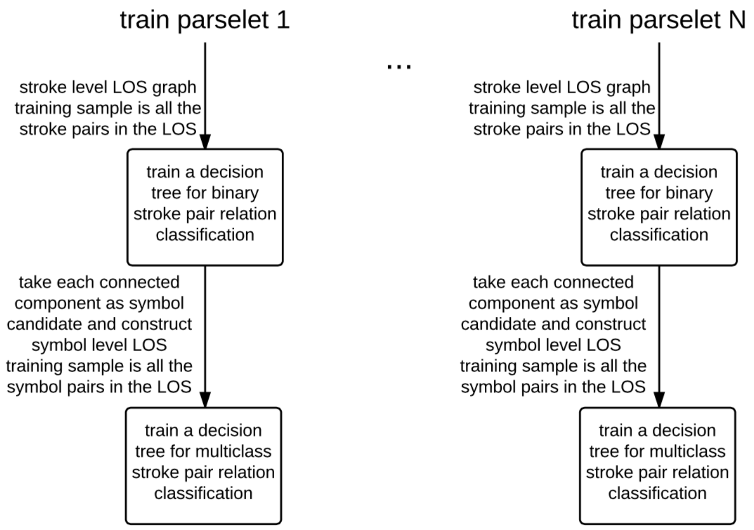

train parselet 1
...
train parselet N
stroke level LOS graph training sample is all the stroke pairs in the LOS
train a decision tree for binary stroke pair relation classification
take each connected component as symbol candidate and construct symbol level LOS training sample is all the symbol pairs in the LOS
train a decision tree for multiclass stroke pair relation classification

Figure 5.11: Training of parselets (symbol level parser ensemble). In testing, the stroke pair classification results and symbol pair classification results produced by each parselet are combined to do the symbol segmentation and layout analysis.

divided by the total number of symbols in the ground truth symbol layout tree. It is easy to find that performance gets better when the number of parselets increases at the beginning. When the number of parselets is larger than 80, the performance does not change much. We did experiment to increase the number of parselet to 200, but the structure rate and expression rate stay the same after 95 parselets.

5.4 Summary

The stroke-level parser ensemble performs worse than a single stroke-level parser. Structured Boosting combines stroke-level parsers sequentially. The training of latter stroke level parser depends on the training of its previous stroke level parser. The reason Structured Boosting works for getting dependency tree for a sentence but not extracting symbol layout tree for a math expression is math expression recognition is more complex than the natural language processing (NLP). There are eight relations for symbol pair in math recognition while there are only two relations for word pair in NLP. For math expression, the stroke-level

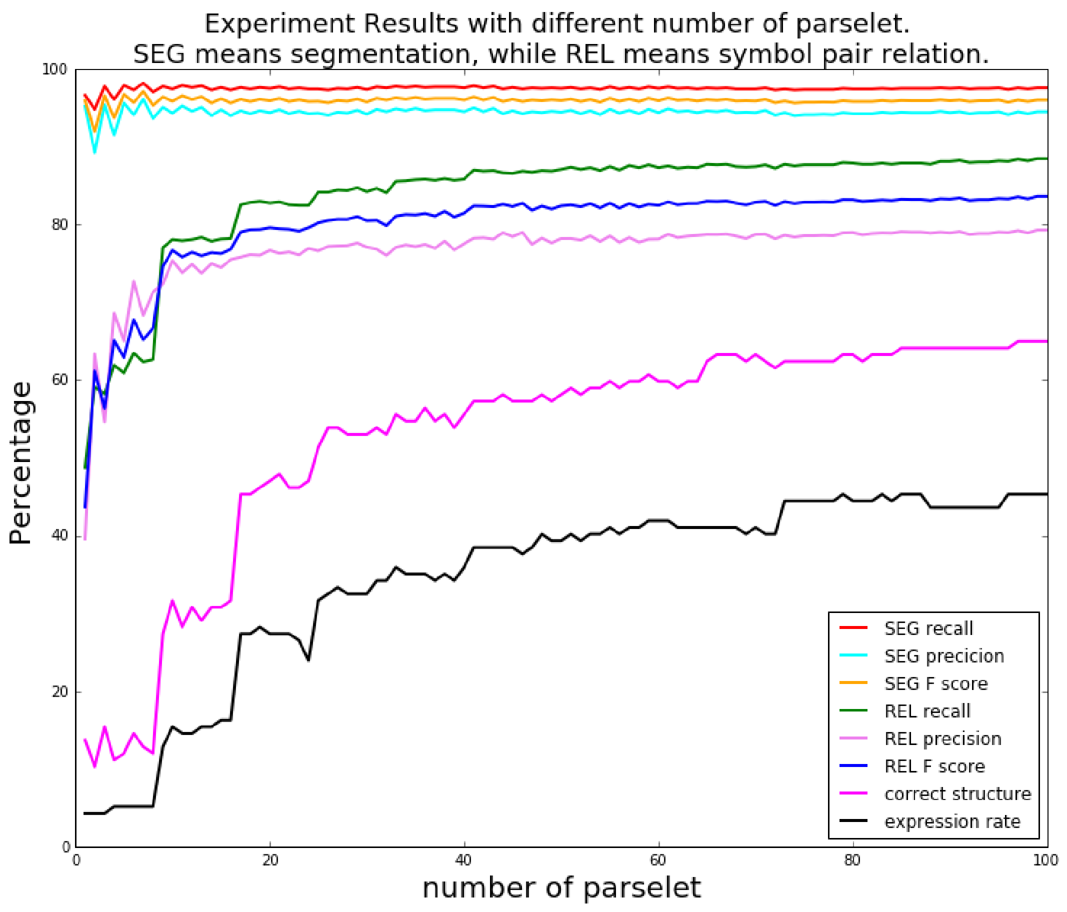


Figure 5.12: Experiment results with different number of parselet on CROHME 2012 validation set. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

parser ensemble needs to deal with both segmentation and parsing; while structured Boosting only needs to solve parsing in NLP as the words to parse are already given. Using the tree to update the sample weight might not work for multiclass classification.

More than 70% stroke pair in the stroke level line of sight has no-relation (_) as its label. Training a multiclass stroke pair relation classifier for the other seven stroke pair relations without no-relation (_) could improve the performance of stroke-level parser a little bit. But the difference is very small. Because we require the Edmonds' algorithm not to choose an edge with label no-relation (_).

The stroke pair feature we use for training stroke pair relation classifier are parzen window shape context feature(angle number = 18, distance number = 15), geometric feature and the time gap feature. We do not think using different parzen window shape context feature could improve the performance of stroke-level parser ensemble much, because the difference is small among different (angle number, distance number) combinations as showed in Section 4.5.

Structured Boosting combines stroke-level parsers sequentially. It is natural to think about combining ensemble parsers parallel as an alternative. We use Random Forest to combine parallel symbol-level parsers. The training of different symbol-level parsers is independent. Unlike stroke-level parser ensemble, symbol-level parser ensemble sees convergence towards increased accuracy. Its performance gets better when the number of symbol-level parsers is smaller than 80.

Motivated by the high accuracy of using Random Forest and visual feature for symbol segmentation followed by symbol relationship parsing, we are going use ensembles (Random Forest) in individual stages (symbol segmentation and symbol pair relation classification followed by MST extraction via Edmonds' algorithm) rather than combining parsing trees directly. We are going to cover this different but simpler approach in Chapter 6.

Chapter 6

MST-Based Parsing

Most of the current math expression recognition systems use Cocke Younger Kasami (CYK) parsing. Hundreds of production rules in context-free grammar need to be designed carefully by people for different datasets. They explore all the possible interpretations and use the score of the production rule to measure the quality of the interpretation. Therefore the quality of grammar is the key to these methods. The time complexity is usually exponential if there are no constraints.

In this section, we are going to find whether simple and intuitive parsing method could compete with the grammar-driven parsing which requires extensive manual design and tweak of the grammar. Based on the observation that the mathematical expression recognition is searching a symbol layout tree in the graph representation, we propose MST-Based parsing with Edmonds' algorithm.

There are two important metrics we use to evaluate the parsing algorithm: structure rate and expression rate. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. When there is no symbol classifier involved, correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct, and symbol's ground truth label is used to label the node in the symbol layout tree.

6.1 Parsing with Perfect Segmentation

To remove the effect of segmentation error on layout analysis, we use ground truth symbol candidate for parsing. In this section, all the parsing are based on the perfection segmentation.

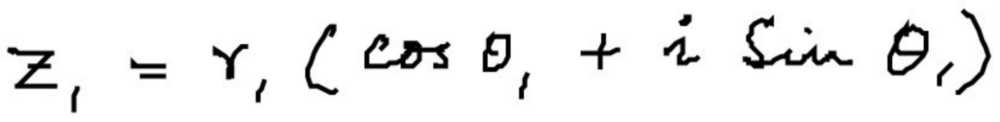
With the ground truth segmentation, symbol level line of sight graph is constructed. Multi-class symbol pair relation classifier produces a score list for each edge in the symbol level LOS graph. The maximum spanning tree extracted by Edmonds' algorithm is taken as the recognition result.

Table 6.3 shows the comparison of parsing results between perfect segmentation and segmentation with errors on validation dataset. Expression rate, structure rate are 78.63% and 83.76% with the perfect segmentation. Structure rate is the percentage of expressions which get the correct structure. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Our segmentation method mentioned in Section 6.2.2 achieves 95.20% segmentation recall, and 93.87% segmentation. Based on this segmentation, expression rate, structure rate are 57.26% and 60.68%. We can find 5% of the segmentation error reduces the expression rate by almost 21%.

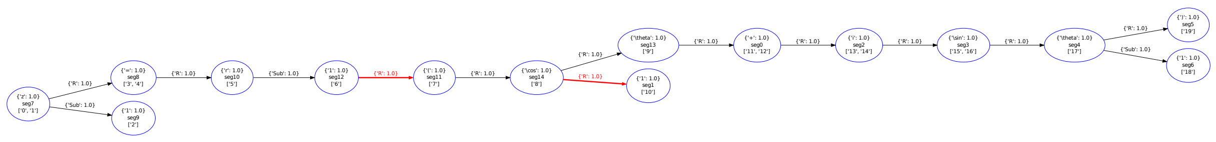
Figure 6.1 shows an recognition failure example of one relation multiple children. In the directed graph, the first 'cos' have two children 'θ' and '1', while the relation between symbol pair 'cos' and 'θ' and symbol pair 'cos' and '1' are both 'Right'. This recognition result is not a valid symbol layout tree. Because it violates the grammars of math expression that for a given parent symbol, it only could have one child symbol for a given relationship.

6.1.1 One Relation One Child Symbol Constraint

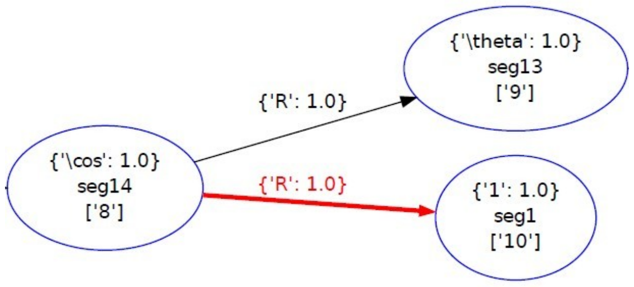
To solve the one relation multiple children symbol error showed in Figure 6.1, we add one relation one child symbol constraint to the Edmonds' algorithm. During the modified Edmonds' algorithm, it would not select an edge if adding the edge violates the one relation one child symbol constraint. This means it would not



(a) ground truth



(b) directed graph



(c) part of directed graph

Figure 6.1: Failure example of one relation multiple children

select the spanning tree if there is one relation multiple children symbol.

With perfect segmentation, we do experiments to compare the parsing with or without the one relation one child symbol constraint. Table 6.3 shows the experiment results comparison between parsing with or without the one relation one child symbol constraint on validation dataset. It is easy to find that adding the one relation one child symbol constraint could improve the performance but does not help that much.

6.1.2 Syntax Grammar Constraint

How to incorporate language model into the parsing is challenging. We add syntax grammar constraint into the parsing. Syntax grammar constraint defines which symbol pair relation tuple (parent symbol, child symbol, symbol pair relation) is legal based on the syntax label and relation between the symbol pair. For the symbol pair 'cos' and 'θ' in the Figure 6.1, its symbol pair relation tuple is ('other', 'letter', 'Right'), as the syntax label of 'cos' is 'other', the syntax label of 'θ' is 'letter' and the relation between them is 'Right'.

The motivation of syntax grammar constraint is removing illegal symbol pair relation tuple before the Edmonds' algorithm. It could reduce the search space for the parsing and then improve the parsing performance when the syntax label is correct. For the symbol pair in the line of sight graph, the multi-class symbol pair relation classifier produces a score list which contains a score for each relation. After adding the syntax grammar constraint, if the symbol pair relation tuple is not a legal one, the score for the corresponding would be 0. This means the illegal symbol pair relation tuple would not be selected by the Edmonds' algorithm.

Manually designed Syntax Grammar Constraint

Like most of the other grammar based on parsing methods, we first design the syntax grammar constraint manually. Table 6.1 shows all the illegal symbol pair relation tuple which are designed based on the CROHME 2012 grammar. For example, ('letter', 'letter', 'Above') is illegal, which means a letter can not be above on the other letter.

Table 6.1: Manually designed syntax grammar constraint. A means Above; B means Below; Sup means superscript; Sub means subscript.

parent symbol syntax label	child symbol syntax label	illegal relation
letter	letter	A, B, Inside
letter	digit	A, B, Inside
letter	other	Sup, Sub, Inside
digit	letter	A, B, Sub, Sup, Inside
digit	digit	A, B, Sub, Sup, Inside
digit	other	Sub, Sup, Inside
other	letter	
other	digit	
other	other	Sub, Sup

Data Based Syntax Grammar Constraint

Manually designed syntax grammar constraints are different for different datasets, and do not have good generalization ability.

Data based syntax grammar constraint means we extract the syntax grammar constraint from the training data. If a symbol pair relation tuple appears in the training data, we take it as legal; otherwise, we take it as illegal. Therefore, all the symbol pair relation tuples which do not show up in the training data are taken as the illegal symbol pair relation tuple set. One potential problem for data based syntax grammar constraint is that symbol pair relation tuple which shows in test set but not in training set would be filtered, and this would make the parsing of the expressions which contains those symbol pair relation tuple wrong.

Figure 6.2 shows the syntax bigram distribution when the relation is ’Right’. This shows in the dataset, there is no (letter, digit, Right) tuple. Therefore, before parsing, all (letter, digit, Right) would be filtered.

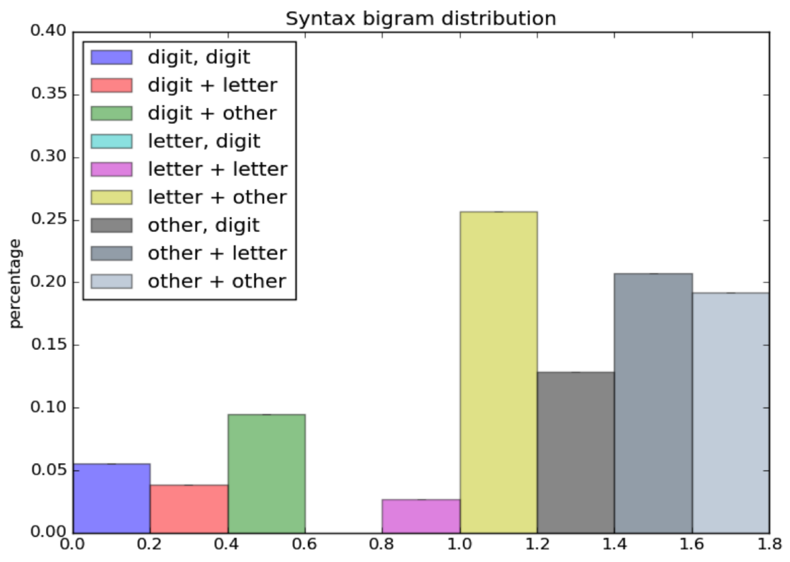


Figure 6.2: Syntax bigram distribution when the relation is ’Right’

Algorithm Based Syntax Grammar Constraint

Osorio et. al [64] propose an algorithm to determine a string is legal or not for a given context-free grammar. The algorithm is based on Cocke-Younger-Kasami parsing, and it could deal with any string which only contains the terminal defined in the language model.

Because symbol pair relation tuple can be represented in a latex string, and the CROHME dataset grammar is context-free grammar, we could use the algorithm in [64] to extract all legal symbol pair relation tuple.

We update the CROHME grammar to make it can cover all the expressions in the datasets. Then we generate the corresponding context-free grammar in Chomsky normal form [20]. For each possible symbol pair relation tuple, we generate its latex string and use the algorithm in [64] to determine it is legal or not. In this way, we can get the illegal symbol pair relation tuple set, and all the illegal symbol pair relation tuples would be filtered before the parsing.

Table 6.2 shows the comparison of parsing results with different syntax grammar constraints on validation set. With data based syntax grammar constraint, the expression rate is improved by 4%.

Because some legal symbol pair relation tuples do not appear in the datasets. There are more constraints in the data based syntax grammar constraint than the algorithm based syntax grammar constraint. This makes more symbol pair relation tuples are filtered with the data based syntax grammar constraint than algorithm based syntax grammar constraint. Data based syntax grammar constraint performs better algorithm based syntax grammar constraint on this dataset. But the situation could change if there are more symbol pair relation tuples which are in the test set but not the training set.

Table 6.2: Comparison of parsing with different syntax grammar constraint on CROHME 2012 validation set. Symbol segmentation is perfect. GT means we use the symbol's ground truth label to get its syntax label, and this means the syntax label is 100% correct. C means we use our symbol classifier to the symbol's syntax label. For the parsing without syntax grammar constraint, we do not need the symbol's syntax label. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. Symbol's ground truth label is used to label the node in the symbol layout tree.

	expression rate	structure rate
without syntax grammar constraint	79.49%	83.76%
manually designed syntax grammar constraint (GT)	81.20%	84.62%
data based syntax grammar constraint (GT)	83.76%	84.62%
algorithm based syntax grammar constraint (GT)	80.34%	84.62%
algorithm based syntax grammar constraint (C)	68.38%	75.21%

6.1.3 Syntax Grammar Constraint Filtering with Classified Syntax Label

All the syntax grammar constraint related experiments mentioned in Section 6.1.2 use the symbol's ground truth label to get the syntax label. Therefore the syntax label is 100% accurate.

If the syntax class label is not 100% accurate, does syntax grammar constraint still improve the parsing? We use Davila's classifier [23] to produce the syntax class label. Then we filter the symbol pair relation tuple based on the classified syntax label.

Table 6.2 shows with the classified syntax label, using the syntax grammar constraint could make the parsing even worse than without any syntax grammar constraint. It supports that if we can get the correct syntax label for symbols, we can filter the redundant symbol pair relation tuple. In this way, we reduce the search space for parsing, and could acquire better structure rate and expression rate as showed in Table 6.2. But if we cannot get the correct syntax label for symbols, the correct symbol pair relation tuple could be filtered, and this would make the parsing cannot select the correct label for the edge between the symbol pair. Therefore, parsing performance would become worse as showed in Table 6.2 with the classified syntax label.

We cannot come to the conclusion that only knowing symbol's syntax label with 100% accuracy can make syntax grammar constraint useful. But the syntax label produced by our classifier cannot provide help for parsing as there are some classification errors. Therefore, we would not use syntax grammar constraint to filter the symbol pair score list produced by the multi-class symbol pair relation classifier.

6.2 Edmonds' Algorithm

The MST-based parsing is based on Edmonds' Algorithm. The time complexity of Edmonds' Algorithm is low $O(|E| \times |V|)$, where $|V|$ is the number of nodes and $|E|$ is the number of edges. In the worst case, $|E| = |V| \times (|V| - 1)$. Therefore, the time complexity of MST-based parsing is $O(|E| \times |V|) = O(|V|^2 \times |V|) = O(|V|^3) = O(N^3)$, where N is the number of symbols in the expression. As we use symbol level line of sight graph as the graph representation, the $|E|$ seldom reaches that worst case.

6.2.1 Simultaneous Segmentation and Layout Analysis

Both symbol segmentation and layout analysis are about the label of edge in the graph; we try to solve symbol segmentation and symbol layout analysis as one problem. All the existing systems take segmentation and spatial relationship classification as two separate tasks.

For a given expression, we construct stroke level line of sight graph. For each stroke pair, we extract the Parzen Window shape context feature, geometric features and time tap feature mentioned in Section 4.5.2. After the multi-class stroke pair relation Random Forest classifier (mentioned in Section 4.5.2) produces a score list for each edge in the line of sight graph, we apply Edmonds' algorithm mentioned in Section 5.4 to extract the maximum spanning tree from the LOS graph. For the score list produced by the Random Forest, there is a confidence score for each possible stroke pair relation. In order to make the stroke level LOS a connected graph, we add a dummy node. There is an edge from dummy node to each stroke node. Symbol layout tree recovered from the stroke level MST is taken as the recognition result.

Algorithm 7 Simultaneous Segmentation and Layout Analysis

```
for Each expression: do
    1. Form stroke level line of sight graph.
    2. Add dummy node and there is an edge from dummy node to each stroke node.
    3. Get MST at stroke level based on Edmonds' algorithm.
    4. Remove the dummy node and the outgoing edges from dummy node.
    6. Recover symbol layout tree from the MST at stroke level.
end for
```

The simultaneous segmentation and layout analysis achieves 95.06% segmentation recall, 90.28% segmentation precision and 32.48% expression rate on validation set.

6.2.2 Segmentation Before Layout Analysis

There are more classes for the stroke level task. Taking segmentation and layout analysis as one problem is harder than either symbol segmentation or symbol pair relation classification. Experiment results in Section 4.5, 4.6 show that we can get higher accuracy when trying to classify relationships at the symbol level than the stroke level.

Instead of solving symbol segmentation and layout analysis as a single stroke pair relation classification, we do symbol segmentation first, and then do layout analysis based on the segmentation result.

Segmentation

For a given expression we still construct stroke level line of sight graph. Then we apply the binary stroke pair relation classifier to the edges in stroke level line of sight graph. Each edge would get a label: either merge or split. We take each connected component in the LOS graph as a symbol candidate.

Layout Analysis

Based on the symbol candidates produced in segmentation, symbol level line of sight graph is constructed. Multi-class symbol pair relation classifier produces a score list for each edge in the symbol level line of sight graph. After Edmonds' algorithm extracting the symbol level maximum spanning tree form the LOS graph, the MST is taken as the recognition result.

Table 6.3 shows the experiment results comparison on validation set between taking segmentation and layout analysis as one task or two tasks. It is easy to find that taking them as two tasks performs better than dealing symbol segmentation and layout analysis at the same time. Dealing segmentation and layout analysis separately achieves 95.20% segmentation recall, 93.87% segmentation precision and 57.26% expression rate on validation set. While simultaneous segmentation and layout analysis get 95.06% segmentation recall, 90.28% segmentation precision and 32.48% expression rate. The reason is the symbol pair relation classification is more accuracy than stroke pair relation classification as showed in Section 4.5, 4.6. The other reason is small improve in segmentation accuracy can cause a big improvement in expression rate. Any mis-segmentation means that an expression will be wrong. On average, there are almost 12 symbols in each expression for the CROHME 2012 training dataset. When the segmentation recall is α, the possibility that an expression gets perfect segmentation is α^{12}. 1% improvement in segmentation recall can increase the possibility by $(\alpha + 0.01)^{12} - \alpha^{12}$.

Table 6.3: Comparison among different MST-based parsing on CROHME 2012 validation set. One task means simultaneous segmentation and layout analysis. Two tasks mean segmentation before layout analysis. Parsing with perfection segmentation also means segmentation before layout analysis, but the segmentation is 100% accurate. Constraint means one relation one child. The one relation one child constraint does not apply to 'one task' and 'two tasks'. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

	expression rate	structure rate
one task	32.48%	36.75%
two tasks	57.26%	60.68%
parsing with perfection segmentation	78.63%	83.76%
parsing with perfection segmentation with constraint	79.49%	84.62%

6.3 Backtracking parsing with Perfect Segmentation

The MST-based parsing only selects the label for the edges in the symbol layout tree, but we also need the label for the nodes. We did try to incorporate the classification score produced by symbol classifier into the score list of symbol pair edge produced by symbol pair relation classifier, and let the MST-based parsing select both the edge label and node label for the symbol layout tree. But the performance is not good.

Then we come to the idea of backtracking parsing. Algorithm 8 shows the pseudo code of backtracking parsing with perfect segmentation. It is a recursive algorithm. It aims to select symbol layout tree ($N - 1$ edges) from the symbol level line of sight graph.

The input for the backtracking parsing algorithm are Edges, N and Solution. The Edges are all edges in the symbol level line of sight graph. Each edge contains (parent symbol label, child symbol label, symbol pair relation, score). The symbol pair relation could be one of the six classes (Right, Superscript, Subscript, Above, Below, Inside). The symbol label is the original symbol label, such as 'a', 'b', '1' and so on. The symbol label is based on the top X results produced by the symbol classifier. The score of each edge is the summation of P(parent symbol), P(child symbol), and P(symbol pair relation). If we use top 5 symbol classification results, for each LOS edge, there are 5 * 5 * 6 possible (parent symbol label, child symbol label, symbol pair relation, score) combinations. We use the symbol pair relation tuple constraint mentioned in Section 6.1.2 to filter the invalid combinations. N is the number of symbols in the given expression. Solution is initialized as an empty list. The output is the Solution with $N - 1$ edges.

Algorithm 9 shows whether adding the edge would cause constraint violation or not. If adding the edge would cause a cycle or multiple parent symbols for a given symbol, or conflict about the symbol's label, then it would be not added.

Algorithm 8 Backtracking parsing with Perfect Segmentation

```
BacktrackingParsing(Edges, N, Solution)
if Solution has N-1 edges then
    return Solution
else
    sort all edges in Edges based on the score from high to low
    for each edge in Edges do
        if NoConstraintViolation(edge, Solution) then
            Solution.push(edge)
            BacktrackingParsing(Edges, N, Solution)
            Solution.pop(edge)
        else
        end if
    end for
end if
```

Algorithm 9 Check Constraint Violation

```
NoConstraintViolation(edge, Solution)
if edge is not used AND no cycle AND all symbol has at most one parent AND symbol label is consistent
then
    return TRUE
else
    return FALSE
end if
```

Table 6.4 shows the comparison of backtracking parsing results with different number of symbol candidates. When the edge score is the summation of P(parent symbol), P(child symbol), and P(symbol pair relation), using the top 3 symbol classification results get the best performance. When the edge score is the product, using top 5 symbol classification results get the best performance. In addition, the performance is better with the product of P(parent symbol), P(child symbol), and P(symbol pair relation) than the summation with the top 3 or top 5 symbol classification results.

Table 6.4: Comparison of backtracking parsing results with different top N symbol classification results on CROHME 2012 validation set. S means the edge score is the summation of P(parent symbol), P(child symbol), and P(symbol pair relation), while P means the edge score is the product of P(parent symbol), P(child symbol), and P(symbol pair relation). Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We use symbol's ground truth label to label the node in the symbol layout tree.

	expression rate	structure rate
top 5 symbol classification results (S)	51.28%	52.99%
top 3 symbol classification results (S)	53.85%	56.41%
top 1 symbol classification results (S)	50.43%	48.72%
top 5 symbol classification results (P)	58.97%	67.52%
top 3 symbol classification results (P)	58.97%	65.81%
top 1 symbol classification results (P)	44.44%	48.72%

6.4 Pre-segmentation

As mentioned in Section 4.5.1 many segmentation errors are related to the multi-stroke symbols, such as these function names, such as $\cos$, $\sin$, $\tan$, $\log$, $\lim$. Pre-segmentation might improve the segmentation performance.

Pre-segmentation is a specific pattern detector. We take those function names $\cos, \sin, \tan, \log, \lim$ as specific patterns, and we train a binary symbol classifier to classify a symbol candidate as a specific pattern or not. We move the sliding window with different window size (1,2,3,4) along the time series to get the training samples. For training set, there are 755 specific patterns, and only 2 can not be covered by the sliding window. For validation set, there are 77 specific patterns and only 1 can not be covered. In the training dataset, more than 99% of the samples are non specific pattern. In order to make the training set balanced, we use the distorter mentioned in [23] to generate some synthetic specific patterns to make the negative positive sample ratio be almost 1.

After training the specific detector, for a given expression, we move the sliding window to get the symbol candidate. If the specific detector produces a confidence score higher than the threshold, we take the symbol candidate as a specific pattern, and it would be taken as a symbol in symbol segmentation.

Figure 6.3 shows the recall, precision and F-score for the binary specific pattern detector. When the score threshold is 0.25, the F-score is the highest. But in our case, a high precision is more desirable than a high recall. Therefore, we would choose a high score threshold in order to get a high precision.

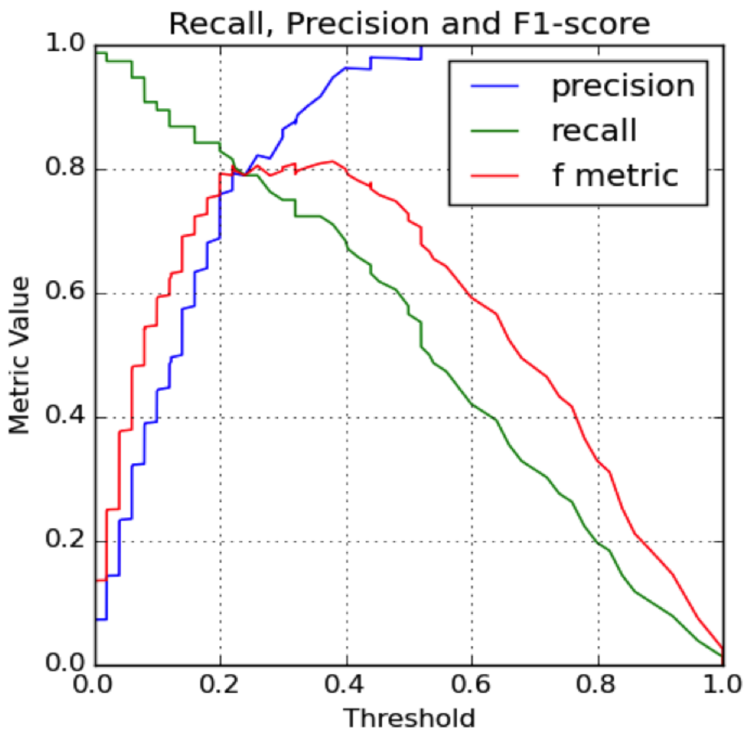


Figure 6.3: Segmentation recall, precision and expression rate with different thresholds.

Figure 6.4 shows the segmentation recall, precision and expression rate with different thresholds. When score threshold is 0.9 or 0.91, the performance is the best. Segmentation recall, precision and expression rate are 95.81%, 95.68%, 62.39%.

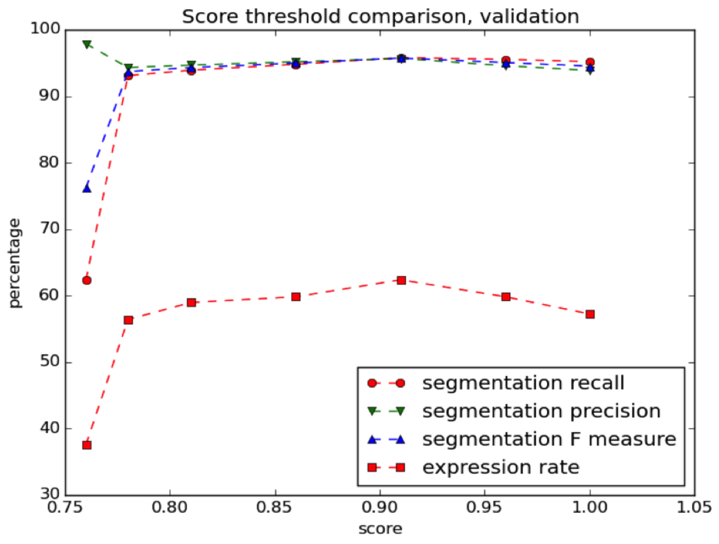


Figure 6.4: Segmentation recall, precision and expression rate with different thresholds.

Table 6.5: Comparison between with and without presegmentation on CROHME 2012 validation set. Segmentation means segmenting symbols. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

	symbol recall	symbol precision	expression rate
without presegmentation	95.20%	93.87%	57.26%
with presegmentation	95.81%	95.68%	62.39%

6.5 Experiment Results and Analysis

The experiment results of parsing based on Edmonds' algorithm for CROHME 2012 and 2014 datasets.

Table 6.6 shows the performance of MST-based parsing when the segmentation is perfect. We construct symbol level line of sight graph based on the perfect segmentation. Then multi-class symbol pair relation classifier produces score list for each edge in symbol level line of sight graph. Finally, Edmonds' algorithm is used to find the symbol layout tree from the line of sight graph.

Table 6.6: Parsing with perfect segmentation on CROHME 2012/2014 datasets. Segmentation means segmenting symbols. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

dataset	relation recall	relation recall + class	structure rate	expression rate
CROHME 2012 train	99.96%	99.96%	99.40%	99.40%
CROHME 2012 test	96.16%	93.16%	72.63%	67.70%
CROHME 2014 train	99.73%	99.67%	96.92%	96.63%
CROHME 2014 test	95.51%	91.08%	76.67%	67.44%

Table 6.7 shows the comparison between our MST-based segmentation and CYK parsing based on scholastic context-free grammars (SCFG) [70] when the symbol segmentation and symbol classification are known. Simistira et. al design three sets of SCFG manually, and apply CYK parsing based on the SCFG. They assume symbol segmentation and symbol label are known. The symbol pair they use is based on SVM and the reported error rate is 2.8% which is close to the error rate of our symbol pair relation classifier. The time complexity of CKY algorithm is $O(n^3|G|)$, where n is the number of symbol in the expression while $|G|$ is the number of grammars.

It is easy to find that our MST-based parsing achieves more than 15% higher structure rate and 10% higher expression rate than the CYK parsing based on SCFG [70]. In addition, we do not assume the symbol's label is known in the parsing while [70] does. With symbol's label, we could expect another almost 2% improvement in structure rate and expression rate by filtering invalid symbol pair with language constraints before Edmonds' algorithm as showed in Table 6.2.

Table 6.7: Comparison between our MST-based parsing and CYK parsing based on scholastic context-free grammar (SCFG) on CROHME 2012 test set. Assume symbol segmentation and symbol classification are known.

	structure rate	expression rate	time complexity
our MST-based parsing	72.63%	67.70%	N^3
Simistira CYK parsing based on SCFG [70]	57.41%	56.37%	$N^3\lvert G\rvert$

Table 6.8 shows the performance of MST-based parsing when the segmentation is produced by the binary stroke pair relation classifier. We construct the stroke level line of sight graph first. Then binary stroke pair relation classifier classifies each edge in the stroke level LOS as merge and split. After taking each connected component as one symbol candidate, symbol level line of sight graph is constructed based on the symbol candidates. Multi-class symbol pair relation classifier produces score list for each edge in the symbol level line of sight graph. Finally, symbol layout tree is extracted based on the Edmonds' algorithm.

The results in Table 6.8 are based on we use ground truth symbol label. While Table 6.9 shows the results when the symbol label is produced by Davila's Classifier [23].

Table 6.8: Parsing with segmentation produced by binary stroke pair relation classifier on CROHME 2012/2014 datasets. R means recall. Segmentation means segmenting symbols. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. We do not classify symbol here and use symbol's ground truth label.

dataset	segmentation R	relation R	relation R + class	structure rate	expression rate
CROHME 2012 train	100%	99.96%	99.96%	99.40%	99.40%
CROHME 2012 test	94.87%	88.19%	85.51%	55.76%	51.85%
CROHME 2014 train	99.98%	99.69%	99.64%	96.77%	96.48%
CROHME 2014 test	92.41%	83.83%	79.89%	57.20%	50.71%

Table 6.9: Parsing with segmentation produced by binary stroke pair relation classifier on CROHME 2012/2014 datasets. R means recall; SEG means segmentation; REL means symbol pair relation; STR means structure, EXP means expression. Segmentation means segmenting symbols. Correct structure means we get the structure of the symbol layout tree correct, even though the label of edge in the symbol layout tree might be wrong. Correct expression means we get the structure of the symbol layout tree correct and all the edge labels are correct. Symbol label is produced by Davila's Classifier [23].

dataset	SEG R	SEG R + class	REL R	REL R + class	STR rate	EXP rate
CROHME 2012 train	100%	99.57%	99.96%	99.96%	99.40%	94.54%
CROHME 2012 test	94.87%	87.13%	88.19%	85.51%	55.76%	33.74%
CROHME 2014 train	99.98%	98.02%	99.69%	99.64%	96.77%	83.07%
CROHME 2014 test	92.41%	81.95%	83.83%	79.89%	57.20%	26.88%

Table 6.10 shows the comparison between our system and previous work on CROHME 2014 test set. Our system can get higher segmentation F-score than other systems, but the expression rate is 10% lower than the winner [2] of CROHME 2014, but higher than the other systems. The gap between our system and Alvaro's system [2] is caused by lower segmentation recall + class and lower relation recall + class. Although MST-based parsing performs very well when the symbol segmentation and symbol classification are accurate. It does not perform well when there are errors in symbol segmentation and symbol classification, not even correct the errors. MyScript achieves the highest expression rate on CROHME 2014 dataset [58]. They are not comparable because they used a different, large training set.

The time complexity of our system is N^3, and it is lower than the time complexity $N^3|G|$ of the systems which use CYK parsing with context-free grammars, where N is the number of stroke in the expression while $|G|$ is the number of grammars. Our MST-based parsing technique could be extended to include n-grams or other language constraints. They can be used to filter the invalid symbol pairs before Edmonds' algorithm and get better performance.

Table 6.10: Performance Comparison between our system and previous work on CROHME 2014 test set. R means recall; SEG means segmentation; REL means symbol pair relation; 'class' means the label is correct; G means the system uses a grammar. Segmentation means segmenting symbols.

dataset	SEG R	SEG F-score	SEG R + class	REL R + class	expression rate	G	time
our system	92.41%	92.43%	81.95%	79.89%	26.88%		N^3
Alvaro [2]	93.31%	92.00%	86.59%	84.23%	37.22%	✓	$N^3\|G\|$
Awal et. al [9]	89.43%	87.75%	76.53%	71.77%	26.06%	✓	$N^3\|G\|$
Le et. al [45]	83.05 %	84.19%	69.72%	66.83%	25.66%	✓	$N^3\|G\|$
Hu et. al	85.52%	85.80%	76.64%	70.78%	18.97%		
Yao and Wang	88.23%	86.17%	78.45%	61.38%	18.97%		
Aguilar [37]	76.63%	78.41%	66.97%	60.31%	15.01%	✓	$N^3\|G\|$
Myscript	98.42%	98.27%	93.91%	94.26%	62.68%	✓	

6.6 Summary

In this section, we proposed an MST-based parsing algorithm with Edmonds' algorithm. The time complexity of our MST-based parsing is lower than the time complexity of CYK parsing with context-free grammars.

We find taking symbol segmentation and layout analysis as two tasks can get better performance than dealing with them at the same time. Our segmentation method (applying binary stroke pair relation classification on stroke level line of sight graph) gets the highest symbol segmentation F-score on CROHME 2014 test set. Adding the syntax grammar constraint could improve the parsing performance when the syntax label is correct. But the errors of the syntax label could make the parsing performance worse than without the syntax grammar constraint.

MST-based parsing could achieve higher structure rate and expression rate than CYK parsing when symbol segmentation is accurate. But it is sensitive to segmentation error, a small increase of segmentation error rate could cause a big drop in expression rate. As MST-based parsing performs better than CYK parsing when the segmentation is accurate, but worse when there are errors in symbol segmentation and symbol classification. It would be very interesting to research how to integrate symbol segmentation and symbol classification into MST-based parsing to correct the errors in segmentation and classification in the future.

Chapter 7

Conclusion and Future Work

7.1 Contributions

The contributions of this thesis are in fourfold: (1) We propose line of sight graph for math expression representation. It can represent math expression better than other graph representations. (2) We propose Parzen window shape context feature and find Parzen window shape context feature can get competitive results in symbol segmentation, symbol classification and symbol layout analysis against other features. (3) We propose and develop MST-based parsing with Edmonds' algorithm. It has better time complexity and better performance than CYK parsing when the segmentation is correct. (4) We apply structured learning on math expression recognition for the first time and identify holistic recognition is a good direction for math expression recognition.

7.1.1 Graph Representation

We can understand recognizing the structure of an expression as selecting a tree from a graph, and then classifying edges in the tree. Therefore, a good graph representation is essential to the math expression recognition.

We propose the line of sight graph representation with convex hull representing the polygon shape of the stroke. LOS achieves the highest F-score among different graph representation, because it represents the visual structure of the math expression. Its performance is stable on the four different date sets (CROHME 2012 training/test, CROHME 2014 training/test). The edge recall is higher than 99.9% while precision is higher than 30%.

7.1.2 Visual Feature

A feature set which can be calculated based on a simple rule and works well for symbol segmentation, symbol classification and symbol pair relation classification is highly desired.

We propose and develop shape context feature modified by Parzen window density estimation. We find that visual features (Parzen window shape context features) could get competitive results in symbol segmentation, symbol classification, symbol layout analysis against manually designed feature, and could be complementary information for different feature sets in the three tasks to improve performance.

Using Parzen window shape context feature and line of sight graph gets the higher symbol segmentation F-score than other systems on CROHME 2014 test set. Also, our Parzen window shape context feature performs better than original shape context feature for stroke pair relation classification. Using syntax context feature and MST context feature could improve performance for symbol segmentation and symbol pair relation classification if we know the symbol's syntax label and edge label.

7.1.3 Structured Learning

We apply structured learning for handwritten math expression recognition for the first time. To the best of our knowledge, nobody has used structured learning for math expression recognition before. We use structured Boosting for stroke-level parser ensemble and holistic recognition for symbol-level parser ensemble.

Structured Boosting combines stroke-level parsers sequentially. The training of later stroke level parser depends on the training of its previous stroke level parser. But structured Boosting does not work well for math expression recognition. Unlike stroke-level parser ensemble, symbol-level parser ensemble sees convergence towards increased accuracy. The result is promising and it shows the holistic recognition might be a good direction for math expression recognition.

7.1.4 MST-based Parsing

We propose and develop MST-based parsing with Edmonds' algorithm. The time complexity of our MST-based parsing is lower than the time complexity of CYK parsing with context-free grammars. MST-based parsing could achieve higher structure rate and expression rate than CYK parsing when symbol segmentation is accurate. Also, our MST-based parsing does not require any manually designed grammar nor constrain user's writing style, and therefore has better generalization ability than Stochastic context-free grammars based parsing. Our math expression recognition system with MST-based parsing and visual features achieves very competitive performance on CROHME 2012/2014 datasets.

7.2 Future Work

For future work, more effort could be put into Structured Boosting and holistic recognition. For structure AdaBoost, we need to figure out what global metric we want to optimize, and how to use the recognition result at expression level to update the weight of training sample at stroke level or symbol level during training. For holistic recognition, how to get a powerful parselet and how to combine recognition results of different parselets would be two interesting research directions. For MST-based parsing, how to incorporate symbol classifier result and correct segmentation error needs more research.

Bibliography

[1] F. Alvaro, J. Sanchez, and J. Benedi. Classification of on-line mathematical symbols with hybrid features and recurrent neural networks. In *International Conference on Document Analysis and Recognition*, pages 1012–1016, Aug. 2013.

[2] F. Alvaro, J.A. Sanchez, and J.M. Benedi. Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 35:58–67, 2014.

[3] F. Alvaro and R. Zanibbi. A shape-based layout descriptor for classifying spatial relationships in handwritten math. In *ACM Symposium on Document Engineering*, pages 123–126, Sep. 2013.

[4] W. Aly, S. Uchida, A. Fujiyoshi, and M. Suzuki. Statistical classification of spatial relationships among mathematical symbols. In *International Conference on Document Analysis and Recognition*, pages 1350–1354, July 2009.

[5] W. Aly, S. Uchida, and M. Suzuki. A large-scale analysis of mathematical expressions for an accurate understanding of their structure. In *International Workshop on Document Analysis Systems*, pages 549–556, Sep. 2008.

[6] Robert H. Anderson. Syntax-directed recognition of hand-printed two-dimensional mathematics. In *Symposium on Interactive Systems for Experimental Applied Mathematics: Proceedings of the Association for Computing Machinery Inc. Symposium*, pages 436–459, 1967.

[7] Lisa Anthony, Jie Yang, and Kenneth R. Koedinger. Adapting handwriting recognition for applications in algebra learning. In *Proceedings of the International Workshop on Educational Multimedia and Multimedia Education*, pages 47–56, Sep. 2007.

[8] Ahmad-Montaser Awal, Harold Mouchere, and Christian Viard-Gaudin. Towards handwritten mathematical expression recognition. In *International Conference on Document Analysis and Recognition*, pages 1046–1050, 2009.

[9] Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. A global learning approach for an online handwritten mathematical expression recognition system. *Pattern Recognition Letters*, 35:68–77, 2014.

[10] Suyash P Awate. *Adaptive, nonparametric markov models and information-theoretic methods for image restoration and segmentation*. PhD thesis, The University of Utah, 2006.

[11] Gükhan H. Bakir, Thomas Hofmann, Bernhard Schölkopf, Alexander J. Smola, Ben Taskar, and S. V. N. Vishwanathan. *Predicting Structured Data (Neural Information Processing)*. The MIT Press, 2007.

[12] Abdelwaheb Belaid and Jean-Paul Haton. A Syntactic Approach for Handwritten Mathematical Formula Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(1):105–111, 1984.

[13] S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(4):509–522, 2002.

[14] Mark de Berg, Otfried Cheong, Marc van Kreveld, and Mark Overmars. *Computational Geometry: Algorithms and Applications*. Springer-Verlag TELOS, 3rd ed. edition, 2008.

[15] Dorothea Blostein and A. Grbavec. Recognition of mathematical notation. *Handbook of Character Recognition and Document Image Analysis*, pages 557–582, 1997.

[16] R.G. Casey and E. Lecolinet. A survey of methods and strategies in character segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(7):690–706, 1996.

[17] M. Celik and B. Yanikoglu. Probabilistic mathematical formula recognition using a 2d context-free graph grammar. In *International Conference on Document Analysis and Recognition*, pages 161–166, Sep. 2011.

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

[19] Jinho D. Choi, Joel R. Tetreault, and Amanda Stent. It depends: Dependency parser comparison using A web-based evaluation tool. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing*, pages 387–396, July 2015.

[20] Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

[21] P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. In *Intelligent Robotics Systems Conference*, pages 852–865, 1989.

[22] Yoeng-Jin Chu and Tseng-Hong Liu. On shortest arborescence of a directed graph. *Scientia Sinica*, 14(10):1396–1400, 1965.

[23] Kenny Davila, Stephanie Ludi, and Richard Zanibbi. Using off-line features and synthetic data for on-line handwritten math symbol recognition. In *International Conference on Frontiers in Handwriting Recognition*, pages 323–328, Sep. 2014.

[24] Jack Edmonds. Optimum branchings. *Journal of Research of the National Bureau of Standards B*, 71(4):233–240, 1967.

[25] Y. Eto and M. Suzuki. Mathematical formula recognition using virtual link network. In *International Conference on Document Analysis and Recognition*, pages 762–767, Sep. 2001.

[26] Claudie Faure and Zi Xiong Wang. Automatic perception of the structure of handwritten mathematical expressions. *Computer processing of handwriting*, pages 337–361, 1990.

[27] J.A. Fitzgerald, Franz Geiselbrechtinger, and Tahar Kechadi. Mathpad: A fuzzy logic-based recognition system for handwritten mathematics. In *International Conference on Document Analysis and Recognition*, pages 694–698, Sep. 2007.

[28] U. Garain and BB Chaudhuri. Recognition of online handwritten mathematical expressions. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 34(6):2366–2376, 2004.

[29] A. Grbavec and D. Blostein. Mathematics recognition using graph rewriting. In *International Conference on Document Analysis and Recognition*, pages 417–421, Aug. 1995.

[30] Jaekyu Ha, R.M. Haralick, and I.T. Phillips. Understanding mathematical expressions from document images. In *International Conference on Document Analysis and Recognition*, pages 956–959, Aug. 1995.

[31] Nina S. T. Hirata and Willian Y. Honda. Automatic labeling of handwritten mathematical symbols via expression matching. In *International Conference on Graph-based Representations in Pattern Recognition*, pages 295–304, May 2011.

[32] L. Hu, K. Hart, R. Pospesel, and R. Zanibbi. Baseline extraction-driven parsing of handwritten mathematical expressions. In *International Conference on Pattern Recognition*, pages 326–330, Nov. 2012.

[33] L. Hu and R. Zanibbi. Segmenting handwritten math symbols using adaboost and multi-scale shape context features. In *International Conference on Document Analysis and Recognition*, pages 1212–1216, Aug. 2013.

[34] Lei Hu and R. Zanibbi. Hmm-based recognition of online handwritten mathematical symbols using segmental k-means initialization and a modified pen-up/down feature. In *International Conference on Document Analysis and Recognition*, pages 457–462, Sep. 2011.

[35] Thorsten Joachims, Thomas Hofmann, Yisong Yue, and Chun-Nam Yu. Predicting structured objects with support vector machines. *Communications of the ACM*, 52(11):97–104, Nov. 2009.

[36] Frank D. Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina S. T. Hirata. Top-down online handwritten mathematical expression parsing with graph grammar. In *IberoAmerican Congress on Pattern Recognition*, pages 444–451, Nov. 2015.

[37] F. JulcaAguilar, N.S.T. Hirata, C. ViardGaudin, H. Mouchere, and S. Medjkoune. Mathematical symbol hypothesis recognition with rejection option. In *International Conference on Frontiers in Handwriting Recognition*, pages 500–505, Sep. 2014.

[38] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques (Adaptive Computation and Machine Learning series)*. The MIT Press, 2009.

[39] M. Koschinski, H.-J. Winkler, and M. Lang. Segmentation and recognition of symbols within handwritten mathematical expressions. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 2439–2442, May 1995.

[40] Andreas Kosmala and Gerhard Rigoll. On-line handwritten formula recognition using statistical methods. In *International Conference on Pattern Recognition*, pages 1306–1308, Aug. 1998.

[41] Andreas Kosmala, Gerhard Rigoll, Stephane Lavirotte, and Loic Pottier. On-line handwritten formula recognition using hidden markov models and context dependent graph grammars. In *International Conference on Document Analysis and Recognition*, pages 107–110, Sep. 1999.

[42] N. Kwak and Chong-Ho Choi. Input feature selection by mutual information based on parzen window. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(12):1667–1671, Dec. 2002.

[43] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*, pages 282–289, Nov. 2001.

[44] Stephane Lavirotte and Loic Pottier. Mathematical formula recognition using graph grammar. In *SPIE Photonics West Electronic Imaging*, pages 44–52, 1998.

[45] Anh Duc Le, Truyen Van Phan, and Masaki Nakagawa. A system for recognizing online handwritten mathematical expressions and improvement of structure analysis. In *International Workshop on Document Analysis Systems*, pages 51–55, Apr. 2014.

[46] S. Lehmberg, H.-J. Winkler, and M. Lang. A soft-decision approach for symbol segmentation within handwritten mathematical expressions. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 3434–3437, May 1996.

[47] Scott Maclean and George Labahn. Elastic matching in linear time and constant space. In *International Workshop on Document Analysis Systems*, pages 551–554, 2010.

[48] Scott MacLean and George Labahn. A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. *International Journal on Document Analysis and Recognition*, 16(2):139–163, 2013.

[49] Scott MacLean and George Labahn. A bayesian model for recognizing handwritten mathematical expressions. *Pattern Recognition*, 48(8):2433–2445, 2015.

[50] Scott MacLean, George Labahn, Edward Lank, Mirette Marzouk, and David Tausky. Grammar-based techniques for creating ground-truthed sketch corpora. *International Journal on Document Analysis and Recognition*, 14(1):65–74, 2010.

[51] S. Marinai, B. Miotti, and G. Soda. Using earth mover's distance in the bag-of-visual-words model for mathematical symbol retrieval. In *International Conference on Document Analysis and Recognition*, pages 1309 –1313, Sep. 2011.

[52] Simone Marinai, Beatrice Miotti, and Giovanni Soda. Mathematical symbol indexing using topologically ordered clusters of shape contexts. In *International Conference on Document Analysis and Recognition*, pages 1041–1045, July 2009.

[53] N.E. Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, May 1999.

[54] Ryan McDonald, Fernando Pereira, Kiril Ribarov, and Jan Hajivc. Non-projective dependency parsing using spanning tree algorithms. In *International Conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 523–530, Oct. 2005.

[55] Yang Mingqiang, Kpalma K. Idiyo, and Ronsin Joseph. A Survey of Shape Feature Extraction Techniques. *Pattern Recognition*, pages 43–90, 2008.

[56] H. Mouchere, C. Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and U. Garain. Crohme2011: Competition on recognition of online handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition*, pages 1497–1500, Sep. 2011.

[57] Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. ICFHR 2012 competition on recognition of on-line mathematical expressions (CROHME 2012). In *International Conference on Frontiers in Handwriting Recognition*, pages 811–816, Sep. 2012.

[58] Harold Mouchère, Christian Viard-Gaudin, Richard Zanibbi, and Utpal Garain. ICFHR 2014 competition on recognition of on-line handwritten mathematical expressions (CROHME 2014). In *International Conference on Frontiers in Handwriting Recognition*, pages 791–796, Sep. 2014.

[59] Harold Mouchere, Christian Viard-Gaudin, Richard Zanibbi, Utpal Garain, Dae Hwan Kim, and Jin Hyung Kim. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition*, pages 1428–1432, Aug. 2013.

[60] Harold Mouchère, Richard Zanibbi, Utpal Garain, and Christian Viard-Gaudin. Advancing the state of the art for handwritten math recognition: the crohme competitions, 2011–2014. *International Journal on Document Analysis and Recognition*, pages 1–17, 2016.

[61] G. Nagy and S. Seth. Hierarchical representation of optically scanned documents. In *International Conference on Pattern Recognition*, pages 347–349, 1984.

[62] Sebastian Nowozin and Christoph H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3–4):185–365, 2011.

[63] Nasayuki Okamoto and Bin Miao. Recognition of mathematical expressions by using the layout structures of symbols. In *International Conference on Document Analysis and Recognition*, pages 242–250, Sep. 1991.

[64] Mauricio Osorio and Juan Antonio Navarro. Decision problem of substrings in context free languages. In *Memorias del X Congreso Interna-cional de Computacion*, pages 239–249, 2001.

[65] L. Ouyang. A Symbol layout classification for mathematical formula using layout context. Master's thesis, Rochester Institute of Technology, Rochester, NY, 2009.

[66] Ling Ouyang and Richard Zanibbi. Identifying layout classes for mathematical symbols using layout context. In *IEEE Western New York Image Processing Workshop*, 2009.

[67] Robert E. Schapire and Yoav Freund. *Boosting: Foundations and Algorithms*. The MIT Press, 2012.

[68] Yu Shi, HaiYang Li, and F.K. Soong. A unified framework for symbol segmentation and recognition of handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition*, pages 854–858, Sep. 2007.

[69] Foteini Simistira, Vassilis Papavassiliou, Vassilis Katsouros, and George Carayannis. Structural analysis of online handwritten mathematical symbols based on support vector machines. In *SPIE Document Recognition and Retrieval*, number 580–586, Feb. 2013.

[70] Fotini Simistira, Vassilios Katsouros, and George Carayannis. Recognition of online handwritten mathematical formulas using probabilistic svms and stochastic context free grammars. *Pattern Recognition Letters*, 53:85–92, 2015.

[71] E. Smirnova and S.M. Watt. Communicating mathematics via pen-based interfaces. In *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 9–18, Sep. 2008.

[72] Steve Smithies, Kevin Novins, and James Arvo. A handwriting-based equation editor. In *International Conference on Graphics Interface*, pages 84–91, 1999.

[73] Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[74] M. Suzuki, S. Uchida, and A. Nomura. A ground-truthed mathematical character and symbol image database. In *International Conference on Document Analysis and Recognition*, pages 675–679, Sep. 2005.

[75] Ernesto Tapia and Raúl Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In *International Workshop on Graphics Recognition, Recent Advances and Perspectives*, pages 329–340, July 2003.

[76] K. Toyozumi, N. Yamada, T. Kitasaka, K. Mori, Y. Suenaga, K. Mase, and T. Takahashi. A study of symbol segmentation method for handwritten mathematical formula recognition using mathematical structure information. In *International Conference on Pattern Recognition*, pages 630–633, Aug. 2004.

[77] Zhuowen Tu, Xiangrong Chen, A.L. Yuille, and S.-C. Zhu. Image parsing: unifying segmentation, detection, and recognition. In *International Conference on Computer Vision*, pages 18–25, Oct. 2003.

[78] Zhuowen Tu, Xiangrong Chen, Alan L. Yuille, and Song Chun Zhu. Image parsing: Unifying segmentation, detection, and recognition. *International Journal of Computer Vision*, 63(2):113–140, 2005.

[79] V. N. Vapnik and A. Ya. Chervonenkis. *Theory of Pattern Recognition*. Nauka, 1974.

[80] B.Q. Vuong, Yulan He, and S.C. Hui. Towards a web-based progressive handwriting recognition environment for mathematical problem solving. *Expert Systems with Applications*, 37(1):886–893, 2010.

[81] Qin Iris Wang. Learning structured classifiers for statistical dependency parsing. In *Human Language Technology Conference of the North American Chapter of the Association of Computational Linguistics*, pages 5–8, Apr. 2007.

[82] Qin Iris Wang, Dekang Lin, and Dale Schuurmans. Simple training of dependency parsers via structured boosting on artificial intelligence. In *International Joint Conference on Artificial Intelligence*, pages 1756–1762, Jan. 2007.

[83] Qiu-Feng Wang, Fei Yin, and Cheng-Lin Liu. Handwritten chinese text recognition by integrating multiple contexts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(8):1469–1481, Aug. 2012.

[84] H.-J. Winkler. HMM-based handwritten symbol recognition using on-line and off-line features. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 3438–3441, May 1996.

[85] H.-J. Winkler and M. Lang. Online symbol segmentation and recognition in handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 4, pages 3377 –3380 vol.4, apr 1997.

[86] Hans-jürgen Winkler and Manfred Lang. Symbol segmentation and recognition for understanding handwritten mathematical expressions. In *Progress in Handwriting Recognition*, pages 407–412, 1997.

[87] Ryo Yamamoto, Shinji Sako, Takuya Nishimoto, and Shigeki Sagayama. On-line recognition of handwritten mathematical expressions based on stroke-based stochastic context-free grammar. In *International Workshop on Frontiers in Handwriting Recognition*, pages 249–254, Oct. 2006.

[88] Daniel H. Younger. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189–208, 1967.

[89] Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition*, 15(4):331–357, 2012.

[90] Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1455–1467, 2002.

[91] Richard Zanibbi, Harold Mouchère, and Christian Viard-Gaudin. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In *SPIE Document Recognition and Retrieval*, pages 810–817, Feb. 2013.

[92] Ling Zhang, D. Blostein, and R. Zanibbi. Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In *International Conference on Document Analysis and Recognition*, pages 972–976, Sep. 2005.