Recognition of Online Handwritten Mathematical Expressions using Contextual Information

Frank Julca-Aguilar

UNIVERSITE BRETAGNE LOIRE

IME-USP

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Frank D. JULCA-AGUILAR

Mémoire présenté en vue de l'obtention du
grade de Docteur de l'Universidade de São Paulo et
Docteur de l'Université de Nantes
sous le sceau de l'Université Bretagne Loire

École doctorale : STIM (503)

Discipline : Informatique et applications
Spécialité : Informatique
Unité de recherche : Institut de Recherche en Communications et Cybernétique de Nantes (IRCCyN) et Instituto de Matemática e Estatística da Universidade de São Paulo (IME-USP)

Soutenue le 29 avril 2016

Recognition of Online Handwritten Mathematical Expressions using Contextual Information

JURY

Rapporteurs :
M. Jean-Yves RAMEL, Professeur, Université de Tours
M. Ricardo da Silva TORRES, Professeur, Universidade de Campinas

Examinateur :
M. Bertrand COUASNON, Maître de conférences, HDR, INSA de Rennes

Co-directrice de Thèse :
M^{me} Nina S. T. HIRATA, Professeur, Universidade de São Paulo

Co-directeur de Thèse :
M. Christian VIARD-GAUDIN, Professeur, Université de Nantes

Co-encadrant (membre invité) :
M. Harold MOUCHÈRE, Maître de conférences, Université de Nantes

Aknowledgments

This thesis would not have been possible without the support of many people. First, I would like to thank my advisors Nina S. T. Hirata, Christian Viard-Gaudin and Harold Mouchère. Nina, thank you for sharing countless hours of discussions, for your advice and support, and for being also a good friend along these last years. Christian and Harold, thank you for sharing your knowledge and experience with me. Thank you all for reading my drafts of reports and papers, and giving me many helpful comments. You provided me with an ideal balance of guidance and freedom.

I am also thankful to my co-authors Willian Y. Honda and Alexandre Noma, for your collaboration during the development of this work. I am grateful to the members of the Computer Vision Group of the Institute of Mathematics and Statistics of University of São Paulo, and members of the *Institut de Recherche en Communications et Cybernétique de Nantes* of University of Nantes. Thanks for providing me a friendly working environment.

I also want to thank my amazing friends Anthony Mendez, Karina Espinoza, Rosario Medina, Jorge Guevara, Leissi Castañeda, Leandro Ticlia, Alfonso Phocco, Juan Gutierrez, Renzo Gomez, Edwin Delgado, Sofiane Medjkoune, Alejandra Rios, Luis Céspedes. Thank you Vanessa Valdiviezo for the wonderful years on my side. Thank you Rosangela Camargo for such lovely moments you gave me. I am really lucky for finding so great people as all of you.

I also acknowledge the financial support of my studies by FAPESP, through a Masters scholarship, Grant number 2010/04491-0; a Direct Doctorate scholarship, Grant number 2012/08389-1; and a Research Internships Abroad (BEPE) scholarship, Grant number 2013/13535-0.

Finally, I would like to thank my parents, Isabel Aguilar and Paulino Julca; my sisters, Carmen, Sonia, Manuela; and my brothers, Samir, Ivan, and Julio. Thanks for the amazing support, encouragement, and love you provide me. The passion for exploring, learning, and hard work that you taught me led me through my studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In this thesis, we study the online handwritten mathematical expressions recognition problem and propose a new technique to automatize the recognition of these type of data.

This chapter introduces the recognition problem and presents an overview of the thesis. Section 1.1 gives practical and scientific motivations. Section 1.2 introduces the recognition problem and its main challenges. Section 1.3 then defines the main goals, outlines the proposed recognition techniques, and lists the main contributions of this thesis. Finally, Section 1.4 describes the thesis organization.

1.1 Motivation

Common ways to enter mathematical expressions into computers are inefficient. They are based on special typesetting commands (such as TEX) or symbol selection tools (as the one of *MS-Word*). Those methods require from users some expertise in managing the input mechanics: in the first case, the user must be able to memorize or have access to a large number of codes and syntactic rules; in the second case, symbols and structures must be selected one by one, using keyboards and mouses as interfaces. Writing complex expressions using such methods is far from being an easy task.

Automatic recognition of mathematical expressions would provide more efficient means to enter these data into computers: expressions could be drawn on a touch screen device using an electronic pen or a finger, and a system would recognize the expression and convert it to a desirable format, TEX , for instance.

Other applications of automatic recognition of mathematical expressions include:

- digitalization of mathematical expressions in scientific documents: handwritten or printed documents could be digitalized, edited, and recorded in a database;
- document readers: software programs could read books, articles or even handwritten notes with a speech synthesizer or braille display;
- simplification of inputting data into mobile devices: mathematical expressions could be drawn directly on smartphones or tablets.

From a general point of view, mathematical expressions are not much different from other *two-dimensional languages*. Mathematical expressions may be represented as groups of symbols placed

over a two-dimensional space and a set of relations among the symbols. Chemical equations, music notation, and flowcharts, for example, could also be represented in a similar way. This suggests that a recognition technique for mathematical expressions should not be much different from others used for those languages. Thus, in an ideal case, an effective recognition technique for mathematical expressions should also be adaptable to the automatic recognition of a variety of two-dimensional languages.

From a scientific point of view, the mathematical expression recognition problem comprises several pattern recognition challenges: segmentation, classification, parsing, and machine learning. The difficulties of the problem are not only related to solving those issues separately, but also in the integration of them. In this regard, Zanibbi and Blostein (2012) describes the problem as an excellent vehicle for studying methods of integrating patter recognition algorithms to improve performance.

Both practical applications and research challenges have been appealing to the pattern recognition community, specially during the last decade. This is evidenced by the variety of published works devoted to propose new recognition techniques (Álvaro and Zanibbi, 2013; MacLean and Labahn, 2013; Simistira et al., 2015; Yamamoto et al., 2006), to build publicly available datasets (Aguilar and Hirata, 2012; Hirata and Julca-Aguilar, 2015; MacLean et al., 2011; Quiniou et al., 2011), and to create new techniques for performance evaluation (Awal et al., 2010a; Lapointe, 2008; Zanibbi et al., 2013). Along with those efforts, several competitions on recognition of mathematical expressions have been organized, with the aim of providing an effective framework to determine the state of the art in this problem (Mouchère et al., 2012, 2013, 2014, 2011).

1.2 The recognition problem

Research on recognition of mathematical expressions began in the 1960s (Anderson, 1968); but, compared to text recognition, it is still considered as being in early stages of research (Zanibbi and Blostein, 2012).

Handwritten mathematical expressions can be *offline* or *online*. In the first case, the expressions are written on paper and then they are digitalized by scanning the paper; in the second case, the expressions are written with an electronic pen on a touch screen device. In online data, the two-dimensional coordinates of the writing as a function of time are recorded. The input data then consists of a sequence (ordered set) of strokes – being a stroke a sequence of coordinates collected from pen down to pen up – ordered by the input time. Figure 1.1 shows an online handwritten mathematical expression example. As shown in that Figure, the stroke coordinates may not be homogeneously sampled. In addition, they often present abrupt changes of direction or other characteristics that may decrease the recognition performance. Therefore, before the mathematical recognition process itself is done, a preprocessing step is usually applied in order to obtain data invariant to those characteristics.

Recognition of online handwritten mathematical expressions requires solving three tasks: (1) symbol segmentation, (2) symbol classification, and (3) structural analysis. The first task consists in grouping strokes that form a same symbol; the second consists in identifying which mathematical symbol represents each group of strokes; and the third aims to identify spatial relations between symbols – as the *superscript* relation between symbols "a" and "b" and the *horizontal* relation

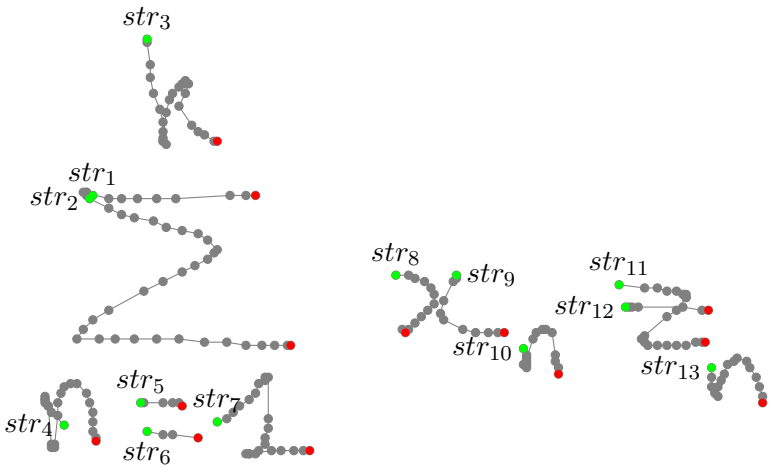


Figure 1.1: *Online handwritten mathematical expression sample of the CROHME-2014 dataset (Mouchère et al., 2014). The sample represents the expression* $\sum_{n=1}^{k} x_n z_n$. *It is composed of a sequence of strokes* $str = \{str_1, \ldots, str_{13}\}$, *where* str_i *is the* i^{th} *stroke, considering the input order. Each stroke is composed of a sequence of two-dimensional coordinates, represented as a sequence of points, where green and red points indicate the first and last point of each stroke, respectively.*

between "a" and "c" in the expression "$a^b c$". Each of the tasks involves several issues that make the recognition problem challenging.

In symbol segmentation, evaluating all possible stroke partitions has an exponential cost. A symbol may consists of strokes that are not consecutive according to their input order. For example, in the expression of Figure 1.1, someone could write the stroke str_{11} (part of symbol "z"), then write the stroke str_{13} (symbol "n"), and, finally, add the stroke $str12$ (the rest of symbol "z"). This lack of consecutiveness makes possible that, given a sequence of strokes $str = (str_1, \ldots, str_n)$, a symbol could be formed by any non empty subset of str. Evaluating all symbol segmentations of str would correspond to evaluate the B_n partitions of str, where B_n (Bell number) satisfies the recursion:

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k, \tag{1.1}$$

with $B_0 = 1$.

In symbol recognition, the major issues include the large number of symbol classes and ambiguity. To cover a considerable domain of mathematics, for example, Awal et al. (2010b) suggest that about two hundred symbol classes should be considered. Among all the classes, those with similar shape are specially difficult to differentiate. Figure 1.2 shows some ambiguous handwritten symbols. It is important to note that this ambiguity is mainly present in handwritten data. In printed data, most of those cases may be solved using symbols' size or relative positions.

(a) (b) (c) (d)

Figure 1.2: *Ambiguous handwritten symbols. They could be interpreted as (a) "+" or "t", (b) "6" or "G", (c) "9" or "q" and (d) "P" or "p". Extracted from CROHME-2014 dataset (Mouchère et al., 2014).*

Structural analysis of mathematical expressions is challenging due to ambiguity in the identification of the different relation types and the need for syntax validation. While in textual data only horizontal relations link consecutive symbols (that is, a symbol is related only to another one at its left or right), in mathematical expressions not only horizontal relations, but also vertical and oblique relations (as in expressions "$\sum_{x}$" and "x^{y}", respectively) link any pair of symbols. Among the relation types, specially difficult recognition cases arise when trying to differentiate *superscript* and *subscript* relations from *horizontal* relations. The variety of relations invalidate the left to right order of symbols assumed in text recognition. The lack of that order disallows the direct application of text recognition algorithms to mathematical expression recognition. On the other hand, syntax validation is required to generate a coherent interpretation of a given input. For instance, if an open parenthesis has been identified at some part of an expression, to have a *valid* expression, a closing parenthesis should also be identified at some other part, and, the closing parenthesis should be in the same baseline of the open parenthesis.

As a result of the above issues, even the recognition of a small expression, as the one of Figure 1.3, may be difficult. The expression in the figure presents ambiguous cases at the three recognition tasks. Given those ambiguities, a system could generate equally likely interpretations as "$P^b < 1$", "$p^b < 1$", "$P^b 4$" or "$p^b 4$".

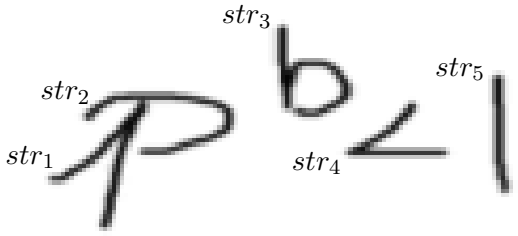


Figure 1.3: *Ambiguous mathematical expression. The expression is composed of five strokes, that is* $(str_1, \ldots, str_5)$*. Strokes* $\{str_4, str_5\}$ *could be interpreted as the symbol "4" or as a subexpression "*< 1*",* $\{str_1, str_2\}$ *could be interpreted as symbol "p" or "P" and the relation between* $\{str_1, str_2\}$ *and* $\{str_3\}$ *could be horizontal or superscript.*

An important observation in this problem is that the interpretation of an expression varies according to the context. For instance, in Figure 1.3, if the strokes $\{str_4, str_5\}$ are interpreted as a symbol "4", it would be more likely that strokes $\{str_1, str_2\}$ represent a symbol "P", rather than "p". In fact, contextual information could be used to solve ambiguities. In Figure 1.3, for example, even though the isolated strokes $\{str_4, str_5\}$ seem to represent the symbol "4", if we consider the whole strokes structure, specifically distances between strokes, it is most probable that the stroke "$<$" alone represents a comparator and the other strokes represent the compared subexpressions.

In order to exploit contextual information into a mathematical expression recognition process, a technique should be able to integrate information related to each of the three subprocesses (symbol segmentation, classification, and structural analysis) in order to generate a best interpretation of a given input. The development of a global optimal solution is a main challenge in this problem.

Mathematical notation is formed by a variety of structural organization of its elements (fraction, subscripting, matrix, case equations, etc). Since being able to deal with all of them at once is a too complex task, recognizers are often developed restricting structure types. For instance, only recently matrix-like structures started to be considered in some recognition algorithms (Mouchère et al., 2014). In this regard, an ideal recognition method is one that is not tied to particular structures

or does not impose strong constraints on the recognizable expressions. Such method could then be applied on new structures without need of modifications.

One of the main difficulties of the evaluation of handwritten mathematical expression recognition techniques has been the lack of publicly available datasets (Lapointe and Blostein, 2009; MacLean et al., 2011). Although some datasets have been built along the last years (MacLean et al., 2011; Mouchère et al., 2014; Quiniou et al., 2011), larger datasets are still needed. For instance, MNIST[1] is a well-known dataset of handwritten digits, that has been used to evaluate and compare a number of classification techniques. The dataset has about 70,000 samples, for 10 symbol classes (isolated numbers from zero to nine), having about 7,000 samples per class. In contrast, the CROHME-2014 dataset (Mouchère et al., 2014) has about 100,000 symbol samples for 101 classes, being the samples per class highly unbalanced (some symbol classes have about 5% of the samples and others less than 0.1%). The need for more data is also motivated by the wide variety of structures that should be covered. As an example of this variety, we can consider the different relative positions between symbols in the expressions "pb", "pq", "Pb", and "Pq". Although all the expressions present an horizontal relation between symbols, the relative positions vary according to the symbol class. The larger the number of symbol classes and relation types, the larger the dataset is needed to cover the variety of structures.

One of the main difficulties for the creation of mathematical expression datasets is the task of attaching ground-truth at different levels of the expressions, that is, stroke, symbol and relation levels (Lapointe, 2008; Lapointe and Blostein, 2009). Manually attaching symbol labels, and relations into the expressions is a long time-consuming process. Building a consistent dataset is by itself a complex problem, which requires addressing issues related to relevance, completeness, and correctness of the data.

1.3 This thesis

1.3.1 Objectives

In this thesis we study the online handwritten mathematical expressions recognition problem. We aim to develop an effective and extensible online handwritten mathematical expression recognition technique.

In order to evaluate the effectiveness of the technique, we measure its performance at different levels, including symbol segmentation and classification, relation classification and complete expression recognition. We report a detailed analysis of our technique's performance and compare it with state-of-the-art techniques. In addition to the practical analysis, we also analyze our technique, and compare it with previous works, at an abstract level, considering aspects as the formulation of the problem, constraints, and limitations. As a secondary goal, we aim to implement a complete recognition system and release it as open-source.

Part of our work in this thesis is devoted to develop a framework to automatize the creation of handwritten mathematical expression datasets. We addressed this problem by developing a system that manages the dataset creation process and a technique to automatically attaches ground-truth into mathematical expressions. The development of the last technique is a based on a previous work proposed to labeling handwritten mathematical symbols (Hirata and Honda, 2011).

[1] http://yann.lecun.com/exdb/mnist/ (visited on 05/05/2016).

1.3.2 Key ideas explored in this thesis

In this thesis, we formulate the mathematical expression recognition problem as a parsing problem. In this formulation, a stroke set str is parsed to generate a parse tree[2] t that defines a particular interpretation of str as a mathematical expression. To cope with ambiguities at symbol segmentation, classification, and structural analysis, the parsing technique generates multiple interpretations when ambiguous cases arise. We denote the set of all possible interpretations of str as $T(str)$. In addition, we define a function $cost(t, str)$ that measures the cost of considering a tree t as an interpretation of str. The parsing problem then becomes a search for a tree $t_{best}(str)$, such that:

$$t_{best}(str) = \arg\min_{t \in T(str)} cost(t, str) \tag{1.2}$$

The recognition process is illustrated in Figure 1.4. The process is composed of two main components: (1) hypotheses graph generator and (2) graph parser. The hypotheses graph generator builds a graph that defines the search space of the parsing algorithm and the graph parser does the parsing itself.

Symbol segmentation and classification

Stroke groups that are likely to represent symbols are calculated considering distance information between the input strokes. This method does not impose time-related constraints, as in Huang and Kechadi (2007); Lehmberg et al. (1996); Tapia and Rojas (2004).

The classification of symbols of mathematical expressions involves issues not included in the isolated symbol classification problem. In the first case, classifiers must deal with the identification of stroke groups that do not represent actual symbols, which does not happen in the isolated case. Also, when recognizing symbols of mathematical expressions, the neighborhood of an evaluated symbol may provide additional information to solve ambiguities. This information includes strokes that do not belong to the evaluated symbol, but that are placed close to it or even crossing over it; and the relative sizes of symbols of a same expression. For instance, the symbol "." in an expression "2.3" could look like a "0", or "o" if we had a *close-up* of it, but it would almost always look as a point if we have a view of the complete expression.

While most approaches treat the classification problem considering symbols as isolated objects, we make use of the existent information of the context of each symbol. We developed features that capture this kind of information. Furthermore, in our training scheme, we use those features to train classifiers to filter out stroke groups that do not represent actual symbols. A main advantage of the these classifiers is that they enable the development of a more efficient recognition method, as more wrong symbol interpretations are filtered out at an early stage.

Relation classification

Once symbol hypotheses have been calculated, a relation classifier identifies relations between each pair of symbols. Similarly to the symbol classification method, we also train relation classifiers to filter out *false spatial relations*.

[2] In an informal way, we could say that a parse tree defines how a grammar generates a specific object.

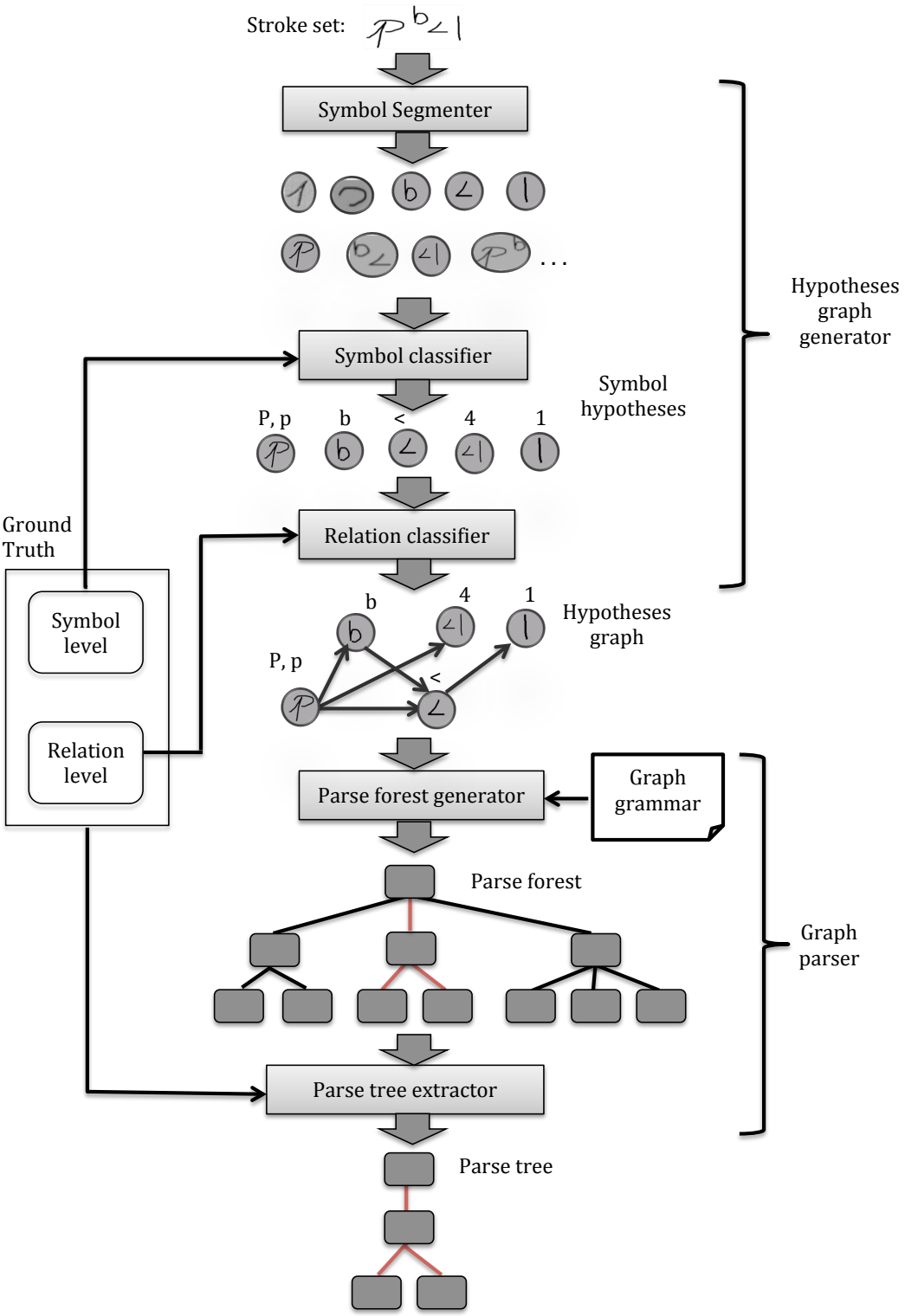


Figure 1.4: *The recognition scheme of the technique proposed in this thesis. Gray rectangles represent the main modules that compose the approach.*

A main advantage of our filtering approach for symbols and relations is that the filter is *learned* from training data, instead of being defined through time order and stroke intersection constraints (Huang and Kechadi, 2007; Lehmberg et al., 1996; Tapia and Rojas, 2004), nor defining specific rules for each relation type (MacLean and Labahn, 2013; Zanibbi et al., 2002; Zhang et al., 2005).

Modeling expressions as productions of a graph grammar

Graphs can model a broad range of two-dimensional structures. Structure components can be represented as vertices and relations among the components as edges. Vertices can encode attributes of the corresponding components and, similarly, edges can encode the type of relation between the two linked components together with relevant attributes of the relation.

In this work, symbols in an expression are represented as vertices and relations among symbols as edges. For instance, the expression $a + b$ can be represented as a graph $v_1 \overset{h}{\rightarrow} v_2 \overset{h}{\rightarrow} v_3$, where v_1, v_2, v_3 are vertices representing a, $+$, and c respectively, and the arrows are edges representing *horizontal* relations.

A general model of expressions with similar structures can be written as NT $\overset{h}{\rightarrow}$ OP $\overset{h}{\rightarrow}$ NT, where NT means a non-terminal symbol (it could be a subexpression). In a+b, OP would be $+$, but other expressions with OP as $-$, $\times$ or / would also be valid expressions.

Thus, a graph grammar describes families of expression structures and any expression that conforms to the structures are valid expressions under the grammar. Since graphs are flexible enough to model arbitrary two-dimensional structures, graph grammars are able to model any structures present in general mathematical notation.

Graph grammar parsing

Graph parsing should be modeled in such a way as to not depend on the particular object being modeled by the graph. In our case, application independence is achieved by representing elements, such as vertices and edges, by sets of labels. Thus, for parsing, labels in the graph grammar are compared to labels previously assigned to the input data, that is also modeled as a graph. Depending on the grammar restrictions and on the application domain, the labels may vary, but the graph matching algorithm (that takes into consideration coherence of the labels) does not need to be changed.

The last point is a useful property of the proposed technique. Since parsing is independent of the application domain and of particular structures being modeled, extension of the set of structures to be recognized, or adaptation of the technique to other application domain structures, can be done by just rewriting the grammar and the set of labels.

A problematic issue in graph parsing is the computational cost (Flasiński and Jurek, 2014). To mitigate this problem, we propose an input pre-processing step that aims to reduce the search space in the parsing process. The pre-processing step consists on building a graph, called hypotheses graph, that defines the structures that can be evaluated by the parsing algorithm. In this graph, we represent all stroke groups that are likely to represent a symbol as vertices with a set of labels indicating possible symbol identities. Additionally, for some pair of candidate symbols, we associate likely relation type labels. If this pre-processing is accurate, then the resulting hypotheses graph will be the expression graph itself. However, since there may be ambiguous interpretations, both

in symbol and relation recognition, the graph will likely contain false symbols and relations, and possibly multiple labels for symbol and relation labels.

1.3.3 Major contributions

The major contributions of this work are:

- A new mathematical expression recognition technique that integrates symbol segmentation and recognition and structural information into a single process.
  - We developed new symbol and relations features that obtain state-of-the-art performance. Part of these features aim to capture contextual information. The contextual features obtain better rejection rates of false symbols and relations, and keep as good accuracy as state-of-the-art features. Much of our work relative to this contributions has been originally described in Julca-Aguilar et al. (2014a,b).
  - We proposed a graph grammar model for mathematical expressions and a graph parsing technique that integrates symbol and structure level information. This integration allows to exploit contextual information within the recognition process. The proposed technique obtained state-of-the-art performance on the CROHME-2014 dataset (Mouchère et al., 2014). Our work relative to the parsing technique was originally described in Julca-Aguilar et al. (2015).
  - The graph-based representation and the application independence of the parsing method makes the recognition framework general, in the sense that it can be easily extended to recognize new structures and also be applied to two-dimensional languages other than mathematical expressions.
- Empirical evaluation of our proposed techniques on a publicly available dataset and comparison of our results with other approaches. We report a detailed analysis of our results that help to understand better the recognition achievements and suggest possible paths for further improvements.
- A system for automatic ground-truth annotation of handwritten mathematical expressions. Using this system, a dataset with 966 handwritten expressions, which contain a total of $20,010$ symbols, has been built and released as open source. Our dataset has been joined to the CROHME-2014 dataset (Mouchère et al., 2014), in order to provide a larger publicly available dataset. Our work relative to these contributions was originally described in Aguilar and Hirata (2012); Hirata and Julca-Aguilar (2015)

1.4 Thesis organization

In Chapter 2, we review previous works on the recognition problem, dividing them into *sequential approaches*, that perform, first, symbol segmentation and classification and then structural analysis; and *integrated approaches*, that integrate the three processes into a single process.

Chapter 3 describes the formalization and architecture of our proposed recognition technique. This chapter focuses on the integration of the modules and gives only a brief description of each of them. Next chapters give more details of the modules.

Chapters 4 and 5 detail our symbol and relation classification methods, respectively. We describe the proposed features, training schemes, and report the corresponding experimental results at symbol and relation recognition levels.

In Chapter 6, we focus on the graph parsing method. The parsing method is divided into two steps: (1) Parse forest construction and (2) Optimal parse tree extraction. The first step aims to determine all possible interpretations of the input. The interpretations are stored in a parse forest, in which an alternative interpretation is represented by a parse tree. The second step traverses the parse forest to extract a best tree according to a ranking function. We propose a ranking function that linearly combines costs assigned to symbol and relations that compose the parse trees.

Chapter 7 analyses the experimental results relative to the mathematical expression recognition method. To comprehensively evaluate the method, we consider performance metrics at symbol, relation, and complete expression levels.

We describe our work on a framework to build mathematical expressions datasets in Chapter 8. It includes an analysis of desired qualities that a dataset must have in order to allow effective performance evaluation; and a description of our proposed framework, a system that implements the framework, and a dataset built using such system.

Finally, Chapter 9 gives the conclusions and some thoughts on future work; and Chapter 10 presents an extended French abstract of the thesis.

Chapter 2

Previous work

Research on recognition of handwritten mathematical expressions dates back to the end of the 60's (Anderson, 1968). Although there was relatively less research in the following thirty years, since the 90's, active research has been carried out, probably due to the widespread availability of touch screen devices and more powerful computing resources.

In this chapter, we review the previous works on recognition of handwritten mathematical expressions, dividing them into:

- *sequential approaches:* that perform the three recognition processes sequentially, that is, first symbol segmentation and classification, and then structural analysis; and
- *integrated approaches:* that integrate the three processes into a single process.

This review does not aim to give an exhaustive description of all the proposed techniques, but to give an overview of previous works that served as basis to this work, and with emphasis on open problems. Additional details about previous works can be found in the surveys Blostein and Grbavec (1997); Chan and Yeung (2000); Plamondon and Srihari (2000); Tapia and Rojas (2007); Zanibbi and Blostein (2012).

A number of the proposed methods for recognition of mathematical expressions are based on parsing techniques. To support the explanation of those methods, we overview some basic concepts on parsing.

2.1 Background on parsing

A linear representation is a sequence of objects or events, as a string (or sentence), a computer program, actions on a ritual behavior, etc. A language of linear representations may be defined by a grammar. We will refer to those kinds of grammars as string grammars. Parsing is generally defined as the process of structuring a linear representation in accordance with a grammar.

A grammar is a 4-tuple (N, T, I, R) such that:

- N is a finite set of symbols, called non-terminals;
- T is a finite set of symbols, called terminals;
- $N \cap T = \emptyset$;

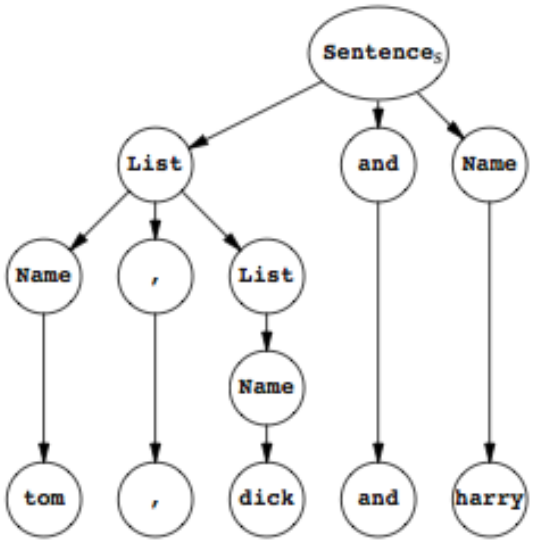
- R is a set of pairs (P, Q), called production or rewriting rules, such that $P \in (N \cup T)^+$ and $Q \in (N \cup T)^*$; and
- I, $I \in N$, is called start or initial symbol.

Figure 2.1(a) shows a grammar example. The grammar is composed of seven production rules and defines a language of lists of names separated by "," or "and". Each rule (P, Q) of the grammar is denoted as $P \rightarrow Q$. In any rule $P \rightarrow Q$, we refer to P and Q as the left-hand side (LHS) and right hand side (RHS) of the rule, respectively. The symbol "|" in a RHS of a rule separates symbols of different rules that have a same symbol in the RHS. For instance, the notation "Name → tom | dick | harry" represents three rules: "Name → tom", "Name → dick", and "Name → harry".

The structure of a linear representation, that is in accordance with a grammar, can be represented by a parse tree. Figure 2.1(b) shows a parse tree of the string "tom, dick and harry", considering the grammar of Figure 2.1(a). The structure defines hierarchical partitions of the string. These partitions are defined by grouping **adjacent** elements of the string into substrings or subexpressions. In the example, a first level partition (derived from the node labeled with the start symbol "Sentence") divides the input into three parts: "List", that includes the substring "tom, dick"; "and", that includes "and"; and "Name", that includes "harry". The substring that corresponds to the symbol "List" is further divided until obtaining substrings that correspond to terminal symbols of the grammar. To build a parse tree, a parse algorithm must determine which partitions of the string may *match* with the grammar rules.

```
0.       Name  ->  tom | dick | harry
1.  Sentence_s  ->  Name | List and Name
2.       List  ->  Name , List | Name
```

(a)



(b)

Figure 2.1: *(a) A Grammar and (b) a parse tree built with the grammar. The elements of the grammar are* $N = \{Name, Sentence_s, List\}$, $T = \{tom, dick, harry, and, ,\}$, *R is composed of the seven rules ("|" separates rules with the same non-terminal in the left hand side), and* $I = Sentence_s$. *Extracted from Grune and J.H (2008).*

Grammars are generally classified according to the Chomsky Hierarchy (Chomsky, 1956). Among the classes in that hierarchy, context-free grammars are probably the most commonly used grammars Grune and J.H (2008). Context-free grammars contain only rules that have a single non-terminal on their LHS. The grammar of Figure 2.1(a) belongs to the context-free grammar class.

Some reasons that justify the use of context-free grammars include:

- Production independence. The production of each parse tree node does not depend on what its neighbors produce. For instance, in Figure 2.1, the node "List", child of "Sentence", does not depend on the string "and harry". Thus, a parsing algorithm can be applied to different parts of a string, and the resulting subtrees can be joined into a single tree.
- Large types of languages covered.
- Efficient parsing is possible ($O(n^3)$, where n is the number of elements of the linear representation). Some algorithms that achieve such efficiency use grammars in a Chomsky Normal Form (CNF). In such grammars, all rules either have the form $A \rightarrow a$, or $A \rightarrow BC$, where a is a terminal and A, B, and C are non-terminals.

Parsing techniques provide a number of benefits when processing a linear representation. First, the result of the parsing (generally a parse tree) allows to process the input further, considering the structure found. Second, a grammar allows to represent our understanding or knowledge of the representation. Prior knowledge could then be introduced through the grammar. Third, parsers can provide completion of missing information (Grune and J.H, 2008).

Parsing techniques have also been proposed to recognize handwritten mathematical expressions. However, before applying these techniques, the grammar or the input data should be adapted to cope with the two-dimensional nature of the expressions. While in a linear representation each of its elements is adjacent only with another one horizontally (at left or right), in a mathematical expression elements can be adjacent to others horizontally, vertically, or diagonally. The multiple adjacency types (relation types) of mathematical expressions break the typical left to right order of elements of a string, and, as a result, increment the number of possible partitions of the elements into subexpressions.

2.1.1 CYK algorithm

Most of the parsing algorithms for mathematical expression recognition are based on the CYK algorithm, attributed to J. Cocke, D.H. Younger, and T. Kasami. An original description of the algorithm is given in (Younger, 1967).

The CYK parsing is a dynamic programming method that consists on building a parse tree starting with the bottom nodes (being the leaves the bottom-most nodes) and combining the partial subtrees until obtaining the complete parse tree. The algorithm has been widely used to parse linear representations, specially when parsing ambiguous languages, that is, when a same input may generate several parse trees through parsing.

The CYK algorithm involves two steps. In the first step, a table that indicates which non-terminals derive specific parts of the input (terminals) is constructed. In the second step, the non-terminals of the table are recursively combined to form all possible parse trees that can be derived by the grammar. Figure 2.2 shows an example of the CYK algorithm proposed in Yamamoto et al.

(2006) for recognition of mathematical expressions. The example uses the grammar shown in Figure 2.3. The input is an ordered set of strokes (considering input time). The input time is used to define an order of the strokes, so that this order allows the partitions to be calculated as in the case of a linear representation. The algorithm calculates the parsing of subexpressions (sets of strokes) from the smallest to the largest ones: first subexpressions of length 1 (at the bottom part of the table or matrix), then subexpressions of length 2 (second line starting from the bottom part of the table), and so on.

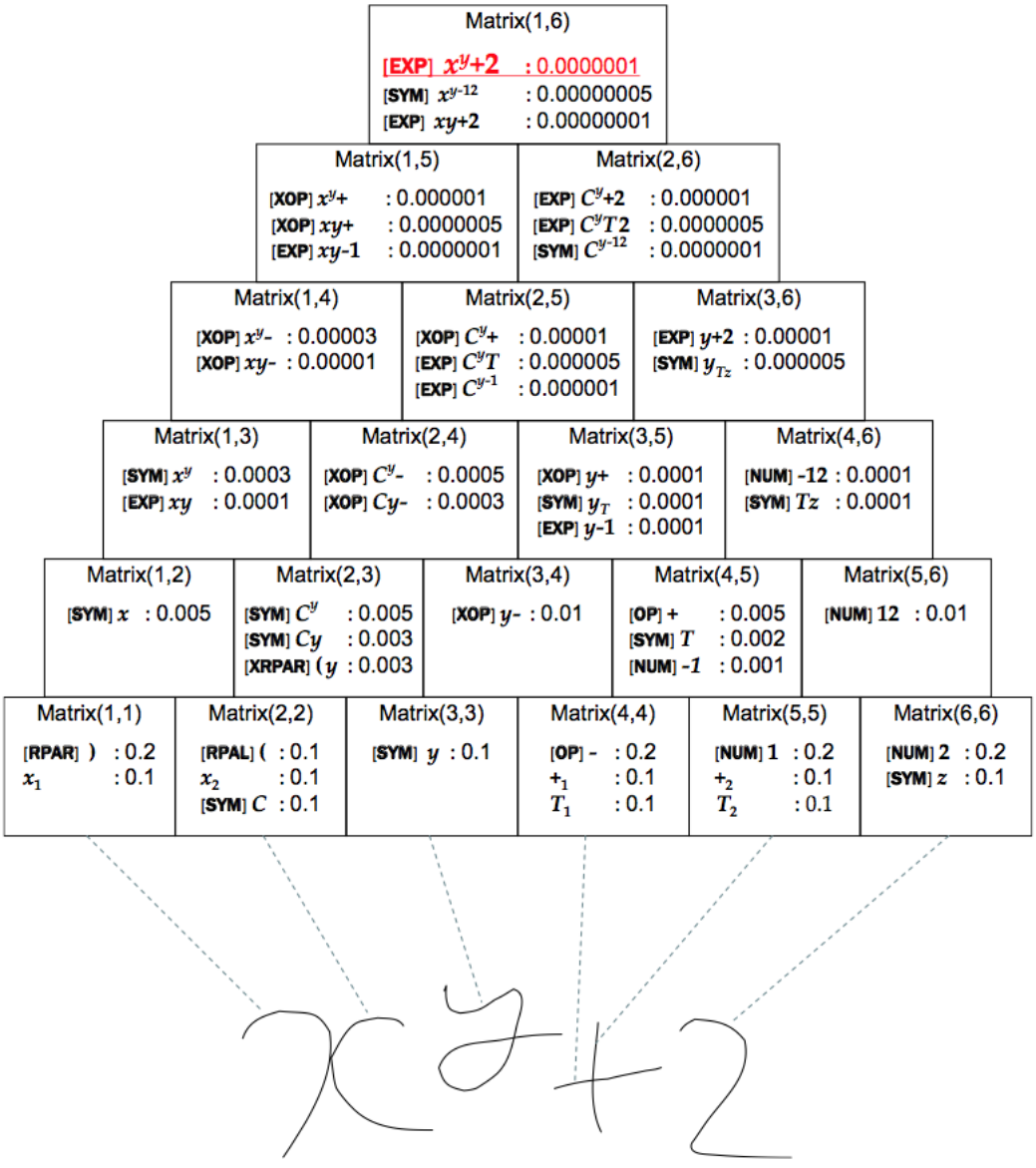


Figure 2.2: *CYK parsing example. Extracted from Yamamoto et al. (2006)*

The CYK algorithm applied to an arbitrary grammar is exponential in the maximum number of elements of the RHS of the production rules. This complexity is cubic if the grammar is transformed to a Chomsky Normal Form (CNF), as it has no more than two elements in the RHS. Although any context free grammar can be transformed into a CNF, this transformation increments the number of production rules, which may be not desirable in some applications (Lange and Leiß, 2009).

No.	Generation rule			Logical Relationship	Notes
1	EXP	→	SYM		
2	EXP	→	EXP SYM	Right	
3	SYM	→	SYM EXP	Upper Right	*
4	SYM	→	SYM EXP	Lower Right	*
5	FUNC	→	FUNC EXP	Upper	*
6	FUNC	→	FUNC EXP	Lower	*
7	DLINE	→	LINE EXP	Lower	**
8	NLINE	→	LINE EXP	Upper	**
9	SYM	→	DLINE EXP	Upper	**
10	SYM	→	NLINE EXP	Lower	**
11	SYM	→	ROOT EXP	Inside	**
12	SYM	→	ACC EXP	Accent	* **
13	SYM	→	ACC SYM	Accent	* **
14	XRPAR	→	EXP RPAR	Right	**
15	XLPAR	→	EXP LPAR	Left	**
16	SYM	→	XRPAR LPAR	Left	**
17	SYM	→	XLPAR RPAR	Right	**
18	SYM	→	$a \mid b \mid c \mid \cdots$		
19	FUNC	→	$\lim \mid \sum \mid \max \mid \cdots$		
20	LPAR	→	$(\mid [\mid \{ \mid \cdots$		
21	RPAR	→	$) \mid] \mid \} \mid \cdots$		
22	f	→	$f_1\ f_2$	Same Symbol	
23	x	→	$x_1\ x_2$	Same Symbol	
24			⋮		
25	a	→	HS		
26	f_1	→	HS		
27	FranLine	→	HS		
28			⋮		

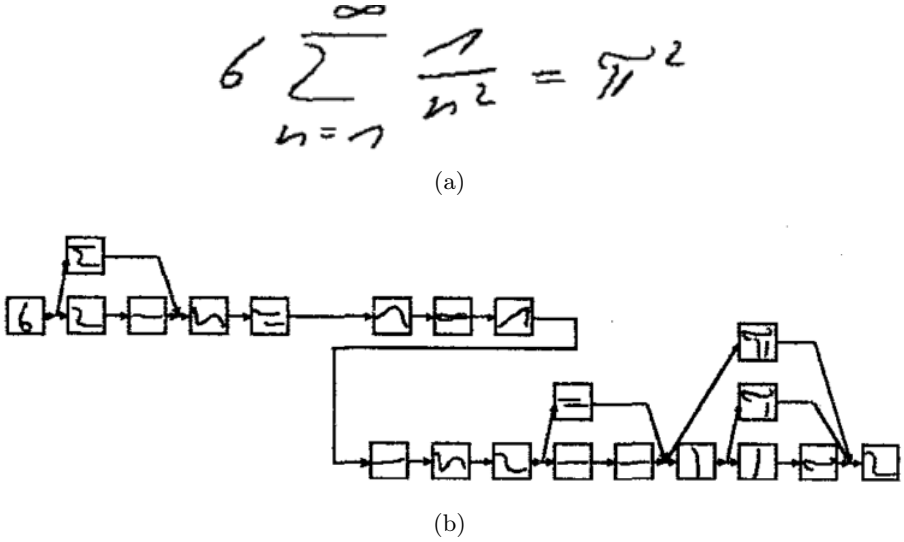
Figure 2.3: *Example of grammar used in Yamamoto et al. (2006). Rules marked with * cannot be applied iteratively. ** means that the writing order of the 2 symbols can change and that the rules with permutation of the order are included. Abbreviated names of expression elements are as follows: EXP: expression, SYM: symbol, FUNC: function, LINE: fraction line, DLINE: fraction line with denominator, NLINE: fraction line with numerator, ROOT: root sign, ACC: accent, RPAR: right parenthesis, LPAR: left parenthesis, XRPAR: expression with right parenthesis, XLPAR: expression with left parenthesis, HS: handwritten stroke.*

2.2 Sequential Approaches

In Chou (1989), the author proposed a two-dimensional stochastic context-free grammar and a parsing technique for recognition of **printed** mathematical expressions. The grammar is assumed to be in CNF. Each production rule of the form $A \rightarrow BC$ includes a flag that indicates if there is a *vertical* or *horizontal* relation between B and C. The probability distributions of the rules are calculated from training data, using a technique called Inside/Outside algorithm. The probability of a parse tree is given by the product of the probability values of its rules.

The parsing technique proposed in Chou (1989) is an adaptation of the CYK algorithm. The algorithm consists in evaluating horizontal and vertical partitions of the input. For example, for the input expression $\frac{\pi^2}{6} = \sum_{n=1}^{\infty} \frac{1}{n^2}$, it is first horizontally decomposed into factors $\frac{\pi^2}{6}$, $=$, $\sum_{n=1}^{\infty}$, and $\frac{1}{n^2}$ (considering a factor as as subexpression that can not be divided horizontally). These factors form the basis of the recognition table. Factors, $\frac{\pi^2}{6}$, $\sum_{n=1}^{\infty}$ and $\frac{1}{n^2}$ can still be divided vertically and the corresponding vertical production rules are applied to get the parsing results for each factor. The parsing results of the factors are joined to get the result of the complete expression.

In Koschinski et al. (1995) and Winkler et al. (1995), the probability of grouping strokes into symbols (segmentation) is calculated by using the minimum distance between the strokes, the portion of the intersection of their corresponding bounding boxes and also shape features. Different segmentations are stored in a symbol hypotheses net, in which a node represents a stroke group and a path from the first node to the last one represents a complete segmentation (Figure 2.4 shows a symbol hypotheses net). The probability of each stroke group being a specific symbol (symbol classification) is given by a Hidden Markov Model classifier. The probabilities given by the classifier and the segmentation step are combined to select the best segmentation and classification. The structural analysis process starts by aligning the symbols from left to right, to create a directed graph with linear structure. The edges are then classified as vertical, subscript, or superscript relations, using relative position of the symbol bounding boxes. The syntactic validation of the resulting graphs, that represent the mathematical structure, is performed by the Mathematica system (Buchberger, 1993).



(a)

(b)

Figure 2.4: *(a) Input expression and (b) symbol hypotheses net. Extracted from Koschinski et al. (1995)*

In Matsakis (1999), symbol segmentation and classification is formulated as a constrained search for a stroke partition with a minimum cost. The proposed constraint aims to avoid evaluating all possible stroke partitions. A minimum spanning tree is built using the strokes of the input expression as vertices and the distances between the strokes centers as edge costs. The search algorithm then considers only parts (stroke groups) that consist of strokes that form a connected component in the tree. Using an artificial neural network classifier, a cost is assigned to each part. This cost represents the likelihood of the part being a specific symbol and the cost of a partition is defined as the sum of the costs of all its parts. Then, the symbol classification and segmentation is determined by selecting the partition with the lowest cost. The structural analysis begins by selecting a "key" symbol – a symbol that is neither a superscript or subscript of another, nor an argument of any operator – from the symbols given by the stroke partition. The selection is based on hand-made rules, like picking the widest symbol in case of several options. Once the key symbol is chosen, all the other symbols are grouped according to their relations relative to the key symbol. The relations are calculated using relative positions between the symbols bounding boxes. The process is repeated in each group of symbols.

The work of Zanibbi et al. (2002) considers that symbols are already segmented and classified and focus only on the structural analysis task. The proposed approach builds the mathematical structure by recursively extracting baselines, where a baseline is a linear horizontal arrangement of symbols perceived as adjacent each other. The process is divided into three steps. In the first step, symbols are sorted by their minimum x coordinate, and scanned from left to right to determine the relations between each pair of consecutive symbols. This step determines a sequence of symbols that are linked by *horizontal* relations, that is a baseline. Symbols that do not belong to the baseline are grouped into regions (for instance, those that form superscript and subscript regions) relative to symbols in the baseline. Thus, the baseline extraction process is recursively applied to symbols in each region. The second step identifies composed symbols (symbols composed of other symbols in the same baseline, for instance, "sin" and "cos") and structural symbols (symbols defined in different baselines, for example, a fraction symbol). The final step determines the order of the application of the mathematical operations.

In Tapia and Rojas (2004) and Tapia (2004) a visual approach is used to segment symbols: two strokes are considered as belonging to the same symbol if there is intersection between them. The structural analysis is based on the recursive baseline extraction method of Zanibbi et al. (2002), but includes a minimum spanning tree of symbols to cope with ambiguous relations.

To identify the relations between pairs of symbols, the approaches described above calculate regions specific to each relation type. That is, given a pair of symbols A and B, the relation of B relative to A is determined by the region relative to A in which B *belongs*. Figure 2.5 illustrates regions relative to a symbol. The limits of the regions were determined by hand. A main problem of the approach is that (given the handwritten nature of the data) some symbols may lie in ambiguous regions or even in a region that does not correspond to the real relation. In addition, the manual definition of the regions makes difficult to scale the approach to recognize new structures.

To cope with symbols that lie in ambiguous regions, in Zhang et al. (2005), the authors proposed to calculate fuzzy regions around symbols. Figure 2.6 illustrates the calculation of fuzzy regions around a symbol. The fuzzy regions are used to assign a confidence value to each possible relation. This technique is integrated into a structural analysis algorithm based on the baseline extraction approach of Zanibbi et al. (2002). In this case, when comparing the relation of a symbol B relative to A, if B belongs to an intercepting region of A, two baselines are created, one per each corresponding relation. As multiple structures may be considered, this method is able to generate several interpretations of the input. The best interpretation corresponds to the one that has a highest confidence value, where the confidence value of an interpretation is computed as the product of the confidence values of the spatial relations used in the interpretation.

Although the work of Zhang et al. (2005) helps to cope with the ambiguity in relations, it still assumes that symbols are correctly segmented and classified. This is a main drawback of the approaches that perform recognition as a pipeline process, as the errors in segmentation and classification are propagated to the structural analysis step.

A first way to get rid of the pipeline limitation is to select several possible segmentations and classifications, build expression structures for each segmentation and select the most probable result based on a probability associated with the three processes. Another approach consists of doing the search for segmentations at the same time of building the structure: a set of symbol hypotheses may be generated and a structural analysis algorithm may select the best hypotheses while building the

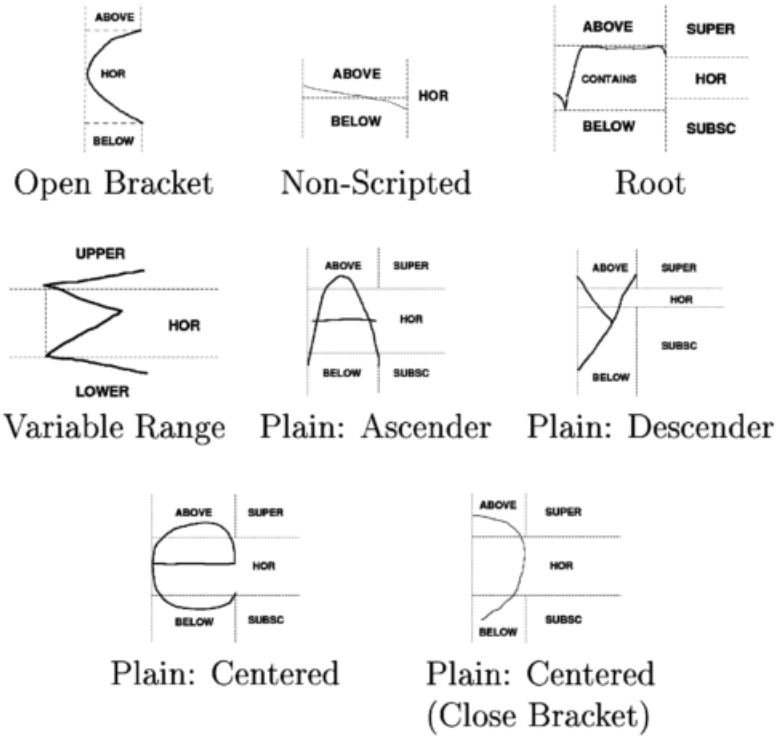


Figure 2.5: *Spatial regions relative to symbols. Extracted from Zanibbi et al. (2002)*

structure. This process can be aided by syntactic information to prune segmentations or classifications that may generate wrong expressions from the syntactical point of view. Some techniques proposed to integrate the three processes are discussed in the next section.

2.3 Integrated approaches

Integrated approaches for recognition of mathematical expressions generally use string parsing techniques (Álvaro et al., 2011, 2012; Awal et al., 2009, 2010b; MacLean and Labahn, 2013; Yamamoto et al., 2006). In these approaches, mathematical expressions are considered as *objects* generated by a grammar. The main goal of these approaches is to solve the ambiguity in symbol segmentation and recognition using the grammatical structure of the whole expression. As the elements in mathematical expressions do not necessarily follow the linear structure of strings, constraints are introduced to make possible the application of the string parsing methods. However, the constraints and the linear structure of the grammars make difficult to extend them to include new expression structures.

In Yamamoto et al. (2006), a production rule defines a structural relation between symbols or a composition of two sets of strokes to form a symbol. A probability value is associated with each rule application. Then, the search for the most probable expression that can be generated by the grammar is done through a CYK algorithm. By keeping the most probable subexpressions in each CYK iteration, the algorithm is able to obtain several possible results. A main drawback of this method is the assumption that symbols are composed only of consecutive (in time) strokes.

In Awal et al. (2009, 2012), the recognition problem is formulated as a joint optimization of symbol segmentation, recognition, and structural analysis, considering grammar restrictions. A symbol

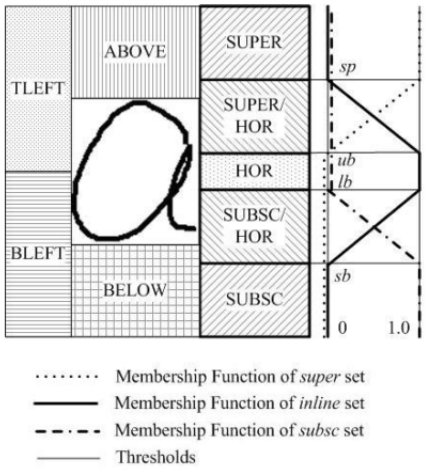


Figure 2.6: *Fuzzy regions associated with a symbol. Intersection of the fuzzy regions define as ambiguity zones.*

hypotheses generator defines stroke groups that might represent symbols (symbol hypotheses). A symbol classifier is used to assign a recognition cost to each symbol hypothesis. Then, a 2D parsing (CYK adaptation) is applied over the symbol hypotheses to form several possible mathematical interpretations. The grammar proposed in their approach consists of a set of one dimensional rules on both vertical and horizontal axes. Those rules are applied until elementary symbols are reached, resulting in a top-down approach. The relation between subexpressions are calculated using Gaussian models derived from training data. A baseline coordinate y and hypothesis height h of each symbol hypothesis is used to perform contextual evaluation. Those features are calculated differently, depending on the type of symbol (a symbol a has different baseline coordinate than a symbol y). Using the baseline coordinates and the hypotheses height, they define a normalized position (dy) and size difference (dh) for each element of a relation (for example, *subscript* has two elements/subexpressions, base and power). Two gaussian models, one for dh and other for dy, are then constructed for each element of a relation. For each relation R, 2N Gaussian models are constructed, where N is the number of elements of the relation, that can be 2 (as in a *subscript* relation), 3 (as in a binary operator) or 4 (as in an integration $\int_a^b c$).

In MacLean and Labahn (2013), the authors propose a *fuzzy relational context free grammar (fuzzy r-CFG)* and a top-down algorithm to parse mathematical expressions. The grammar defines production rules of the form: $A \xrightarrow{r} A_1 A_2 \dots A_k$, where r indicates a relation between adjacent elements of the RHS of the rule. For example, an integration is defined with two rules:

$$[INTEGRAL] \overset{\rightarrow}{\rightarrow} [ILIMITS][EXPR]d[VAR],$$

$$[ILIMITS] \overset{\downarrow}{\rightarrow} [EXPR] \int [EXPR],$$

where the arrows over the symbol $\rightarrow$ indicate an horizontal relation in the first rule and vertical relation in the second. Figure 2.7 shows a simple grammar example of MacLean and Labahn (2013).

The parsing technique of MacLean and Labahn (2013) is based on the Unger's algorithm (Unger, 1968). The algorithm consists on recursively calculating stroke partitions according to the grammar rules, starting from the rule derived from the start symbol until the rules that generate terminals. An advantage of this algorithm is that it does not constrain the grammar to be in a CNF.

$$
\begin{aligned}
&\texttt{[ST]} \Rightarrow \texttt{[ADD]} \mid \texttt{[TRM]} \\
&\texttt{[ADD]} \overset{\rightarrow}{\Rightarrow} \texttt{[TRM]} + \texttt{[ST]} \\
&\texttt{[TRM]} \Rightarrow \texttt{[MUL]} \mid \texttt{[SUP]} \mid \texttt{[CHR]} \\
&\texttt{[MUL]} \overset{\rightarrow}{\Rightarrow} \texttt{[SUP]}\,\texttt{[TRM]} \mid \texttt{[CHR]}\,\texttt{[TRM]} \\
&\texttt{[SUP]} \overset{\nearrow}{\Rightarrow} \texttt{[CHR]}\,\texttt{[ST]} \\
&\texttt{[CHR]} \Rightarrow \texttt{[VAR]} \mid \texttt{[NUM]} \\
&\texttt{[VAR]} \Rightarrow a \mid b \mid \cdots \mid z \\
&\texttt{[NUM]} \Rightarrow 0 \mid 1 \mid \cdots \mid 9
\end{aligned}
$$

Figure 2.7: *Fuzzy r-CFG example. Extracted from MacLean and Labahn (2013)*

To limit the number of stroke partitions, MacLean and Labahn (2013) associated either a vertical or horizontal ordering constraint with each production rule. These constraints require that all elements of a stroke set associated with an element A_i in the RHS of a rule begin – vertically or horizontally, depending on the rule's order – before any element of a stroke set associated with A_{i+1}. According to the authors, a problem of this constraint is that it might prevent from evaluating some valid partitions. Figure 2.8 shows an expression violating the ordering assumption.

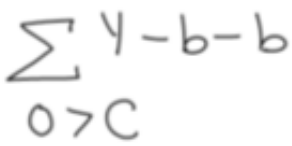


Figure 2.8: *An expression violating the horizontal ordering assumption. The summation and its arguments must start before Y, but the letter C– in the argument of the summation – begins after Y.*

Considering the grammar of Figure 2.7, and an input expression $A^x + b$, the parsing algorithm would work as follows:

1. From each production rule with the initial non-terminal [ST] in the LHS, all right hand sides are evaluated, that is, [ADD] and [TRM].

2. Considering that the rule with LHS [ADD] is associated with a horizontal order, the valid horizontal partitions to that order are:

Table 2.1: *Horizontal partitions*

Part 1	Part 2	Part 3
Ax	+	b
A	x+	b
A	x	+b

Each partition is then evaluated as in step 1, but considering rules that have its corresponding non-terminal ([TRM] for part 1, + for part 2, and [ST] for part 3) in the left hand side,

instead of [ST]. For each partition, if one part fails, then the whole partition fails. The process is similarly applied for [TRM].

To determine the likelihood of applying a production rule, MacLean and Labahn (2013) introduced scores based on fuzzy sets. Fuzzy binary functions determine the degree in which two adjacent elements of the RHS of a rule are compatible with the relation associated with the rule. Then, the compatibility of the application of a rule is given by the multiplication of the compatibility of each adjacent pair of elements of the RHS. For a parse tree result, the compatibility is given by a normalized product of the compatibility scores of its production rules.

The algorithm of MacLean and Labahn (2013) follows a top-down technique because it builds a parse tree from the start symbol (non-terminal) to the leafs. According to Anderson (1968), top-down techniques provide a natural way of hypothesizing the global properties of a configuration at an early stage (when analyzing rules that derive from the start symbol) of the recognition procedure. This allows to avoid generating tree nodes that can not be reached from the start symbol, which are nodes that will not be part any valid parsing tree. In addition, MacLean and Labahn (2013) argue that, in practice, when comparing their top-down implementation with a bottom-up (CYK), they obtained better results, in terms of efficiency, using the first approach.

Álvaro et al. (2012) propose a stochastic context-free grammar and a CYK-based algorithm. The grammar is defined similarly to MacLean and Labahn (2013), but, as a constraint imposed by the CYK algorithm, rules have only up to two elements in the RHS. The approach assumes that symbols are formed only by strokes that intersect each other (in an image rendered from the strokes). To recognize symbols that do not have intersecting strokes, like "i" and "j", the authors add grammar rules to model those symbols. For instance, in the case of "i", they add rules that define a vertical relation between the *body* of "i" and the point. To calculate the likelihood of applying a rule, geometric features were extracted from the bounding boxes of the two stroke groups associated with the two elements of the RHS of the rule. These features were used to train Support Vector Machine classifiers and the output of these classifiers were then expressed as posterior probabilities.

In Celik and Yanikoglu (2011), the authors proposed a recognition framework that uses a graph grammar to model mathematical expressions. Although the technique provides a more flexible model than string grammars (thanks to the graph-based representation that does not assume a linear structure of the elements in a RHS of a rule), the technique was limited to recognize only some constrained structures due to efficiency issues.

The grammar of Celik and Yanikoglu (2011) consists of production rules of the form: $r = (g_r, g_l, C)$ where g_l is a pattern graph, g_r is a single node graph that will replace g_l and C is an applicability predicate. The predicate defines angle and distance constraints that must be satisfied for the application of the rule. A rule r is then applied if a graph g_m, which matches g_l and satisfies, C is found. The patterns g_l are defined as star graphs: a graph that has a central node, and surrounding nodes connected **only** to the central node. Figure 2.9 shows sample rules. A side effect of the constraint imposed in the rules' structure is that it prevents the grammar from modeling relations between subexpressions; as in expression $(a+b)^{(x+y)}$, that contains a superscript relation between the subexpressions $(a+b)$ and $(x+y)$.

Celik and Yanikoglu (2011) assume that symbols are correctly segmented, with alternative labels for each segmented symbol. Once symbols are segmented, an initial graph is created by constructing edges between the symbols. The edges are created considering distances and adjacency constraints,

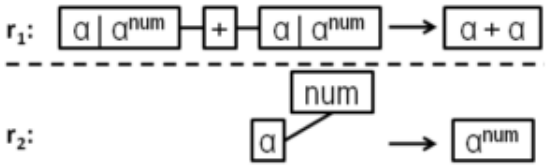


Figure 2.9: *Sample rules of Celik and Yanikoglu (2011), The symbol "|" indicates that any of the symbols at their sides may be a label of the vertex.*

and do not define specific relations, like superscripts or vertical relations. Those relations are defined when the production rules are applied.

The parsing algorithm follows a bottom-up approach: at each iteration, all possible rules are evaluated, and for each applied rule, a new node is created. After evaluating all production rules, new edges between the new nodes are created. In addition, the new nodes inherit edges from their component nodes. Figure 2.10 shows iterations of the parse algorithm.

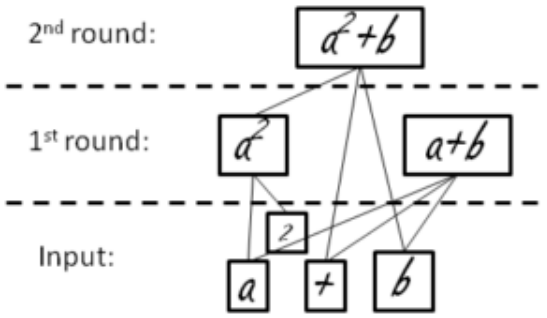


Figure 2.10: *Parse algorithm of Celik and Yanikoglu (2011)*

For experimentation, Celik and Yanikoglu (2011) built a grammar with 17 production rules, as well as additional rules to represent symbols written with non-overlapping strokes, like "=" and "÷". The dataset included up to 30 symbols per expression. Although the grammar was relatively small in terms of structure and symbol classes, the authors reported time out problems.

2.4 Discussion

To the best of our knowledge, until about five years ago there was no large publicly available datasets of online handwritten mathematical expressions. The approaches developed before that (most of the sequential approaches) were then evaluated on small sets of expressions, created by their own authors. This prevented the comparison between more recent approaches, that generally consider an integrated process, and the sequential approaches. However, taking into account the ambiguity issues inherent to handwritten data, integrated approaches seem to be more natural techniques to treat the recognition problem. While the sequential approaches take isolated decisions in the segmentation and classification steps, the integrated methods keep a list of probable segmentations and classifications and determine the best of them considering the whole expression structure.

Grammars provide effective models to integrate contextual information into the recognition process and to formally define the language that can be recognized by a technique. Parsing algorithms allow to restrict the recognition process to interpretations that are valid according to a grammar. For instance, they allow to easily express that if an open parenthesis has been recognized, a closing parentheses should be recognized too. If no strong constraints are imposed to a grammar and its corresponding parsing algorithm, a technique could recognize new structures by adding grammar rules that model those structures.

Most grammars used to represent mathematical expressions are based on grammars to parse strings (Álvaro et al., 2012; Anderson, 1968; Chou, 1989; MacLean and Labahn, 2013; Yamamoto et al., 2006). However, as these grammars were originally designed to represent only horizontal relations between symbols, it is difficult to extend the model to represent languages with multiple relation types. With regard to parsing algorithms, most of them follow a pure bottom-up approach; for instance, different adaptations of the CYK algorithm can be seen in Álvaro et al. (2012); Awal et al. (2009, 2010b); Simistira et al. (2015); Yamamoto et al. (2006).

The disadvantages of pure bottom-up algorithms have been deeply studied in the problem of parsing of linear representations. In that problem, instead of using pure bottom-up algorithms, the parsers usually incorporate a top-down component to avoid the generation of unnecessary nodes (Grune and J.H, 2008). The Earley algorithm is a well-known algorithm that follows this approach. The top-down component of that algorithm uses the grammar rules to determine which nodes are not needed to build a valid parse tree, according to the already parsed elements. The algorithm has shown to be experimentally more efficient than the CYK (Grune and J.H, 2008) and does not constrain the grammar to be in CNF. Developing a 2D parsing algorithm considering the key ideas of the Earley algorithm could improve the efficiency of current CYK parsers of mathematical expressions. Another alternative to the CYK that avoids the generation of unnecessary nodes is the top-down Unger's algorithm (Unger, 1968). When considering parsing two-dimensional data, Unger's algorithm seems to be a more adequate option than Early algorithm. While the first processes the input elements in an arbitrary order (non-directional parsing), the second processes each input element following a specific order (directional parsing), which is possible in the case of strings, but is not (directly) in the case of mathematical expressions.

Graph grammars (Pflatz and A., 1969) provide a natural model to represent mathematical expressions: sentences (mathematical expressions) can be represented as labeled graphs, where vertices represent (terminal) symbols and edges represent relations among symbols. As arbitrary relations can be expressed as edges, the model provides a flexible representation, so that it is easy to extend a particular grammar to define new structures. On the other hand, if no strong constraints are imposed, a similar parsing technique can be used to recognize different two-dimensional languages, as handwritten chemical expressions and diagrams. However, the general representation of graph grammars comes with a considerable increase on the computational cost of the parse algorithm. This has been shown experimentally in Celik and Yanikoglu (2011), where a small application presented time out failures.

Although graph grammars have been successfully used in a number of small-scale applications, it remains as an open problem to see if they scale up (Flasiński and Jurek, 2014). As a parsing algorithm may introduce restrictions on the grammar, both the algorithm and the grammar should be designed in such a way to obtain an efficient parsing, but without reducing the power of repre-

sentation or generalization of the grammar.

Chapter 3

The proposed recognition framework

This chapter describes the formalization and architecture of the proposed approach for online handwritten mathematical expression recognition.

The proposed approach integrates the three recognition processes – symbol segmentation and recognition and structural analysis– into a graph parsing technique. This technique models mathematical expressions as directed graphs generated by a graph grammar. Given an input set of strokes, the parsing technique then determines a parse tree that represents an interpretation of the input as a mathematical expression graph.

The proposed approach integrates several modules that aim to solve specific aspects of the recognition. To make the explanation clear, this chapter focuses on the integration of the modules and gives only a brief description of each of them. Further chapters give more details of the modules. We will refer to those chapters accordingly. Before we explain the recognition technique itself (Section 3.3), we introduce the graph grammar formalization (Section 3.1), and the parse tree structure (Section 3.2). At the end of the chapter, we discuss the differences between our technique and those previously proposed in the literature (Section 3.4).

3.1 Graph Grammar to model Mathematical Expressions

Graph grammars were proposed as a method to define *languages* whose *sentences* are digraphs (or directed graphs) with symbols (labels) at their vertices (Pflatz and A., 1969). In this Section, we define a context-free graph grammar to generate mathematical expression graphs. To clarify the explanation, we illustrate the general model with a small grammar example.

3.1.1 A context-free graph grammar for mathematical expressions

Similarly to string grammars, a context-free graph grammar is defined by a tuple $M = (N, T, I, R)$, where:

- N is a set of non-terminal symbols (or non-terminals);
- T is a set of terminal symbols (or terminals), such that $N \cap T = \emptyset$;
- I is a graph, called initial or start graph; and
- R is a set of production, or rewriting, rules.

While in string grammars a production rule defines the replacement of a string by another string, in graph grammars a production rule defines the replacement of a graph by another graph. We denote the left hand side (LHS) graph of a rule r as $LHS(r)$, and the right hand side graph of r as $RHS(r)$. A graph grammar is context-free if for each rule r, $LHS(r)$ is a single vertex graph (Pflatz and A., 1969).

Figure 3.1 shows a small context-free graph grammar to model mathematical expressions. The terminals of the grammar correspond to mathematical symbols. The vertices of each graph represent subexpressions, which are labeled with terminal or non-terminal symbols, and the edges represent the spatial relations between the subexpressions.

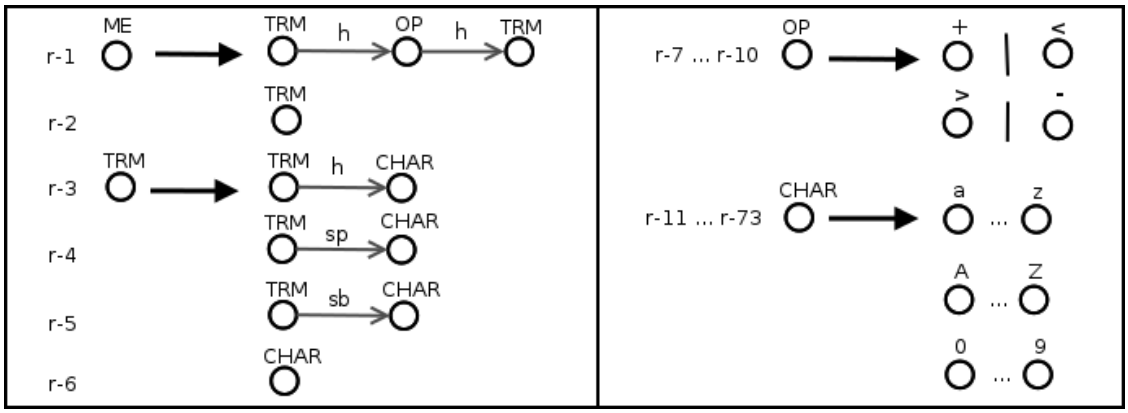


Figure 3.1: *Graph Grammar example. Black arrows separate the replaced and replacing graphs of the rules and gray arrows represent the edges of each graph. The elements of the grammar are: non-terminals N={ME, TRM, OP, CHAR}, terminals T = {+,-, <, >, a, ...,z, A, ..., Z, 0, ..., 9}, initial graph I corresponds to the left hand side graph of rule r-1, and rules R = {r-1, ..., r-73}.*

As it can be seen in Figure 3.1, vertices and edges of all graphs are labeled. For each rule's graph $G = (V_G, E_G)$, we define its vertex labels by a mapping function α: $V_G \rightarrow N \cup T$, and edge labels by a mapping function β: $E_G \rightarrow SR$, where SR is a set of structural relation labels. In the grammar of Figure 3.1, SR only includes the following labels: *superscript* (sp), *subscript* (sb) and *horizontal* (h). We say that a graph $G' = (V_{G'}, E_{G'})$ is generated by a graph grammar $M = (N, T, I, R)$ if G' can be derived from the initial graph I, by applying a sequence of rewriting rules, and labels of $V_{G'}$ all belong to T.

In string grammars, rewriting rules of the form $A \rightarrow B$ are used to define the replacement of a string A by another string B. This replacement is defined by a concatenation of substrings: the replacement of B by A in a string of the form $\omega = \alpha A \beta$ (that is, a string that contains A as a substring) produces another string $\omega' = \alpha B \beta$.

As mentioned above, in graph grammars, a rewriting rule defines a replacement of a graph by another. In this case, a *concatenation* becomes an operation that connects the replacing graph with a host graph and the way in which the *concatenation* is done must be specified for each rule. Thus, given a graph A that is a subgraph of a host graph ω, a rule that defines the replacement of A by another graph B must specify how B is attached to the difference $\omega - A$. This attachment could be done, for example, by inserting edges between some vertices of B and others in $\omega - A$. Such specification is called *embedding*.

The embedding specification depends on the language to be generated. To define the embedding rule for our mathematical expressions graph grammar, we introduce the following definitions over

the digraphs of production rules:

- **Baseline.** A baseline of a digraph is a path whose vertices are connected by edges that only have *horizontal (h)* labels (this definition can be seen as a graph version of the baseline definition of Zanibbi et al. (2002)).
- **Nested baseline.** A baseline is nested to a vertex v if the baseline is connected to v by an edge (v, v'), where v' is the first vertex [1] of the baseline.
- **Dominant baseline.** The dominant baseline of a digraph is a baseline that is nested to no vertex.

We then define the following embedding for all rules of a grammar of mathematical expression. Let $r = X_0 \rightarrow X_1 \dots X_k$ be a rule, where X_0 denotes the vertex of $LSH(r)$ and $X_1 \dots X_k$ denote the vertices of $RHS(r)$; and $\omega = (V_\omega, E_\omega)$ be a graph that contains a vertex X_0. The embedding function of r, denoted as ϵ, defines the edges that connect ω to $RHS(r) - X_0$ as:

$$\begin{aligned}\epsilon = &\{(Y, X_i), \forall i = 1 \dots, k | (Y, X_0) \in E_\omega \text{ and } X_i \text{ is the first vertex} \\ &\text{of the dominant baseline of } RHS(r)\} \cup \\ &\{(X_i, Y), \forall i = 1 \dots, k | (X_0, Y) \in E_\omega \text{ and } X_i \text{ is the last vertex} \\ &\text{of the dominant baseline of } RHS(r)\}\end{aligned} \tag{3.1}$$

Figure 3.2 illustrates the generation of a graph that represents the mathematical expression $a^b + c^d$, using the grammar of Figure 3.1 and the proposed embedding. The rule r-1 is first applied to transform the initial graph into a graph with three vertices linked by an *horizontal* relation. The rule r-4 is then applied to transform the left-most vertex of the new three vertices into a graph that is composed of two vertices linked by a *superscript* relation. Similarly, the graph transformations are applied to all the vertices labeled with non-terminals, until obtaining a graph with vertices labeled only with terminals.

Using the embedding of Equation 3.1, the graph grammar model allows to represent a wide range of mathematical expressions. For the experimental part of this work, we built a graph grammar with 205 production rules and the proposed embedding (as described in Chapter 7). The grammar covers the mathematical expressions language defined on the CROHME-2014 dataset (Mouchère et al., 2014). Figure 3.3 shows some mathematical expressions (and their graphs) that may be generated using the proposed grammar.

It is important to note that we do not assume specific forms for the RHS graphs of the rules, as in Celik and Yanikoglu (2011), neither constrain the grammar to be in the Chomsky Normal Form, as in Álvaro et al. (2012); Simistira et al. (2015). A main assumption of our model is that there exists a dominant baseline in each graph, which is not a strong constraint and allows to model a variety of two-dimensional languages as flowcharts, music scripts and chemical expressions.

3.1.2 Comparison with other mathematical expression grammars

Graph grammars provide a more expressive way to represent mathematical expressions than models used in previous approaches. As mentioned in Chapter 2, most approaches for recognition

[1] The first vertex of the baseline path.

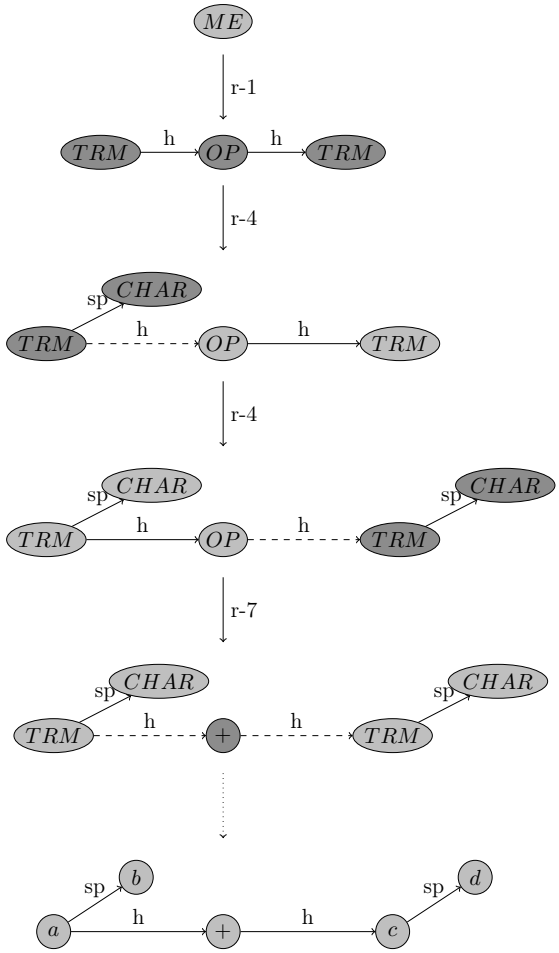


Figure 3.2: *Graph generation through rule applications. At each rule application, the replaced graph nodes are depicted in dark gray nodes and the embedding edges with dashed arrows. Rule applications after r-7 are omitted (dotted arrow).*

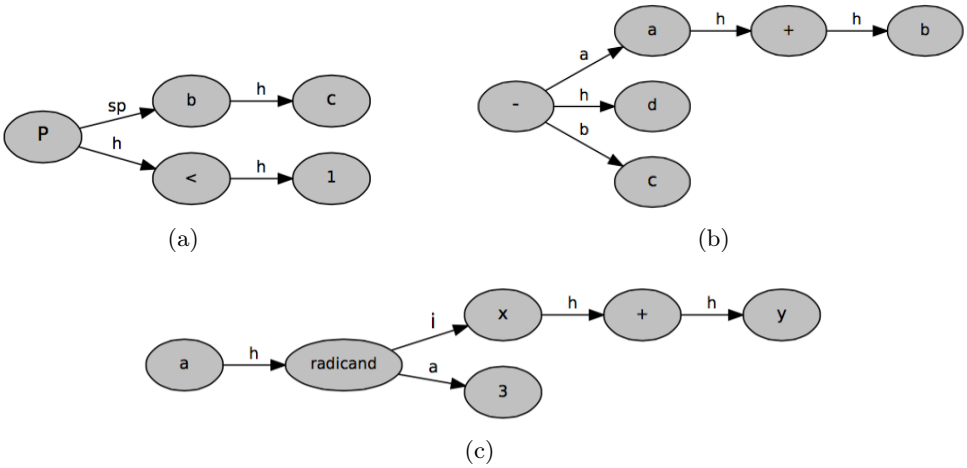


Figure 3.3: *Graphs generated with the grammar implemented for experimentation. The graphs correspond to expessions (a) $P^{bc} < 1$, (b) $\frac{a+b}{c}d$ and (c) $a\sqrt[3]{x+y}$. Vertices and edges are labeled with their symbol and relation classes , respectively.*

of mathematical expressions extend the grammars used for strings. Figure 3.4 shows a comparison of grammar rules generally used by other approaches (as in Álvaro et al. (2012); MacLean and Labahn (2013)) and the corresponding rule using our approach. The grammars used by other approaches adapt grammars used to parse strings and define rules of the form: $X \xrightarrow{r} X_1 X_2 \dots X_k$, where r indicates the relation between consecutive elements in the RHS [2]. As the model defines a unique relation type between elements of the RHS, rules that include different relation types must be split into several rules (as illustrated in Figure 3.4). Furthermore, when a CYK parse algorithm is used (as in Álvaro et al. (2012); Simistira et al. (2015); Yamamoto et al. (2006)), the grammar needs to be transformed to a Chomsky Normal Form, which limits the number of elements in the RHS of a rule to two. As a result, grammar rules with more than two elements in the RHS must be split, incrementing the total number of rules. These restrictions make difficult to extend string grammars to model two-dimensional languages.

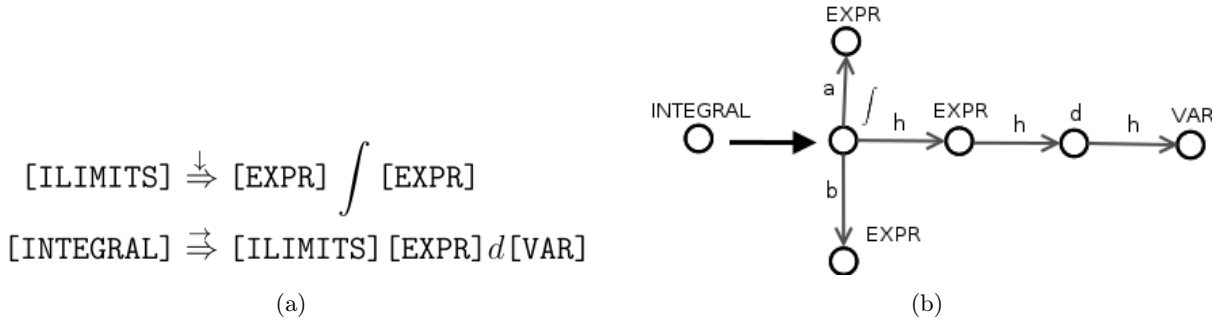


Figure 3.4: *Integration rule example used in MacLean and Labahn (2013) (a) and its corresponding representation using graph grammar (b).*

The graph-based representation of graph grammars provides flexibility to represent a variety of two-dimensional languages. This powerful representation has inspired applications in syntactic pattern recognition problems, such as image or diagram interpretation (Bunke, 1982; Han and Zhu, 2009; Lin et al., 2009), recognition of tessellated image regions (Sanchez and Llados, 2001) and recognition of mathematical expressions (Celik and Yanikoglu, 2011; Lavirotte and Pottier, 1997).

The flexibility of graph grammars and its powerful representativeness have, however, a downside, which is a high computational cost of the parsing. The cost is mainly due to the search of all matchings between a set of input strokes and the RHS graph of a rule (more details on this are given in Section 6.1). The more complex the RHS graph of a rule (in terms of number of edges and vertices and allowed structures), the more expensive the search cost. The creation of graph grammar models that combine generality with efficient parsing techniques is an interesting and important problem in syntactic pattern recognition (Flasiński and Jurek, 2014).

To improve the parsing efficiency, previous works based on graph grammars introduced some constraints or defined relatively simple grammars. However, these constraints make difficult the application of the same technique to model different languages. In recognition of mathematical expressions, for instance, Celik and Yanikoglu (2011) proposed graph grammars to model mathematical expressions and restricted the production rules to have only star like graphs. In these graphs, a central vertex must be labeled with a terminal symbol and other vertices (if exist) are only adjacent to this central vertex. The constraint aims to improve the search for rules to be ap-

[2] In this context, X_i, X_j are consecutive if $j = i - 1$ or $j = i + 1$.

plied, by starting the search with strokes labeled with the central vertex symbol and pruning other possibilities. Figure 3.5 shows examples of those rules. A side effect of this constraint is that it prevents the grammar from modeling relations between subexpressions, as in expression $(a+b)^{(x+y)}$, that contains a superscript relation between the subexpressions $(a+b)$ and $(x+y)$. Even with a small grammar (17 production rules), the authors of Celik and Yanikoglu (2011) reported time out problems.

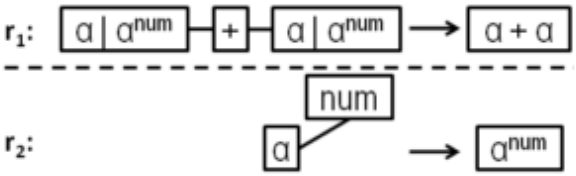


Figure 3.5: *Graph grammar rule examples proposed in Celik and Yanikoglu (2011). Each rule defines the replacement of a single vertex graph in the RHS with a graph that has a central node, and surrounding nodes connected* ***only*** *to the central node (star graph). The symbol "|" indicates that the symbol at its left or right may be a label of the vertex.*

An approach for recognition of printed mathematical expressions using graph grammars was proposed in Lavirotte and Pottier (1997). In that approach, during the parsing step, only one production rule is applied, to avoid evaluation of multiple interpretations (only one result is generated). When dealing with printed expressions, considering only one interpretation is often enough to find the right result. Nonetheless, when dealing with recognizing handwritten expressions, ambiguous cases usually require the evaluation of multiple interpretations to determine the right result.

3.2 Parse tree and forest

Given a graph generated by a graph grammar, a parse tree defines which and how the rewriting rules are used to generate the graph. For instance, Figure 3.6 shows a parse tree for a graph that represents the expression $P^b < 1$, using the grammar of Figure 3.1. Each rectangle contains a graph generated by the application of a production rule, starting from the initial graph of the grammar.

Different interpretations of a mathematical expression correspond to different parse trees. To deal with ambiguous expressions, the proposed parsing technique generates multiple interpretations of a same input. However, enumerating all the trees corresponding to the interpretations may be infeasible. Parse forest are used to efficiently store all the generated parse trees. This technique has been used by several approaches for parsing of strings (Grune and J.H, 2008).

Figure 3.7 illustrates an ambiguous mathematical expression and a parse forest calculated using the graph grammar of Figure 3.1. For easy visualization, strokes are depicted inside each grammar vertex. At the top of the structure, the set of strokes is partitioned according to the rules derived by the non-terminal **ME** (the start symbol of the grammar). A partition of a set of strokes according to a rule r, consists of a mapping of the strokes to the vertices of the graph $RHS(r)$. We call a graph that defines a partition **instantiated graph**. In the case of rules for **ME**, two partition candidates are represented, one using rule r-1 and the other using rule r-2. New partitions are recursively stored for each vertex of each instantiated graph, until a vertex labeled with a terminal symbol is found. The parse forest example stores a total of eight parse trees, where a tree can be generated

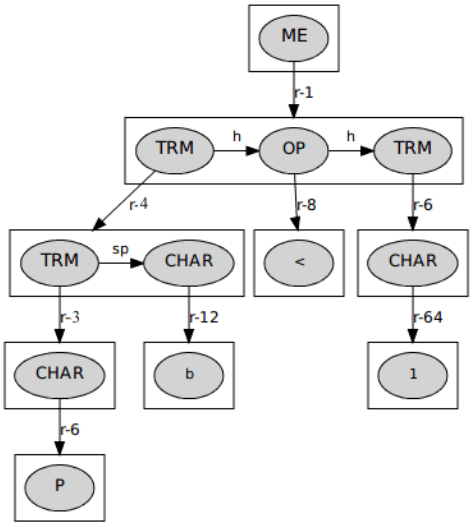


Figure 3.6: *Parse tree example for the expression $P^b < 1$, considering the grammar of Figure 3.1. The root of the tree (inside the top-most rectangle) corresponds to the initial graph of the grammar. The applied rules are depicted as arrows from a vertex (or single vertex graph) to a rectangle that contains the transformed graph. Each rule is identified using the codes defined in the grammar.*

by following the rules from the top node to nodes that contain terminal symbols, and by choosing one rule out of each instantiated graph node.

3.3 The proposed approach

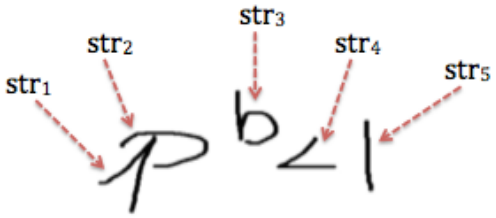
The proposed recognition technique is illustrated in figure 3.8. The input is a sequence of strokes and the output is a parse tree that is an interpretation of the input as a mathematical expression graph. The approach is composed of two main components: the hypotheses graph generator and the graph parser.

3.3.1 Hypotheses graph generator

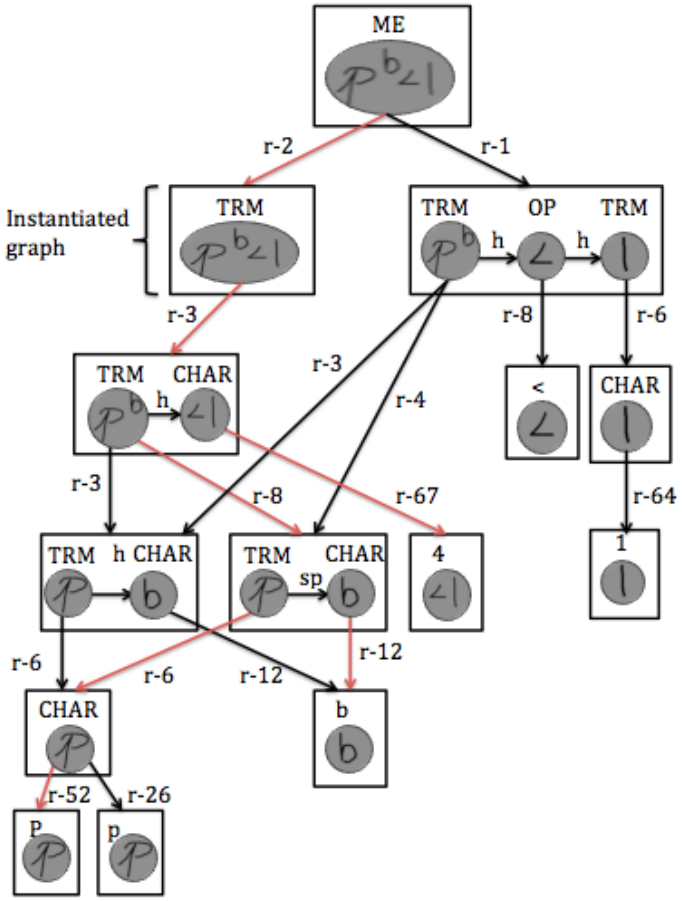
The hypothesis graph generator aims to identify all possible groups of strokes that might represent symbols and relations among them. Both the identified symbol and relation hypotheses are stored in a hypothesis graph.

Formally, a symbol hypothesis is a set of strokes that can be interpreted as a terminal symbol. Each symbol hypothesis has a label set as an attribute. The label set is defined by a function $\gamma : SH \rightarrow P(T)$, where SH is a set of symbol hypotheses, and $P(T)$ is the power set of T (the set of terminal symbols of a predefined grammar). A hypotheses graph is a digraph $H = (V_H, E_H)$, where V_H is a set of symbol hypotheses and E_H is a set of relation hypotheses, defined over pairs of *compatible* symbol hypotheses [3]. As in the case of symbol hypotheses, a label set is assigned to each relation hypothesis. In this case, the label set is defined by a function $\delta : E \rightarrow P(SR)$, where SR is a set of relation labels (defined by a grammar), and $P(SR)$ is the power set of SR. Figure 3.9 shows a handwritten mathematical expression example and a hypotheses graph calculated from it.

[3] Two symbol hypotheses sh_i, sh_j are compatible if $sh_i \cap sh_j = \emptyset$.



(a)



(b)

Figure 3.7: *A parse parse forest (b) representing multiple interpretations of a mathematical expression (a). Labels on arrows indicate the grammar rules. Red arrows represent a parse tree that corresponds to the interpretation "P^b4".*

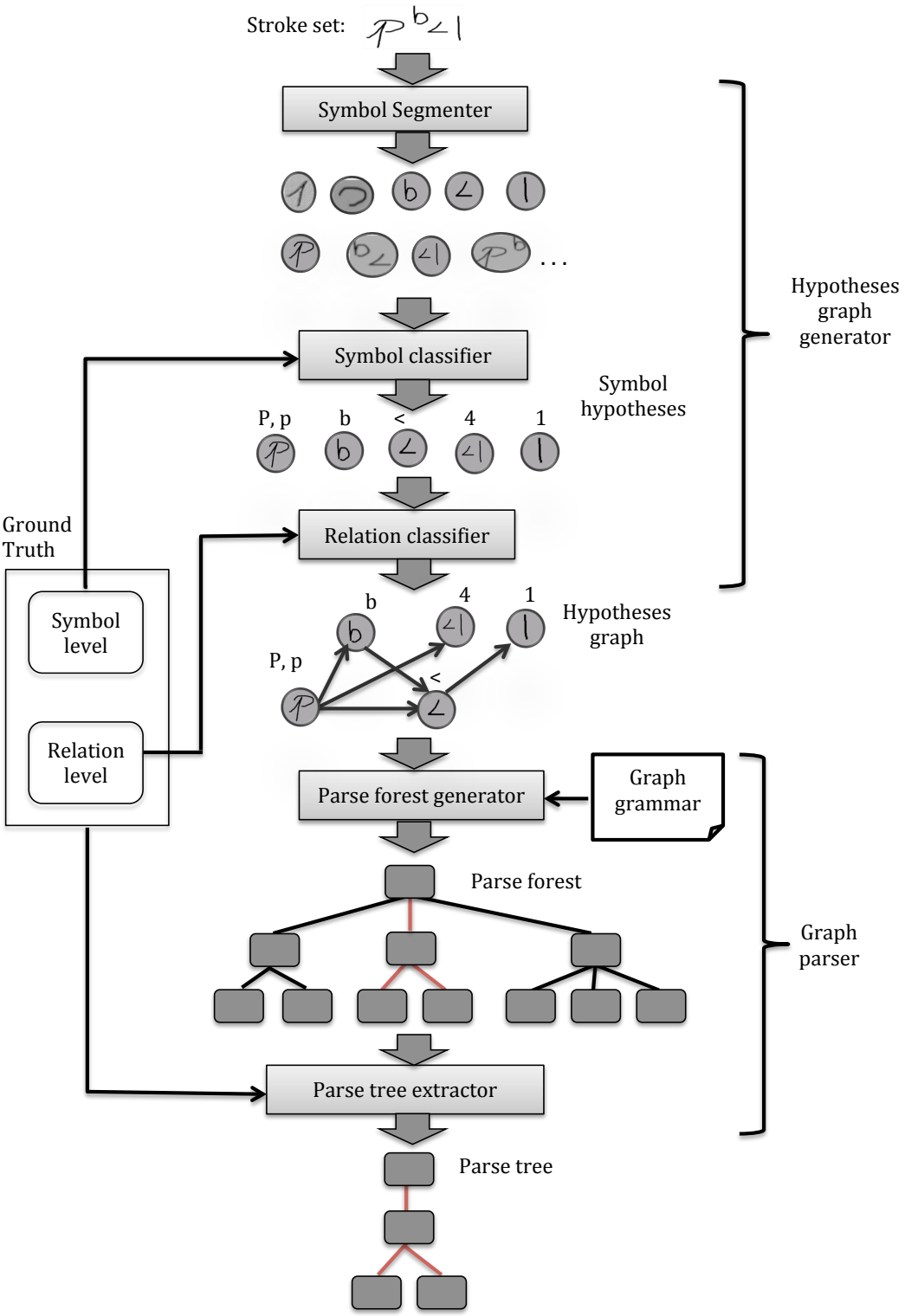


Figure 3.8: *The recognition scheme of the technique proposed in this thesis. Gray rectangles represent the main modules that compose the approach.*

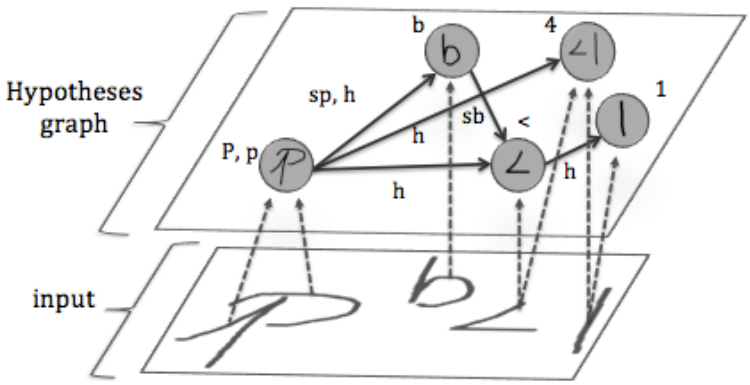


Figure 3.9: *Hypotheses graph example. Vertices represent symbol hypotheses and edges represent relations between symbols. The labels associated to symbols and relations indicate their most likely interpretations.*

Symbol hypotheses generation

Symbol hypotheses generation involves (1) the generation of stroke groups and (2) the classification of each stroke group as a true symbol hypothesis, with its corresponding label set, or as a false symbol hypothesis, that is, a group of strokes that do not represent any symbol.

Given a set of n strokes, $str = \{str_1, \ldots, str_n\}$, the total number of groups of strokes that can be generated from str is equivalent to the number of non empty subsets of str, that is, $2^n - 1$. As evaluating all possible groups is infeasible, several constraints have been introduced in the literature; for example, limiting the number of strokes (between 4 or 5) that form a symbol (Awal et al., 2012; Matsakis, 1999), considering only groups of consecutive (in time order) strokes (Huang and Kechadi, 2007; Lehmberg et al., 1996), and considering only groups of intersecting strokes (Tapia, 2004). In this work, the stroke groups are generated by grouping strokes that are close enough, considering a graph-based distance, but not time order neither intersection constraints are considered. This technique is explained in Section 4.1.

Once the stroke groups are computed, each of them is processed to assign it a label set or reject it. To do that, we propose new symbol features, combine them with state of the art features and train neural network classifiers. The classifiers are trained over a mathematical expression recognition scenario, where groups of strokes that do not represent actual symbols are also considered. This approach differs from previous works (Álvaro and Zanibbi, 2013; MacLean and Labahn, 2013; Tapia, 2004; Zanibbi et al., 2002) that extracted only the *true* symbols and trained classifiers on those isolated symbols. The proposed features and the training scheme is detailed in Chapter 4. The proposed features take advantage of contextual information to improve the rejection of false symbol hypotheses, keeping state of the art recognition rates of true symbol hypotheses. The improvement in rejection of false hypotheses is translated into an improvement in the parsing process (less irrelevant stroke groups are evaluated during the parsing).

The neural network classifiers described in Chapter 4 calculate, for each symbol class c, the likelihood of a stroke group representing c. In addition, the classifiers include a special class, called *junk*, to measure the likelihood of the stroke group be a false symbol hypotheses. As mentioned above, a label set (that may include different symbol classes) is assigned to each symbol hypothesis. This configuration aims to capture all likely interpretations of ambiguous symbols. The calculation of a label set is based on scores given by a classifier: given a symbol hypothesis sh and a set of labels $(l_1, \ldots, l_m)$, sorted in descending order by their likelihood scores $score(l_i)$, for $i = 1, \ldots, m$,

a set of k labels is assigned to sh, such that:

$$k = \arg\min_{k} \sum_{i=1}^{k} score(l_i) > tr, \tag{3.2}$$

where tr is a threshold calculated experimentally.

The threshold tr defines a minimum confidence value to select a label set and to reject a stroke group. A stroke group is rejected if its $junk$ label score is the biggest one and it is greater than the threshold.

Relation hypotheses generation

To calculate relation hypotheses, we built a Neural Network classifier that uses our proposed contextual relation features, combined with geometric features proposed in Álvaro and Zanibbi (2013). The feature set, classifiers and the training scheme is detailed in Chapter 5.

Given a pair of symbol hypotheses (sh_i, sh_j), for each relation class c, the relation classifier calculates a likelihood score for the existence of a relation c between sh_i and sh_j. As in the case of the symbol hypothesis classifier, a $junk$ class is included to score the likelihood of no relation between the symbols. When calculating the relations set, we evaluate all pairs of compatible symbol hypotheses, that is, all pairs (sh_i, sh_j) such that, $sh_i \cap sh_j = \emptyset$.

As in the case of symbol hypotheses, we selected the label sets or rejected relations using their corresponding classifier scores, according to equation 3.2.

3.3.2 Graph Parser

The graph parser uses a hypotheses graph and a graph grammar to generate a parse tree that represents an interpretation of the input strokes as a mathematical expression. As explained in Section 3.3.1, to cope with ambiguity, a hypotheses graph may include multiple interpretations for both symbols and relations. The parsing technique, evaluates the different symbol and relation classes to build a parse forest. This parse forest then stores all possible mathematical expression interpretations that can be derived by combining the symbol and relation interpretations. After that, a best tree, according to a ranking function, is extracted from the parse forest and returned as a result. The selection of the best tree after several trees have been generated aims to select a most probable interpretation by considering the whole structure of the expressions.

Given an input set of strokes str, the parsing technique assigns a score $cost(t, str)$ to each $t \in T(str)$, where $T(str)$ is a parse forest calculated from str. The parsing problem then is a search for a tree $t_{best}(str)$, such that:

$$t_{best}(str) = \arg\min_{t \in T(str)} cost(t, str) \tag{3.3}$$

To build a parse forest, we propose an algorithm based on the Unger's algorithm to parse strings (Unger, 1968). The algorithm is considered a top-down approach because it parses the input by calculating recursive partitions of it, starting from the rules derived by the start graph until obtaining stroke groups that represent terminal symbols. In other words, the parse forest is built from the top vertex (root) to the bottom vertices.

A partition of a stroke set $str = \{str_1, \ldots, str_n\}$ according to a graph grammar rule r, with $LHS(r) = A$ and $RHS(r) = B$, consists on calculating a matching between the strokes str and the vertices of $B = (V_B, E_B)$. Without any constraint, this problem is exponential in the number of strokes, that is, $O(|V_B|^n)$.

The above analysis suggests that evaluating all possible partitions is infeasible. In order to prune non valid partitions, we consider only stroke partitions that can be derived from subgraphs of the hypotheses graph. For instance, considering the hypotheses graph of Figure 3.9, no rule matches the strokes of hypotheses P and 1 to any vertex because they do not form a connected subgraph. This approach makes the search space of the parsing algorithm to be defined by the hypotheses graph. As the hypotheses graph is calculated using classifiers optimized over a training data, the parsing algorithm can be extended to parse other two-dimensional languages by defining a grammar according to the language and re-training the classifiers. This extension could be difficult if we considered hand-made constraints as those proposed in MacLean and Labahn (2013); Yamamoto et al. (2006); Zanibbi et al. (2002).

Chapter 6 details the parsing algorithms to generate the parse forest and extract the best tree, and the proposed models to calculate scores.

3.4 Discussion

According to Flasiński and Jurek (2014), constructing a parsing algorithm for graph languages is a much more difficult task than for string languages because:

1. A graph structure is unordered by its nature, whereas a linear order is defined in a string structure. During parsing, succeeding pieces of an analyzed structure (sub-words in case of strings, subgraphs in case of graphs) are teared off repetitively in order to be matched with predefined structures of right-hand sides of production rules. Determining what is a succeeding piece in the structure is easy when there is an ordering among the pieces. In the absence of an ordering, this is equivalent to the problem of subgraph isomorphism, which is NP-complete.

2. For string grammars, we know how to embed the right hand-side of a rule in a structure transformed during the application of the rule (it is just a concatenation of strings). However, for graph grammars, we have to explicitly specify how to embed the right-hand side graph in the host graph (embedding). The embedding allows one to modify a derived graph structure. On the other hand, it acts at the border between both sides of the rule and their context, i.e. its behavior is *context sensitive-like*.

The arbitrary structures represented by graphs and the possibility of embedding graphs according to their application give graph grammars a big representation power. Nevertheless, this feature also makes the grammar parsing problem NP-complete for classes of graph grammars interesting from the application point of view, as shown in Turan (1982).

Our mathematical expression recognition technique has three main differences in relation to previous approaches:

- Symbol and relation classification methods of previous approaches are based on features extracted only from isolated symbols and relations. To filter false symbols and relations, previous

techniques introduced constraints that make difficult their application to other data. In our work, we propose contextual features extracted from a neighborhood of the evaluated symbols and relations. We then train classifiers with both real and false hypotheses (symbol and relations) so that classifiers learn to filter false hypotheses. The trained classifiers are able to effectively filter false hypotheses. This filtering technique does not assume specific symbol classes or relations and then may be applied to objects of other two-dimensional languages.

- The proposed graph grammar-based approach provides a more general recognition framework than approaches based on string grammars (e.g. Álvaro et al., 2012; Simistira et al., 2015; Yamamoto et al., 2006). String grammars have a limited representativeness due to the assumption of a unique type of relation and are usually constrained to be in a CNF. The proposed graph grammar model allows to represent arbitrary structures through graphs. The proposed parsing algorithm is based on a search for isomorphisms between structures of the grammar and structures of a hypotheses graph. This algorithm does not introduce constraints into the grammar (as the CYK, that requires the grammar to be in a CNF). The general formulation of the grammar and the parsing algorithm then facilitate the application of the technique to other two-dimensional languages.

- Previous works that proposed graph grammars for mathematical expression recognition reduced the parsing complexity by defining specific structures for production rules and limited the number of rules (Celik and Yanikoglu, 2011), or considered only a unique interpretation of the input (Lavirotte and Pottier, 1997). In our approach, we reduce the parsing complexity by pruning not likely interpretations, according to classifiers optimized over training data. The extension of this technique to other data is then possible by retraining the classifiers. Our experimental results show that the proposed parsing technique is efficient enough to be used in an online context.

Chapter 4

Symbol hypothesis classification

The mathematical expressions recognition method proposed in this thesis starts with the identification of groups of strokes that may represent symbols. This process involves two tasks: (1) generation of stroke groups and (2) classification of a each stroke group as a true symbol, with its corresponding symbol class, or as a false symbol.

Those tasks must be performed at some part of any mathematical expression recognition method. The importance of this problem has recently been emphasized in a competition that considers a symbol classification problem that includes false symbol instances (Mouchère et al., 2014).

In this chapter, we describe our stroke grouping method, the feature set used to classify each stroke group, and compare our results with those of Mouchère et al. (2014).

4.1 Generation of stroke groups

Given a set of n strokes, $str = \{str_1, \ldots, str_n\}$, the total number of groups of strokes that can be generated from str is equivalent to the number of non empty subsets of str, that is, $2^n - 1$. As evaluating all possible groups is infeasible, several constraints have been introduced in the literature, for example, limiting the number of strokes (between 4 or 5) that form a symbol (Awal et al., 2012; Matsakis, 1999), considering only groups of consecutive (in time order) strokes (Huang and Kechadi, 2007; Lehmberg et al., 1996), and considering only groups of intersecting strokes Tapia (2004).

Our strokes grouping method assumes that symbols are composed of at most k strokes. Thus, we determine symbol hypotheses by calculating groups of strokes that are close enough, considering a k-nearest neighbor graph. To do that, given a stroke set $str = \{str_1, \ldots, str_n\}$, we build a k-nearest neighbor graph from a complete graph that has the n strokes as vertices and the euclidean distances between the bounding box centers of strokes as edge weights. For each stroke, we then generate all combinations of the stroke with its adjacent strokes given by the k-nearest neighbor graph. The resulting stroke groups are filtered to avoid repeated stroke combinations, and, after that, each combination goes to the second step, that determines if the stroke group is a true symbol.

It is important to note that the stroke grouping method assumes that each stroke belongs to only one symbol. That is, an expression like "$\cos\theta$" is assumed to have more than one stroke (it should be written with at least two strokes, one for each symbol). This assumption is not a strong constraint, as handwritten mathematical expressions are generally printed rather than cursive. The constraint has been used in most of the previous approaches, like Álvaro et al. (2012); MacLean and Labahn (2013); Matsakis (1999); Tapia (2004); Yamamoto et al. (2006). On the other hand, preprocessing

techniques could be applied to divide strokes that belong to several symbols, if an application requires to deal with those cases.

4.2 Symbol hypothesis classification

This process aims to classify a stroke group as a true mathematical symbol (with its corresponding label) or as a false symbol, that is, a stroke group that does not represent any mathematical symbol. To identify instances of stroke groups that do not represent symbols, we introduce an additional class called *junk*. In short, the classification problem has as input a set of strokes and as output a mathematical symbol label or *junk*. It is important to note that the *junk* class does not have a specific shape or configuration, but has an infinite number of configurations.

While most works focus on using feature sets previously proposed for isolated symbol classification, and evaluate them on different kinds of classifiers (Álvaro et al., 2014; Tapia, 2004), we focus on the development and evaluation of histogram based features that, in addition to the symbol features, capture contextual information. The goal of this work is to generate features that capture the raw shape of the symbols and their neighborhood and let Neural Networks to *learn* the patterns.

4.2.1 Preprocessing

When considering online data, noise appears in the form of different sampling scales (for example, expressions collected through small devices may have different patterns from those collected through electronic boards), velocity, and abrupt changes of coordinates. To reduce the impact of noise, we apply smoothing and resampling processes (in that order) to the raw data.

To reduce the impact of abrupt trajectory changes, each stroke coordinate, except the first and the last one, is recalculated as the weighted sum of the coordinate itself, its predecessor and successor. Formally, a smoothed coordinate of $p_i = (x_i, y_i)$ is another coordinate $p'_i = (x'_i, y'_j)$ such that:

$$\begin{aligned} x'_i &= x_{i-1} * 0.25 + x_i * 0.5 + x_{i+1} * 0.25 \\ y'_i &= y_{i-1} * 0.25 + y_i * 0.5 + y_{i+1} * 0.25 \end{aligned} \tag{4.1}$$

Each symbol hypothesis was resampled by replacing all its points, except the first and the last of each stroke, with points that are equally distributed along the writing trajectory (that is, almost all the consecutive points of the resampled symbol, considering time order, have the same distance). As symbols are sampled over different scales, we define the distance between consecutive points as a fraction (1/40) of the largest dimension (between width and height) of the symbol. This resampling method is based on Delaye and Anquetil (2013).

Figure 4.1 shows some raw symbol samples and their corresponding preprocessing results.

4.2.2 Online symbol features

Online features aim to capture patterns related to the writing process, as the starting and end points of a symbol, angles between consecutive points and direction of the trajectory.

Our online feature set is composed of shape context-based (Belongie et al., 2002) features and a pair of stroke related features.

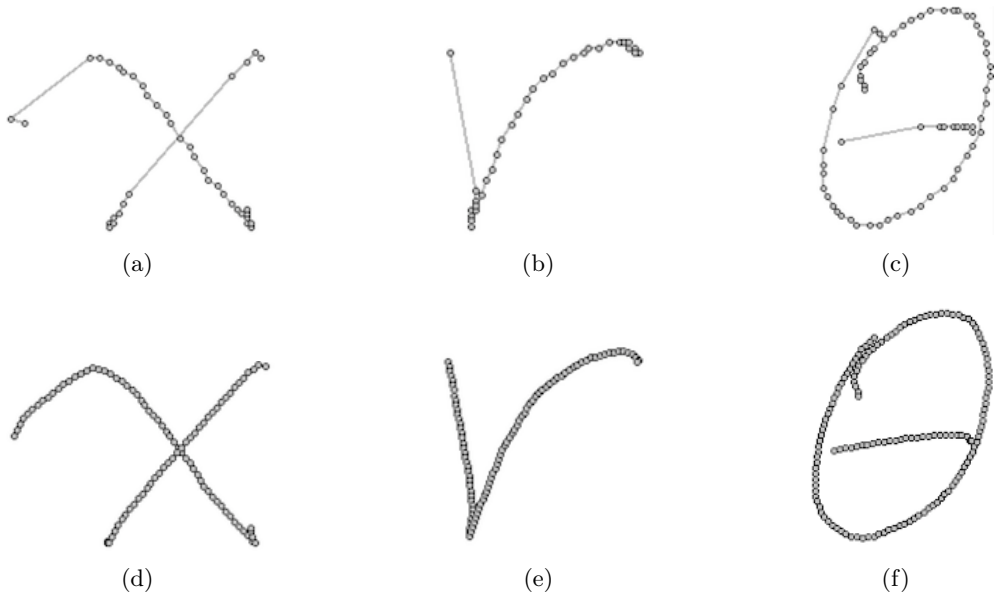


Figure 4.1: *(a), (b), and (c) show raw symbol samples and (d), (e) and (f) their corresponding preprocessed results.*

Shape Context

Shape context was proposed as a shape descriptor in Belongie et al. (2002) and has been applied in several offline symbol recognition problems with outstanding results. Applications include recognition of handwritten digits and 3D objects Belongie et al. (2002), handwritten Tamil scripts with 156 symbol classes Prasanth et al. (2007) and leaf image classification (220 leaf classes) Wang et al. (2011).

Given a set of points $P = \{p_1, p_2, \ldots, p_n\}$, the shape context of a point p_i in P is a log polar histogram that represents the distribution of the remaining points relative to p_i (Belongie et al., 2002). Figure 4.2 illustrates the shape context calculation for two points of a symbol. In that example, the surrounding region of a point is divided into 8 angular bins and 3 radial bins (giving a total of 24 log polar bins). A shape context consists of an histogram that stores the number of points placed in each bin.

Given that a shape context histogram is defined over a log polar space, the shape context relative to a point p_i is more sensitive to nearby points (local features) than points that are far away from p_i (global features) Belongie et al. (2002).

Fuzzy shape context

The shape context descriptor was originally defined over crisp bins, that is, it considers that a point belongs only to one bin.

During the sampling process or due to handwritten variability, small changes in sampled points may be generated. These changes may affect points falling near to the limits of bins: small displacements near to the end of a bin may change the total of number of points in that bin. To reduce the effect of those small displacements on points, we modify the shape context definition by considering bins as fuzzy sets, as shown in figure 4.3.

Let a shape context histogram be denoted $H = \{Bin_{ij}\}$, where i indexes the bins relative to

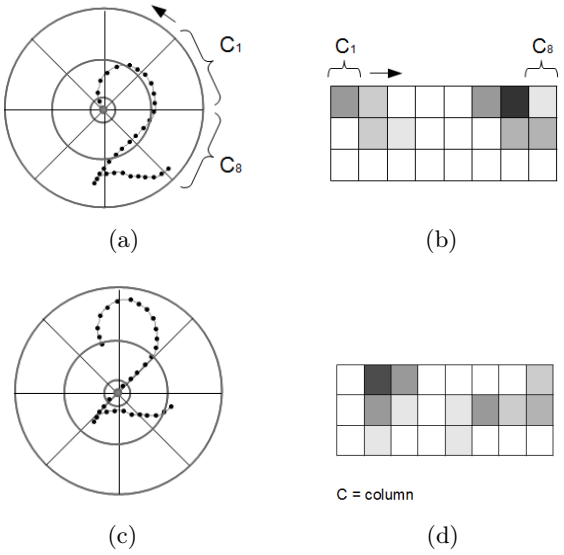


Figure 4.2: *Shape context of two points of a symbol "2": (a) and (c) show the sampled points and the log polar histogram bins used to calculate shape context. (b) and (d) show the shape context histogram relative to (a) and (c) respectively; dark cells mean higher values.*

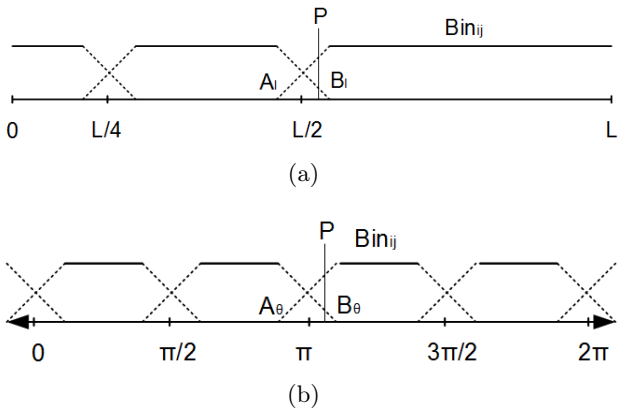


Figure 4.3: *Fuzzy bins in the (a) radial and (b) angular coordinates. The arrows in the angular coordinate represent its circular nature.*

radial coordinate and j indexes bins in the angular coordinate. For a point $P = (P_l,\ P_\theta)$, that lies in a bin Bin_{ij}, the membership value of point P is given by:

$$M_{ij}(P) = \alpha_i * \beta_j \tag{4.2}$$

where α_i and β_j are between 0 and 1 and indicate the confidence value of P lying in Bin_{ij} relative to the radial and angular coordinates, respectively. When P lies in an intercepting region, the values α_i and β_j are calculated according to its position relative to the corresponding coordinate, otherwise they take the value 1. For example, for the point P in figure 4.3, the confidence values will be:

$$\alpha_i = \frac{P_l - A_l}{B_l - A_l} \tag{4.3}$$

$$\beta_j = \frac{P_\theta - A_\theta}{B_\theta - A_\theta} \tag{4.4}$$

Note that with these new membership definitions a point may belong up to four bins, depending on its position. Figure 4.4 shows the four positions that generate different membership cases:

- P position, only bin B_{ij} gets a non-zero value (one),
- R position, bin B_{ij} and a radial-connected bin get non-zero values,
- S position, bin B_{ij} and an angular-connected bin get non-zero values,
- Q position, bin B_{ij} and three connected bins get non-zero values.

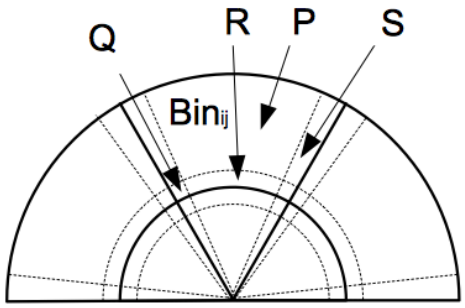


Figure 4.4: *Four different positions in a bin Bin_{ij}. Regions between dotted lines indicate transition areas between bins*

Shape contexts as feature vectores

Most shape context applications use a k–nearest neighbor approach for classification. In this configuration, the training symbols are used as prototypes and the classification of an input symbol consists on selecting a class that has the most similar prototypes to the input, according to a shape context matching measure. Given two sets of points $P = \{p_1,\ \ldots,\ p_n\}$ and $Q = \{q_1,\ \ldots,\ q_n\}$, to calculate a similarity between P and Q, shape context is calculated at each point and a best matching between points of P and Q is calculated using the χ^2 metric and the Hungarian algorithm Belongie et al. (2002). Two drawbacks of this method are the computational load to find the nearest neighbor(s) and store the class prototypes (Hastie et al., 2009).

Regarding to the nearest neighbors search, given a query, classification implies a comparison of the query with all prototypes, to select the most similar prototype(s). By using the Hungarian algorithm, the cost to determine the similarity between an input symbol and a prototype is $O(n^3)$ (where n is the number of sampled points). So, the cost of classifying an input symbol using m prototypes is $O(mn^3)$.

The computational space to store prototypes is directly related to the number of classes and variability of each class – for example, we can expect that prototypes of a symbol "4" presents more variability those of "0", so we could need more prototypes for fours than for zeros Belongie et al. (2002). Although several methods have been proposed to reduce the set of prototypes without reducing accuracy, the scalability of the method is still a weak point (Hastie et al., 2009).

An alternative to the nearest neighbor approach is to use shape contexts as input features for neural networks. Results of Julca-Aguilar et al. (2014b) have shown that a neural network configuration can obtain a performance comparable to the matching based approach, but with a considerable efficiency improvement.

When converting shape context histograms to input vectors, a main concern is the vector size. Considering reported applications (Belongie et al., 2002; Mori et al., 2005; Prasanth et al., 2007), we can see that the number of sampled points and bins for shape context calculation vary around 30 and 40, respectively. Using such parameters, the number of input features for neural networks could reach values of 1200 features.

To reduce the number of features, we maintain the rate of sampled points, but extract shape contexts of only a subset of points. For instance, with the parameters mentioned above, we could calculate only the shape context of points at positions 1, 15 and 30, and get a total of 120 features (3 shape contexts, each containing 40 bins). This method is based on the fact that shape contexts of near points of a same symbol may be similar, thus, it allows us to reduce redundancy on features. Figure 4.5 illustrates the shape context-neural networks configuration.

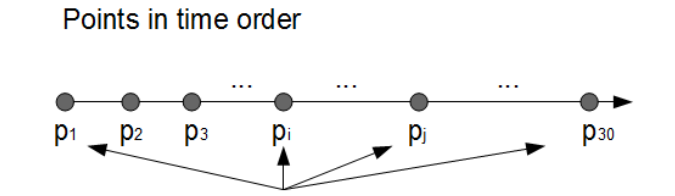
Number of strokes and stroke jumps

The number of strokes of a symbol introduces important information that may not be captured by shape based features. Symbols are generally written with a fixed number of strokes, for example, a symbol "i" is usually written with two strokes, while "0" is written with one. When dealing with hypotheses classification, a hypotheses composed of a *body* of a symbol "i" may receive high probability of being interpreted as a symbol "i". This may happen because the "point" of the symbol does not represent a substancial part of its shape. However, it can have low probability of being a symbol if we consider its number of strokes (1).

When writing a symbol, one can write part of the symbol, stop to write other symbols, and then back to write the rest of the initial symbol. That process is called a jump. Although some symbols may present jumps, it is a pattern that is more likely to be present in false symbol hypotheses. Thus, we also consider the number of jumps of a symbol hypothesis as an input feature.

4.2.3 Offline symbol features

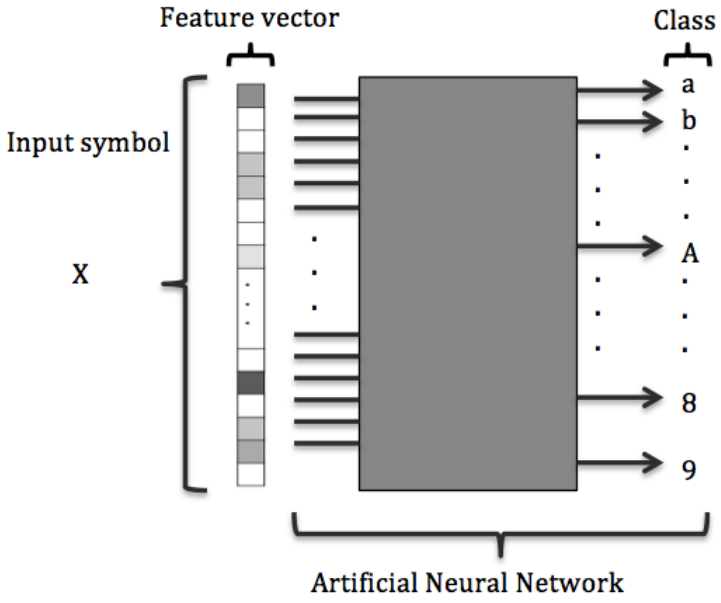
While online features introduce important information regarding the witting process, they can be misleading: for instance, the extracted features of a line written from left to right are different



Feature vector = {SC(p_1) U SC(p_i) U SC(p_j) U SC(p_{30})}

SC = shape context

(a)



(b)

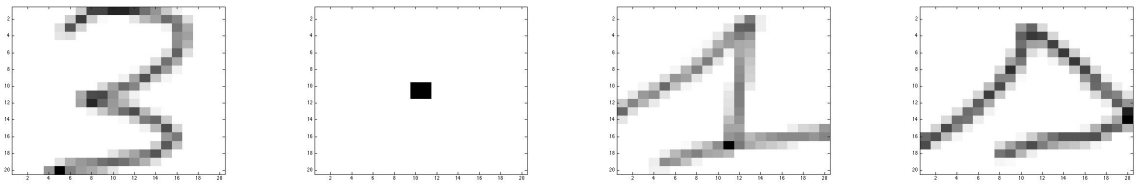
Figure 4.5: *Shape context extraction for artificial neural networks. (a) Shape context is extracted only at some sampled points. (b) The extracted shape context vectors are concatenated to form a feature vector that is used as input to a neural network.*

from those of a line written from right to left, even if the shapes are the same. Offline features are intended to capture symbol shape patterns disregarding the writing process.

Our offline features include 2D histograms calculated over a square box that encloses the symbol. The square box side is defined as the biggest value between the width and height of the symbol. We place the enclosing box so that its center corresponds to the symbol's bounding box center. Thus, the enclosing box is partitioned into $k \times k$ cells, and the histogram is calculated by counting the fuzzy weighted contribution of each symbol's point to its four closest cells (Almazan et al., 2011).

Among the symbol classes, those similar to "." (dot) can be difficult to differentiate. The histogram corresponding to those symbols may have a single point or a small circle that covers the whole enclosing box. To overcome this problem, we keep the enclosing box side bigger than a predefined value. As expressions can be written over different resolutions, defining a fixed value can be difficult. Instead, we calculate the minimum side, for each expression, as follows. First, a complete graph is created with the centers of the strokes of the expression as vertices and the distances between vertices as edge weights. Second, a minimum spanning tree is calculated over that graph. Finally, the minimum side value is calculated as two times the mean of the edge weights of the tree. Figure 4.6 shows some examples of symbol histograms.

In addition to the bidimensional histogram of a symbol, we calculate a second histogram that aims to capture contextual information of the symbol. This contextual histogram is calculated using the bounding box of the symbol as enclosing box, and counting points of the expression that do not belong to the symbol. Figure 4.7 illustrates the calculation of the contextual histogram for some symbol hypotheses of a mathematical expression. These features are able to capture information useful to solve ambiguities. For instance, a vertical line that is part of a symbol + may be similar to the symbol 1 or to one line of a symbol ||. Nonetheless, the corresponding contextual histograms would probably be different.



(a) (b) (c) (d)

Figure 4.6: *Bidimensional (*20×20 *cells) histograms calculated from samples of symbols "3", ".", "1" and "s".*

We also computed an offline version of shape context as features. The offline shape context histograms are defined as in the online case. However, to obtain time independence, we calculate the shape context relative to some parts of the symbol bounding box, not to points of the symbol. That is, for a given symbol, we divide the symbol's bounding box into $k \times k$ regions, and calculate a shape context histogram centered at the center of each region.

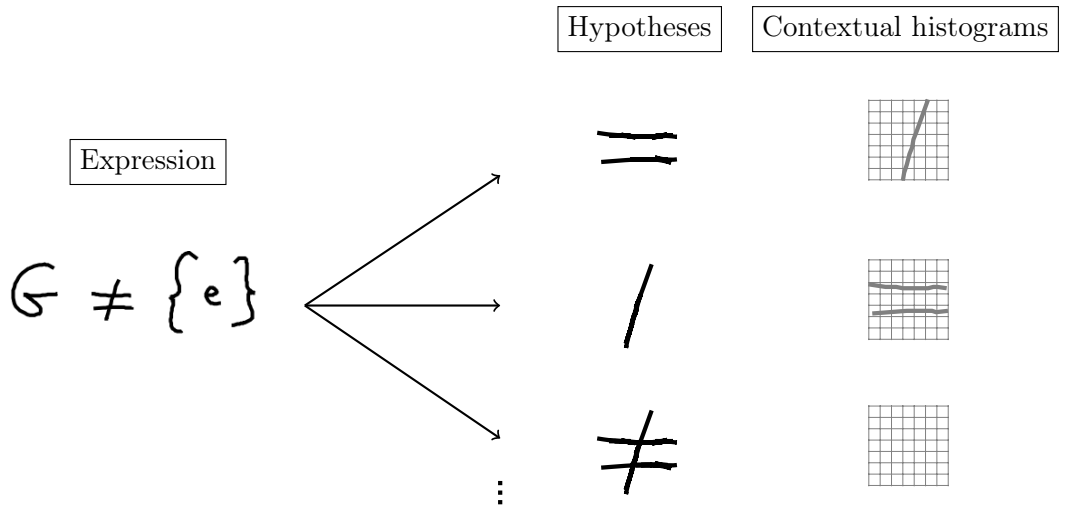


Figure 4.7: *Symbol hypotheses examples and their corresponding contextual histograms.*

4.3 Experimentation

4.3.1 Experimental setup

To evaluate the proposed methods, we used the CROHME-2014 dataset[1] (Mouchère et al., 2014). This dataset includes expressions collected from several laboratories around different countries, and using a variety of input devices, as digital pen technologies, white-boards and tablets. Given this variety of devices, symbols were sampled in different scales and resolutions.

The dataset is divided into a training part with 9,507 mathematical expressions and 91,670 symbols and a test part with 986 expressions and 10,061 symbols. The number of symbol classes is 101. We generated 91,401 false symbol hypothesis from the training expressions and 9,994 from the test expressions (we keep the ratio between true and false hypothesis close to 1 in both training and test sets). For each mathematical expression, the generation of false symbols was done by randomly selecting 9 false hypotheses from those generated by the k-nearest neighbor graph method (9 corresponds to the rounded mean of the number of symbols of the expressions).

In our online shape context implementation, we used 3 radial regions and 8 angular regions, for a total of 24 bins. We calculated shape contexts of 6 equally distributed (in time order) points. By including the number of strokes and jumps, this configuration gives a feature vector of size 6 x 24 + 2 = 192. The 2D Histogram features consisted of 9x9 cells for symbols and 3x3 for contextual histograms. The offline shape context vector was calculated with 3 radial regions and 6 angular regions, for a total of 18 bins. We extracted 9 (offline) shape contexts centered at each cell center of a 3x3 partition of its bounding box. Note that for our evaluation, we consider five feature sets: (1) online, (2) 2d histograms (3) offline shape context, (4) online combined with 2D histograms, and (5) online combined with shape context. All cases include contextual histograms (as previous experimentations showed that including such features improved the recognition performance).

We used a Multi Layer Perceptron neural network with one hidden layer containing a number of units equivalent to the mean between the number of input and output units (102). We used *sigmoid* activation functions in the input and hidden layers and *softmax* functions in the output.

[1]The CROHME 2014 dataset will be publicly available at: http://www.iapr-tc11.org/mediawiki/index.php/Datasets_List.

All parameters were determined using validation data extracted from the training set, as described in Section 7.1. Although other architectures were evaluated, no considerable improvements were found.

We evaluated our proposed methods in two tasks: (1) binary classification, that is, given a symbol hypothesis, the goal is to determine if it is a true symbol or a false symbol and (2) among true symbols, determine which is the symbol class (from the total of 101 symbol classes of CROHME dataset). To do that, we use symbol recognition rate (SRR), false acceptance rate (FAR) and true acceptance rate (TAR) metrics. Considering Table 4.1, FAR and TAR are calculated as (note that these rates do not depend of the ratio between the number of symbol and junk samples):

$$FAR = \frac{\#\{\text{Accepted junk}\}}{\#\{\{\text{Accepted junk}\} \cup \{\text{Rejected junk}\}\}} \tag{4.5}$$

$$TAR = \frac{\#\{\text{Accepted symbol}\}}{\#\{\{\text{Accepted symbol}\} \cup \{\text{Rejected symbol}\}\}} \tag{4.6}$$

Table 4.1: *Symbol hypotheses classifier outcomes.*

System output	Actual class	
	Symbol	Junk
Accepted	Accepted symbol	Accepted junk
Rejected	Rejected symbol	Rejected junk

In the context of a mathematical recognition system, the lower the FAR rate of a classifier, the more false symbols are rejected, which reduces the search space of the whole recognition process. On the other hand, the higher TAR rate of a classifier, the more true symbols are detected, which allows us to keep more true hypotheses. TAR values also correspond to the recall rate of symbols.

4.3.2 Results

Our first experiment aimed to select a best model using the training and validation data. Table 4.2 shows results relative to this experiment. According to the results, the best recognition rates, with and without junk class, were achieved combining online and offline S.C. features – although the combination of online and 2D histogram features achieved almost as good results. On the other hand, the offline features seem to discriminate the junk class better than the online features: while the first have better results without the junk class, the second have better results when the junk class is included. The combinations of both types of features achieve better results in all the evaluated metrics, with and without junk.

After selecting our best model, we evaluated the generalization of the selected model using the test part of CROHME-2014 dataset and compared our results with the top three results (out of eight) of the CROHME-2014 competition. Results of this evaluation are shown in Table 4.3.

Results of Table 4.3 show that while our method performs almost as good as the best results (with a difference of less than 1%) without the junk class, our method performs considerably better than the others when the junk class is included. A main explanation of this behavior may be the fact that we include contextual features, while the other methods use only features that belong to

Table 4.2: *Symbol Classification results using the training-validations sets.*

	Without Junk	With Junk		
Feature set	Rec. rate	Rec. rate	TAR	FRR
Online	91.27	90.14	94.54	7.88
Histogram 2D	86.50	88.00	92.50	6.40
S.C. offline	88.72	89.86	93.69	5.55
Online + Histogram 2D	92.43	93.06	95.65	3.60
Online + S.C. offline	**92.87**	**93.34**	**95.83**	**3.66**

Table 4.3: *Symbol hypothesis classification results using the CROHME-2014 test set and comparison of our method with the top three results of the CROHME-2014 competition.*

	Without Junk	With Junk		
System	Rec. rate	Rec. rate	TAR	FAR
Ours	**89.21**	**90.50**	**92.53**	**3.93**
III	91.04	85.54	87.12	10.39
I	89.79	84.14	80.29	6.44
IV	88.66	83.61	83.52	9.03

the evaluated symbols.

To determine the main sources of wrong classification (in the test set), we analyzed the symbol classes that got the worst recognition rates. Figure 4.8 shows part of those classes (those that got recognition rates lower than or equal to 60%).

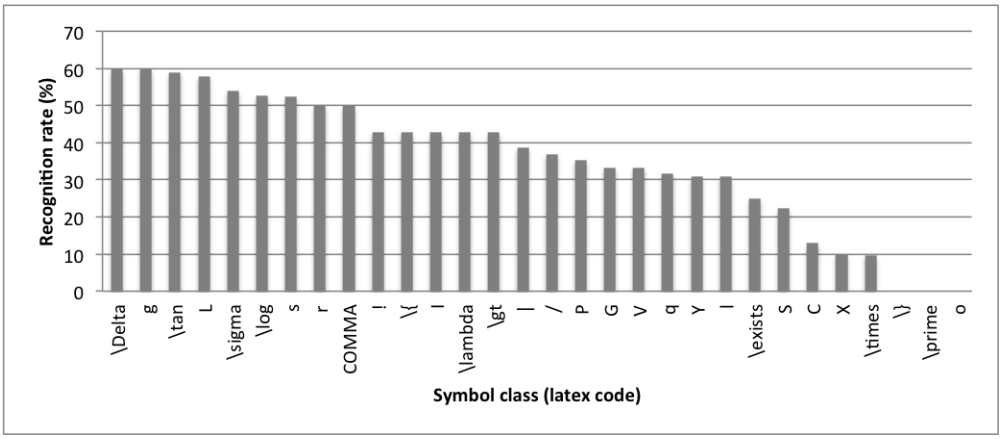


Figure 4.8: *Symbol classes with lowest recognition rates*

Main sources of recognition errors are related to:

- **Low samples per class.** As each person has a particular writing style, there is a large variability in the shape and time related data of each symbol class. Thus the training data must have enough symbol samples per class, so that they represent the different writing patterns. In our dataset, thirteen symbol classes, like $\exists$, λ and $\in$, have low number of samples (less than one hundred samples).

- **Similar symbol classes.** By analyzing the confusion matrix (see Appendix A), it can be seen that high error rates are due to the misclassification of symbols with others with similar shapes. Table 4.4 shows some misclassification rates for those cases. Most of those cases are difficult to solve by considering only shape information. However, in the context of recognition of mathematical expressions, the ambiguity may be solved by considering the location of the symbols (for instance, to differentiate a symbol "," from ")"), relations or relative sizes.

Table 4.4: *Percentage of symbols missclassified with others with similar shape.*

Class	Missclassified as	%
× (\times)	x	86
X	x	87
V	v	47
C	c	61
COMMA	)	16
o	0	100
P	p	59
∃	3	50
Y	y	69

4.4 Discussion

This chapter described new techniques to generate and classify symbol hypotheses from a set of input strokes. We proposed and evaluated a fuzzy version of shape context, contextual histograms and their combination with other previously defined online and offline features. The performance of the proposed techniques were compared with those described in the CROHME-2014 competition.

When considering only true symbol hypotheses, the proposed features obtained state of the art recognition rates. On the other hand, when false symbol hypotheses are included, the proposed features overcame other approaches. This difference seems to be explained by the use of contextual information as part of the features.

It is important to consider that our work was focused mainly in features development and chose Multilayer Perceptron neural networks to evaluate the proposed features. We did not explore other more complex classifiers, like Convolution neural networks or Deep SVMs, etc. The use of those deep learning techniques could improve the current results, as they deal better with histogram-based features.

As described above, ambiguity between similar shape symbols is difficult to solve using only shape information. Those cases may be solved, if possible, by using the context of the symbol. To deal with those cases, the mathematical expression recognition technique of this thesis uses the best classifier described in this chapter to generate a set of symbol hypotheses, assigns a set of alternative labels to each hypothesis, but leaves the final classification decision to be taken during a parsing process (see Chapter 3).

Chapter 5

Relation classification

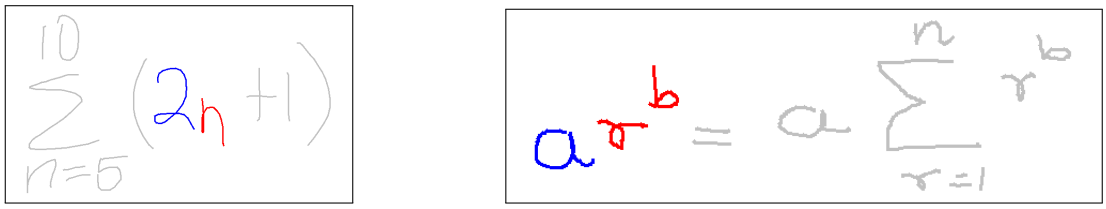
The relation classification problem consists on determining a *mathematical relation* (such as *superscript*) between a pair of subexpressions. In contrast to the symbol classification problem, that has been studied for years and several published works can be found, there are only a few recent works that focus on the relation classification problem, for instance Álvaro and Zanibbi (2013); Simistira et al. (2014).

In this chapter, we describe the relation classification problem issues, our proposed feature sets and techniques to treat the problem and their corresponding evaluation on the CROHME-2014 dataset.

5.1 Problem overview and state of the art

As mentioned above, the relation classification problem consists on determining relations between pairs of subexpressions. In this context, a subexpression is a set of strokes that form a set of symbols. It is assumed that we know when the subexpression is composed of a unique symbol or it is composed of several ones. In the first case, it is also assumed that the symbol class is known. Hereafter, we denote a relation R between a pair of expressions (SE_1, SE_2) as $R(SE_1, SE_2)$.

A main characteristic of handwritten mathematical expressions is that they present irregular arrangements of subexpressions. This irregularity introduces ambiguity to the recognition problem. Figure 5.1 shows ambiguous relation examples.



(a) (b)

Figure 5.1: *Mathematical expressions with ambiguous relations between subexpressions in blue and red. The relation in (a) may be considered as horizontal or subscript and in (2) it can be interpreted as horizontal or superscript.*

Early works on relation classification defined crisp regions relative to symbols and determined relations according to the region that other symbols lie (Anderson, 1968; Tapia, 2004; Zanibbi et al., 2002). The regions were defined according to the symbol category. This categorization is motivated

by the fact that different symbols may be vertically located at different positions over a same baseline. See, for instance, the different relative positions of a *subscript* relation in A_x and y_x. Although they represent the same relation, the relative positions between subexpressions are different. Figure 5.2 shows different region definitions relative to some symbols.

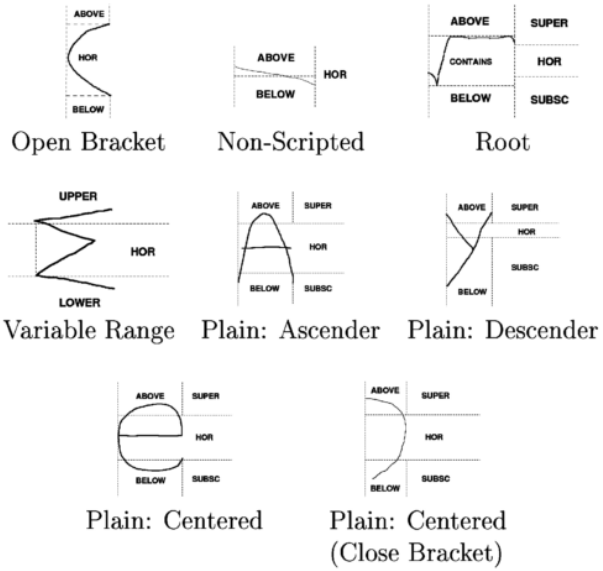


Figure 5.2: *Crisp regions that define relations relative to a symbol. Regions are defined according to the symbol category, that is indicated below each symbol. Extracted from Zanibbi et al. (2002).*

A problem of the method of relation classification based on crisp regions is that they do not consider ambiguous cases that may arise in handwritten data. At this regard, it may be specially difficult to distinguish *horizontal* relations from *subscripts* or *superscripts* (a couple of examples is shown in Figure 5.1). To cope with this problem, Zhang et al. (2005) proposed a method based on fuzzy regions that allowed to consider up to two possible classification results with their corresponding score of confidence. Figure 5.3 illustrates the fuzzy regions definition.

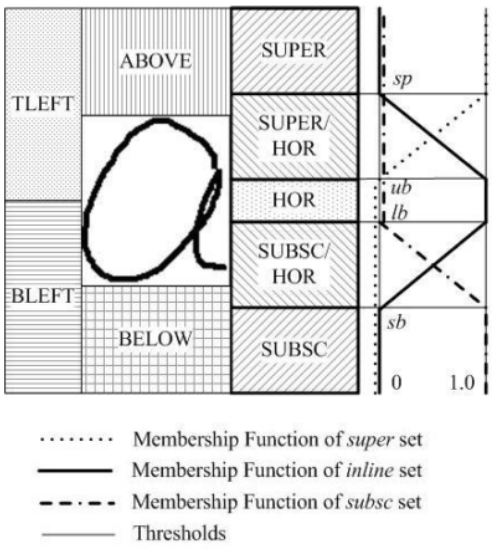
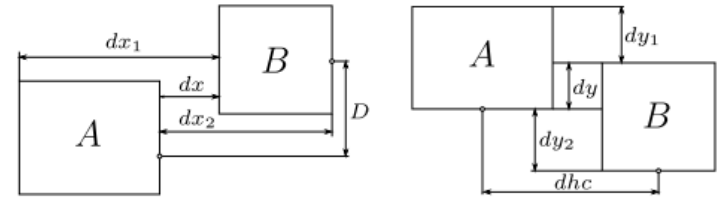


Figure 5.3: *Fuzzy regions that define relations relative to a symbol. Extracted from Zhang et al. (2005).*

A main drawback of the methods described above is that the region limits were still defined

manually and not from a training set. In Álvaro and Zanibbi (2013), the authors proposed nine geometric features calculated from the evaluated subexpression bounding boxes and trained Support Vector Machine classifiers using the MathBrush dataset (MacLean et al., 2011). Given a pair of subexpressions (A, B), the geometric features were calculated as illustrated in Figure 5.4. To capture the different vertical positions of symbols, the authors grouped the symbols into four typographic categories (similar to the aproaches described above): ascendant (e.g., "*d*"), descendant (e.g. "*p*"), normal (e.g. "*x*") and middle (e.g. "7"). Then, the y coordinate of a symbol's centroid was shifted down or up, according to its category. Additionally, the authors also proposed polar histograms as features, similar to Shape Context (Belongie et al., 2002). According to their results, best performance was obtained using geometric features and with categories (the use of categories reduced 0.64% the recognition error in comparison to the geometric features without considering categories).



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

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

Figure 5.4: *Geometric features. F is defined as the distance between the centers of the bounding boxes of the subexpressions. cen_h and cen_v represent the coordinates, in x and y axis respectively, of the centroid of its parameter. Extracted from Álvaro and Zanibbi (2013).*

5.2 Feature set

In this work, we introduce two complementary feature sets: **relation images** and **category vectors**. As a baseline for comparison, we also implemented geometric features based on the work of Álvaro and Zanibbi (2013). The next sections describe each of these feature sets.

5.2.1 Relation image

As described in Section 5.1, relation features proposed in the literature (Álvaro and Zanibbi, 2013; Simistira et al., 2014) are computed using the subexpressions bounding boxes. However, this method of computing features has the drawback of requiring manual design and selection of the best feature set, and thus, the possibility of missing important information.

Instead of manually defining relation features from the bounding boxes, we generate images that capture the complete subexpressions bounding boxes and their relative positions. The idea is to use these images and allow a classifier to learn the useful features.

To compute the image, we, first, define a square region with side length equal to the maximum value between the width and height of the bounding box of both subexpressions. The region is centered at the subexpressions bounding box center. Then, the region is split into $k \times k$ rectangular bins that define two *images*, I_{SE_1} and I_{SE_2}, of size $k \times k$, one for each subexpression's bounding

box, as follows:

$$I_{SE_l}(i,j) = \frac{area(intersection(bin(i,j), b(SE_l)))}{area(bin(i,j))}, \; l = 1,\, 2 \text{ and } 1 \leq i,j \leq k, \tag{5.1}$$

where $area(A)$ is the area of a rectangle A, $bin(i,j)$ is the bin at (i,j), $intersection(A,B)$ is the intersection between rectangles A and B and $b(SE_l)$ is the bounding box of subexpression SE_l. Due to the division by the area of the bin, the values of I_{SE_l} are in the range $[0,1]$.

Finally, we compute the relation image as a weighted sum of the two subexpression's images:

$$I_{SE_1,SE_2} = \alpha * I_{SE_1} + (1-\alpha) * I_{SE_2} \tag{5.2}$$

The weights are used to distinguish between the *pixels* of each subexpression's bounding box (α different from 0.5) and keep the sum of values in the range $[0,1]$. Figure 5.5 shows some relation histogram examples for $\alpha = 0.7$.

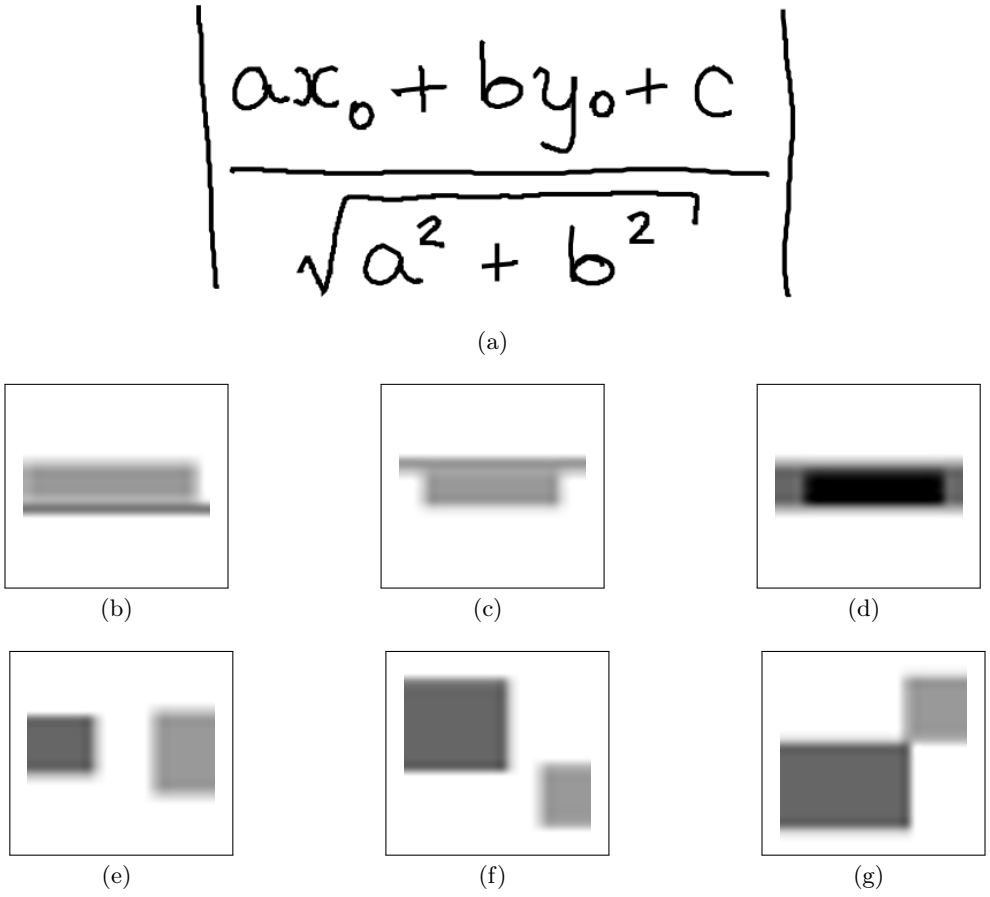


Figure 5.5: *Relation image examples calculated from relations of the expression of Figure (a). Relation images correspond to relations: (b) above(−, ax_0+by_0+c), (c) below(−, $\sqrt{a^2+b^2}$) , (d) inside($\sqrt{\ }$, a^2+b^2), (e) horizontal(+, c), (f) subscript(x, 0) (g) superscript(a, 2).*

5.2.2 Geometric features

In this thesis, we implement the features described in Álvaro and Zanibbi (2013) to compare the performance of the relation image features. However, as it was not possible to get a division of all the subexpression into the categories defined in Álvaro and Zanibbi (2013), the y coordinates of

the symbols centroids are not shifted. To cope with the problematic cases related to categories, we introduce category vectors, as described in the next section.

5.2.3 Category vectors

We propose category vectors as a means to introduce prior knowledge of the symbol types. To do that, we define a four dimensional vector that indicates the baseline relative positions of the subexpressions and wether it corresponds to a single symbol or not.

The first and the second dimensions indicate the normalized position and height of a printed symbol relative to a baseline. Figure 5.6 illustrates these dimensions calculation.

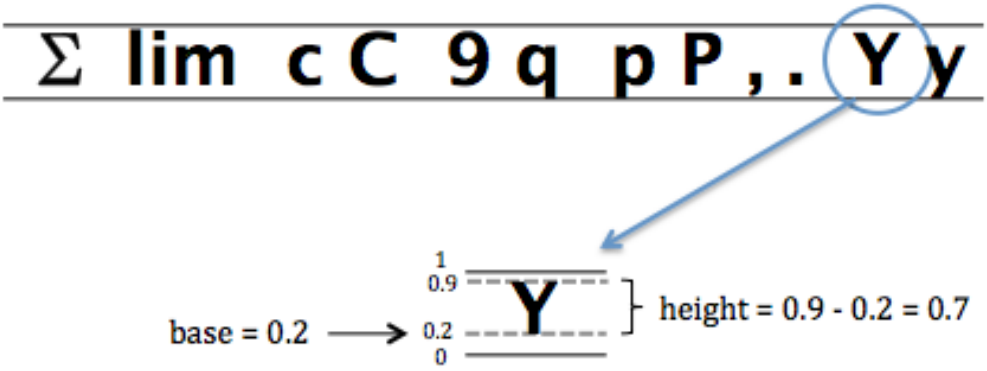


Figure 5.6: *Normalized (values between 0 and 1) base position and height.*

The third and fourth dimensions are binary. The third dimension indicates if the subexpression is a composed symbol (like *cos* and *sin*) or a single symbol (as x and a). The fourth dimension indicates if the subexpression is composed of several symbols (as $a{+}b$) or it is a one symbol (composed or single) subexpression.

5.3 Experimentation

5.3.1 Experimental setup

To evaluate the proposed features, we extracted subexpression relations from the CROHME-2014 dataset. As a result, we got 89,363 relations from the training part of the dataset and 9,481 relations from the test part. The dataset includes six relation classes: *Right* (*horizontal*), *Below*, *Above*, *Superscript*, *Subscript*, *Inside*. Figure 5.7 shows the number of extracted relation samples per class. In both training and test sets, the *Right* relation has a considerable larger number of samples – it represents 74.49% and 75.28% of the total number of samples of the training and test sets, respectively. However, it can be seen that both sets present similar proportions of samples per class.

To find the best parameters for the relation images and select the best feature set, we divided the training part into training and validation parts (as described in Section 7.1). Using those parts, we found that the best relation image size was 10×10.

We used Multilayer Perceptron Neural networks as classifiers, with one hidden layer containing a number of units equivalent to the mean of the number of input features and the number of output units. We used *sigmoid* activation functions in the input and hidden layers and *softmax* functions in the output layers.

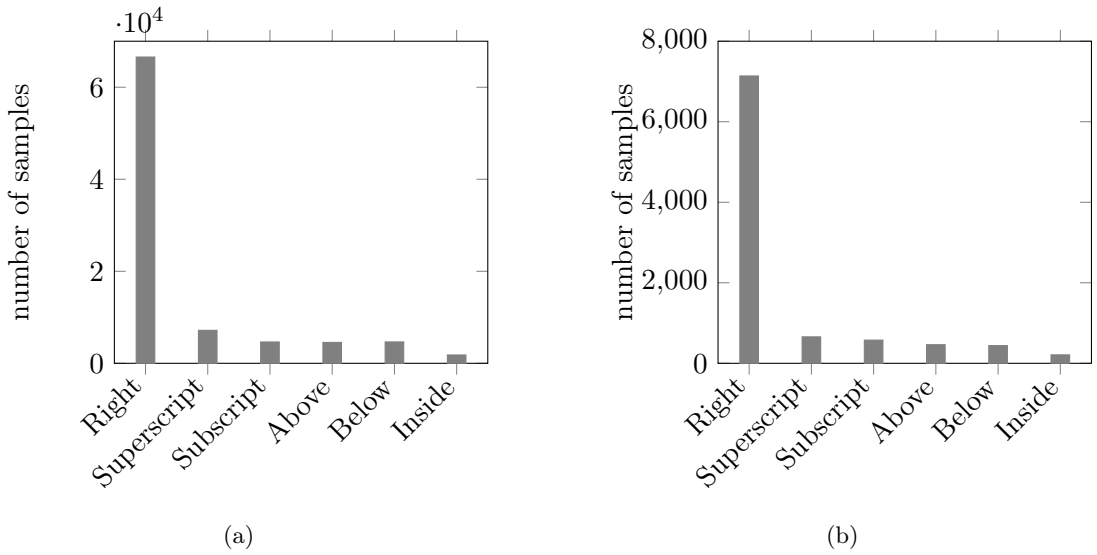


Figure 5.7: *Number of relation samples per class in the (a) training and (b) test sets.*

5.3.2 Results

Table 5.1 compares results obtained with the evaluated features, using the training and validation sets. According to those results, the best recognition rates were obtained for the combination of geometric features and category vectors (CV). Also, it shows an increment of about 3% in comparison to the use of the geometric features alone.

We also evaluated the combination of geometric and histogram features, but no considerable improvement was found.

Table 5.1: *Relation classification rates (%) using the CROHME-2014 train-validation sets. CV = category vectors.*

	Relations						
Feature set	Right	Below	Superscript	Above	Subscript	Inside	Total
Histogram	98.01	95.32	90.68	96.35	64.14	98.51	95.36
Geometric	97.43	99.15	91.24	97.95	62.03	99.00	95.16
Histogram + CV	99.42	98.30	94.63	97.49	89.66	99.50	98.35
Geometric + CV	99.58	99.15	94.92	99.32	91.35	100.00	98.73

To evaluate the generalization performance of the best features (geometric + CV), we evaluated them on the CROHME-2014 test set. Recognition rates on this set was 96.89%. Although a high percentage of the data corresponds to the *Right* relation (about 75%), the recognition rate is considerably higher than that. This means that our classifier is much better than a classifier that only considers likelihood information.

Table 5.2 represents the confusion matrix obtained from the test set results. According to the matrix, misclassification of *Subscript* as *Right* or *Below* and misclassification of *Superscript* as *Right* or *Above* are the most common errors.

After analyzing the misclassified relations, we identified the most frequent types of errors and grouped them into three categories. These categories are illustrated in Figure 5.8.

The first category consists of errors that can be solved by using information from the analyzed

Table 5.2: *Confusion matrix for relations classification of CROHME-2014 test set. The actual relations at rows and the predicted ones at columns.*

	Right	Below	Sup	Above	Sub	Inside
Right	98.53	0.01	1.01	0.00	0.36	0.08
Below	0.46	98.41	0.00	0.00	0.68	0.46
Sup	5.18	0.00	89.19	5.33	0.30	0.00
Above	0.65	0.22	0.22	98.70	0.00	0.22
Sub	8.33	8.51	0.69	0.00	82.47	0.00
Inside	1.90	0.00	0.00	0.48	0.00	97.62

subexpressions. Figures 5.8(a), 5.8(b), and 5.8(c) show examples of this category. In those cases, the bounding boxes alone are not enough to determine the relation. Subexpressions that contain several symbols are specially difficult to recognize: as symbols take arbitrary arrangements, the bounding boxes do not have an specific size or relative position. Having information about the structure of the pair of subexpressions might solve this problem. For instance, the relative positions between symbols of one subexpression that are close to symbols of the other subexpression might be a more robust feature in cases where the subexpressions have many symbols. For our experimentations on mathematical expression recognition (Section 7.1), we take advantage of the fact that we can extract that information after the parse forest is built. Then, the relations classifiers described there also include features extracted from adjacent dominant symbols of the pair of subexpressions.

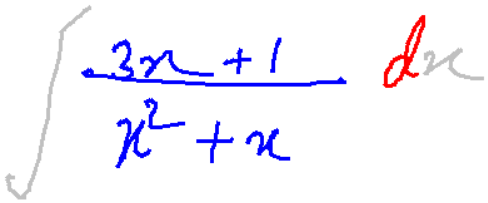
The second category includes relations that would probably not be solved by considering only information from the analyzed subexpressions. Instead, it would be necessary to look at the rest of the expression to solve the ambiguity (if possible). Examples of these category are shown in Figures 5.8(d), 5.8(e), and 5.8(f). In Figure 5.8(d), for instance, it would be useful to detect the writing direction of the symbols adjacent to the considered subexpressions. In Figure 5.8(e), the subscript relation may be inferred by looking at the index in the summation and in Figure 5.8(f) the ambiguity is solved by realizing that there are no other elements that could be used as numerator of the fraction.

In the third category, we included the relation errors generated by inconsistency of the ground-truth. Some of these cases are illustrated in Figures 5.8(g), 5.8(h), and 5.8(i). In those cases, the ground truth specified that the relations were *Above* or *Below*, but they actually look like *Sup* or *Sub*, respectively.

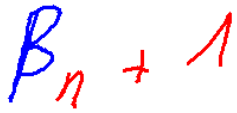
5.4 Discussion

In this chapter, we described the relation classification problem, proposed category vector and relation image features and compared them with the geometric features proposed in Álvaro and Zanibbi (2013). Results showed a comparable performance (with difference of less than 1%) between the relation image and geometric features alone. When combined with category vectors, both geometric and image features increased their rates about 3%. The best recognition rate obtained on the CROHME-2014 test set was 96.89%, combining geometric features with category vectors.

The misclassified relations included three main cases: relations solvable using local information, relations that need contextual information and relations with inconsistent ground-truth. The first



(a) Right→Sup



(b) Sub→Right

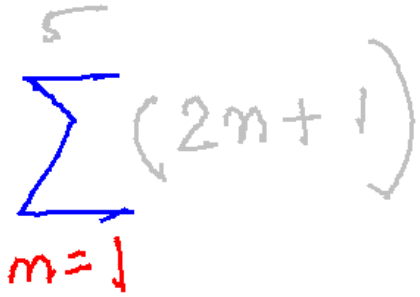
(c) Right→Sub

(d) Right→Sup

(e) Sub→Right

(f) Above→Right

(g) Sub→Below



(h) Sub→Below

(i) Sub→Above

Figure 5.8: *Misclassification examples. Each image label indicates the true relation class followed by the classifier's output, that is: true relation→output.*

category suggests that using only local features might improve the state of the art recognition rates. With respect to the second category, the introduction of contextual information could be done while performing the recognition of the whole expression. These results recall the need for integrated techniques that allow to take context dependent decisions during the recognition process.

While the relation images combined with category vectors generated results almost as good as the best ones, they still have potential for improvement. In our work, we used Multilayer Perceptron Neural Networks due to their easy configuration. However, other techniques, specially those based on Deep-learning, might capture better the image patterns. Experimentation with those techniques is out of the scope of this thesis, but will be considered on future work.

Chapter 6

Mathematical expressions parsing technique

As described in Chapter 1, ambiguity is a main problem in recognition of online handwritten mathematical expressions. Ambiguity in the recognition process appears at symbol (segmentation and classification) or relations levels. Different choices at any of those levels generate different interpretations of the input.

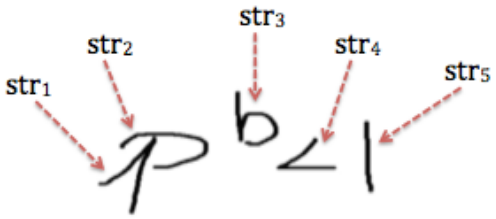
The parsing technique proposed in this thesis is divided into two steps: (1) Parse forest construction (Section 6.1) and (2) Optimal parse tree extraction (Section 6.2). The first step aims to determine all possible interpretations of the input. As a large number of interpretations may be generated, a parse forest is built in order to efficiently store such interpretations. The parse forest construction is based on a top-down approach (Section 6.1.1), but also integrates bottom-up information to solve a key issue of the technique: given a production rule r, how to partition a stroke set according to r? (Section 6.1.2). The second step traverses the parse forest to extract a best tree according to a ranking function (Section 6.2.1). We propose a ranking function that linearly combines costs assigned to symbol and relations that compose a parse tree. In addition, we adapt a ranking function based on fuzzy sets, proposed in MacLean and Labahn (2013).

6.1 Parse Forest Construction

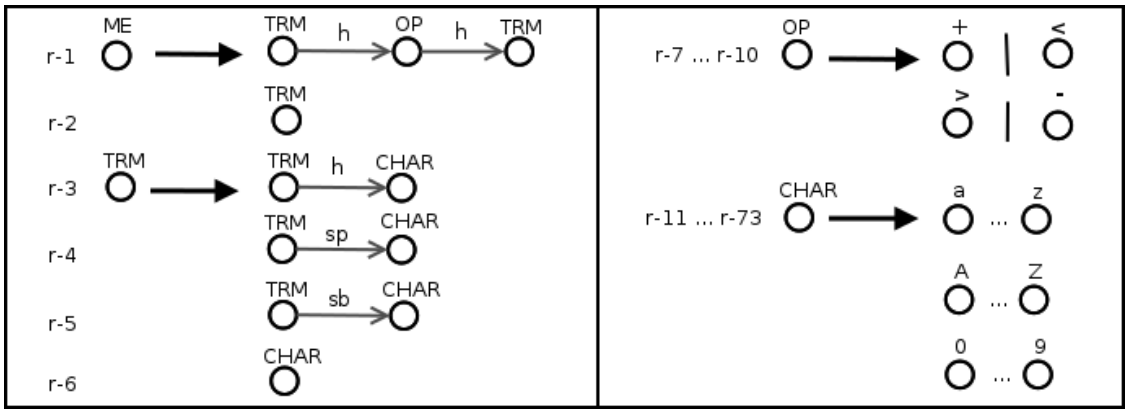
Given an input consisting of a set of strokes, denoted as $str = \{str_1, \ldots, str_n\}$, and a graph grammar, denoted as M, this process builds a parse forest that contains all possible parse trees that can be generated from the input. To recall the data involved in this process, Figure 6.1 shows the stroke set, grammar and parse forest examples described in Chapter 3.

The parse forest is built by calculating recursive partitions of the input strokes, according to the grammar rules and a set of valid stroke partitions. First, we describe the complete algorithm to build the parse forest (Section 6.1.1), then, we describe which stroke partitions are considered as valid (Section 6.1.2).

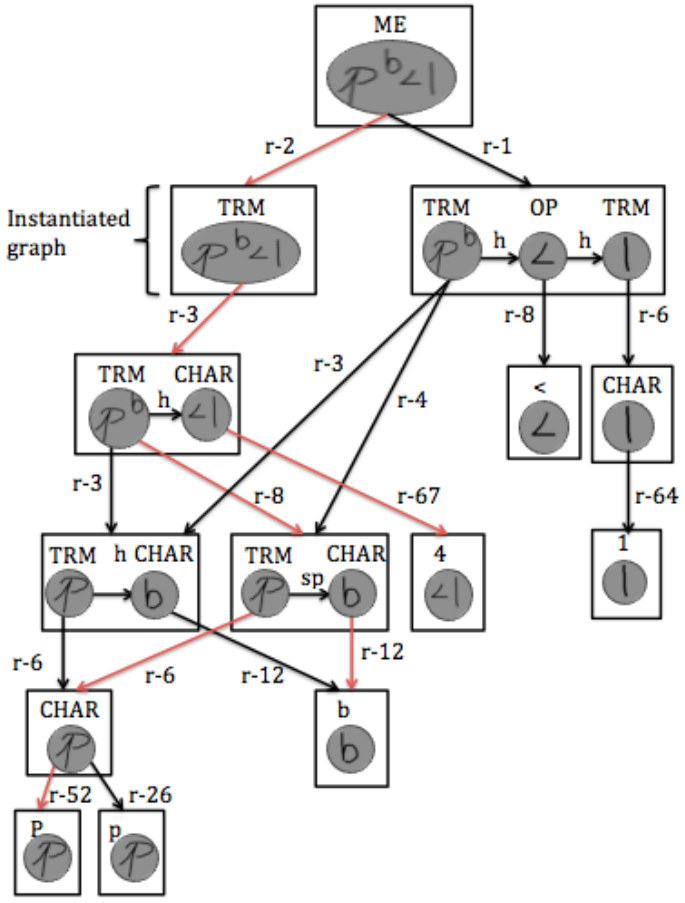
As mentioned in Section 3.2, a parse forest defines different interpretations of a stroke set through multiple partitions. Each partition is defined by an instantiated graph. A parse tree is then composed of instantiated graphs that define a unique interpretation of the stroke set. Given a parse tree t and a node $x \in t$, with instantiated graph $g(x) = (V_{g(x)}, E_{g(x)})$, we introduce the following



(a)



(b)



(c)

Figure 6.1: *The parse forest construction step receives a set of strokes as input (a) and uses a graph grammar (b) to build a parse forest structure (c). A parse forest stores multiple interpretations of the input as parse trees. Red arrows of the parse forest represent a a parse tree that corresponds to the interpretation "$P^b 4$".*

notation to help the further explanations:

- $\forall\, v \in V_{g(x)}$, $label(v)$ is the terminal or non-terminal symbol assigned to the vertex v, $str(v)$ is the set of strokes assigned to v, and $child(v)$ is the node of t that defines a partition of $str(v)$ (if $label(v)$ is terminal, $child(v)$ is $\emptyset$).
- $symb(x)$ is the set of all symbols *generated* by the subtree of t with root in x.
- $rel(x)$ is the set of all subexpression relations *generated* by the subtree of t with root in x.

6.1.1 Top-down parsing algorithm

Algorithms 1, 2, 3 summarize the parse forest construction. The technique follows a top-down parsing approach. To simplify the explanation, we assume that a grammar is defined only with two types of rules: *terminal* and *non-terminal*. In a terminal rule, denoted as $A \rightarrow b$, the RHS graph is a single vertex graph, whose unique vertex is denoted as b and it is labeled with a terminal symbol – rules from r-7 to r-73 of the grammar of Figure 6.1 for instance. A non-terminal rule refers to a rule whose RHS graph contains one or more vertices, all of them labeled only with non-terminal symbols – as rules r-1 to r-6 of the grammar of Figure 6.1. In addition, we will refer to the non-terminal of the start graph as the *start symbol*.

The parsing process starts with a call to the method described in Algorithm 1. The input to this method is a stroke set, denoted as $str = \{str_1, \ldots, str_n\}$ [1]. The output is a set of pairs composed of instantiated graphs and rules that "generate" such graphs (parsingResult). All generated results are recorded in a table denoted as T. This table is indexed by a pair of strokes and non terminal symbol, that is:

$$T(str = \{str_1, \ldots, str_n\}, NT) = \{(g_1, r_1), \ldots, (g_q, r_q)\}, \tag{6.1}$$

where g_i is an instantiated graph and r_i is the rule that "generated" g_i. The parse table T stores parsing results in order to avoid recomputation [2]. This is done by table lookup in Lines 2-3 of Algorithm 1: if results for a pair (str, NT) have already been calculated, the results are retrieved from the table. On the other hand, if results have not been calculated yet, the rules that have NT in its LHS graph are selected (line 5 of Algorithm 1). For each selected rule, all partitions that can be derived using the rule are then calculated and encapsulated as new parsing results (lines 6-17). This is done according to the type of rule being analyzed. If the rule is terminal, Algorithm 2 is called with the stroke set and the rule as arguments. In this case, it suffices to check if the RHS vertex label of the rule (which is a terminal symbol) is contained on a set of labels attributed to the stroke set. This verification is made by checking if the hypothesis graph contains a symbol hypothesis whose stroke set corresponds to the evaluated strokes and contains a label set that includes the vertex label.

If the rule is non-terminal, the instantiatedGraphs method is called, with the stroke set and the rule as arguments. This method calculates multiple stroke set partitions according to the rule. The results of this method are stored as instantiated graphs. After that, for each instantiated graph,

[1] a set is denoted as a braced list of elements, that is $set = \{element_1, \ldots, element_n\}$ and a sequence (ordered set) as a bracketed list of elements, that is $sequence = (element_1, \ldots, element_n)$, where $element_i$ is the i^{th} element considering a particular order.

[2] This technique has originally been proposed in strings parsing (Grune and J.H, 2008)

Algorithm 3 is called, with the instantiated graph and the rule as parameters. This algorithm iterates over each vertex of the graph to parse the vertex's stroke set with its corresponding non-terminal symbol. In this case, if the parsing fails for one stroke set–non-terminal pair, the parsing for the whole graph fails, and an empty set is returned. Otherwise, the instantiated graph and the rule are returned as a result.

Algorithm 1: parseMathExpression. Parses a set of strokes over a non-terminal, following a top-down approach.

Input : $(str = \{str_1, \ldots, str_n\}, nonTerminal)$.
Output: $parsingResult = \{(g_1, r_1), \ldots, (g_q, r_q)\}$.

```
1  parsingResult ← ∅;
2  if parsed(str, nonTerminal) then
3  |  parsingResult ← T(str, nonTerminal) ; // Result is already in T
4  else
5  |  rulesOfNonTerminal ← rulesWithLHS(nonTerminal);
6  |  foreach rule ∈ rulesOfNonTerminal do
7  |  |  if rule is terminal then
8  |  |  |  parsingResult = parsingResult ∪ parseTerminalRule(str, rule);
9  |  |  else
   |  |  |  ; // Rule is non-terminal
10 |  |  |  instantiatedGraphs ← instantiateGraph(str, RHS(rule))
11 |  |  |  foreach iG ∈ instantiatedGraphs do
12 |  |  |  |  parsingResult = parsingResult ∪ parseNonTerminalRule(iG, rule);
13 |  |  |  end
14 |  |  end
15 |  end
16 |  T(str, nonTerminal) ← parsingResult;
17 |  parsed(str, nonTerminal) ← True;
18 end
19 return parsingResult;
```

Algorithm 2: parseTerminalRule. Parses a set of strokes over a terminal rule.

Input : $(str = \{str_1, \ldots, str_n\}, rule = A \rightarrow b)$.
Output: $parsingResult = \{(g_1, r_1), \ldots, (g_q, r_q)\}$.

```
1  parsingResult ← ∅ ;
2  if α(b) ∈ γ(str) then
   |  ; // The instantiated graph has only one vertex (b)
   |  ; // and the parsingResult only one element
3  |  instantiatedGraph ← buildGraph(str, α(b));
4  |  parsingResult ← {(instantiatedGraph, rule)}
5  end
6  return parsingResult;
```

Turan (1982) showed that the graph grammar parsing is a NP-complete problem. This complexity is dominated by the isomorphism search between the RHS graph of a rule and a graph derived from the input (in this thesis, the graph is derived from the hypotheses graph). Some previous approaches proposed small or constrained grammars to reduce the parsing complexity (Celik and Yanikoglu, 2011; Han and Zhu, 2009; Lavirotte and Pottier, 1997). In this work, we

Algorithm 3: parseNonTerminalRule. Parses a set of strokes over a non-terminal rule.

input : $(iG = (V_{iG}, E_{iG}), rule)$.
output: $parsingResult = \{(g_1, r_1), \ldots, (g_q, r_q)\}$.

```
foreach v ∈ V_iG do
    partialResult ← parseMathExpression(str(v), label(v));
    if partialResult = ∅ then
        return ∅;
    end
end
return {(iG, rule)};
```

keep the grammar and the parsing methods general, and manage the complexity through the hypotheses graph. Although this makes the parsing algorithm to have an exponential cost in the worst case, results described in Chapter 7 show that the proposed algorithms are efficient enough to be applied in an online context.

6.1.2 Valid stroke partitions

A key step of the proposed parsing algorithm is to determine a set of stroke partitions that match the RHS graph of a production rule. Given a stroke set $str = str_1, \ldots, str_n$ and a production rule r, a partition of str that matches $RHS(r) = B = (V_B, E_B)$ consists on assigning a non-empty subset of str to each vertex of $RHS(r)$. Without any constraint, the total number of stroke partitions is $O(|V_B|^n)$.

Among all the partitions of a stroke set, most of them may not be worth evaluating. For instance, in Figure 6.2, it would not be useful to evaluate the partition indicated by red lines. Furthermore, it would not be useful to evaluate any partition where the stroke "b" is mapped to the vertex with label OP.

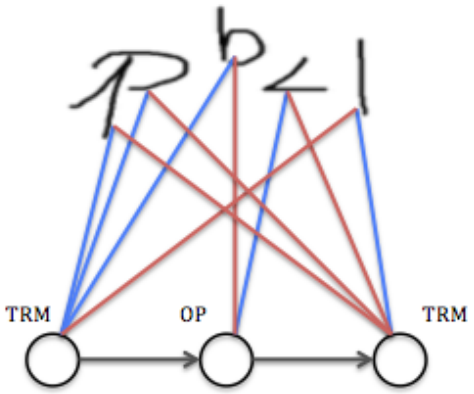


Figure 6.2: *Two partitions of a stroke set according to a production rules's graph. Blue lines indicate a right partition and red lines indicate a wrong one.*

To improve the search for stroke partitions, some previous approaches that used string grammars proposed different ways to introduce an order in the strokes set. For instance, Yamamoto et al. (2006) sorted the strokes according to the input time, and MacLean and Labahn (2013) sorted the strokes horizontally or vertically, using the minimum (x, y) coordinates of each stroke. Having an order in the stroke set and in the production rules (in string grammars the elements of a production

rule are sorted from left to right), the total number of partitions is exponential in the number of elements of the right hand side of the rule (not in the number of elements of the stroke set).

The approach proposed in this thesis does not constrain the stroke set to have a specific order. Instead, the search space for partitions is derived from the hypotheses graph of the stroke set. As described in Section 3.3.1, a hypotheses graph is composed of a set of symbol hypotheses and a set of relation hypotheses. The symbol hypotheses set represents groups of strokes that may be interpreted as symbols, and the relation hypotheses set represents possible relations between those symbols. Given a rule r, and a stroke set str, the set of partitions P of str according to r is given by all the minor graphs of the hypotheses graph of str [3] that are isomorphic to RHS(r). This means that a group of strokes are mapped to a vertex of $RHS(r)$ only if the strokes belong to a *vertex* formed by contracting edges of the hypotheses graph. The vertices formed by contracting edges (partition vertices) inherit the relation labels of their children. The isomorphism search requires that the relation labels of the $RHS(r)$ match those of the partition vertices. Figure 6.3 shows an example of a hypotheses graph and a set of vertices formed by contracting their edges.

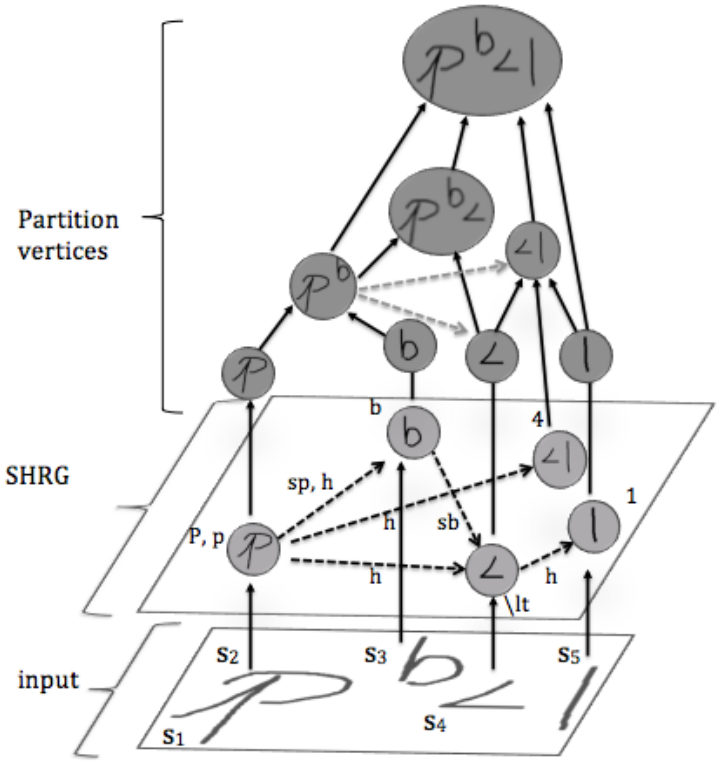


Figure 6.3: *A hypotheses graph and vertices formed by contracting edges. The dashed arrows of the vertex composed of strokes P^b represent inherited edge.*

6.2 Optimal parse tree extraction

Once the parse forest is built, the final step consists on extracting a tree that is more likely to represent an interpretation of the input. To do that, we need a ranking function over the trees of the parse forest. In this thesis, we propose a cost-based ranking function and compare it with

[3] A graph H is a *minor* of a graph G if H can be obtained from a **subgraph** of G by contracting edges Wagner (1937).

a geometric averaging function based on the work of MacLean and Labahn (2013); Zhang et al. (2005).

From an application point of view, it is better to have a general and efficient method to extract a k-best tree, so that if the best tree does not correspond to a correct interpretation, the following best trees could be retrieved. The best tree extraction technique proposed in this thesis is based on the work of Boullier et al. (2009). The technique consists of two processes. The first process traverses the parse forest in a bottom-up way, calculating, for each pair (str, NT), a table that contains alternative partitions of str sorted in descending order by their likelihood score. Each row of a table stores a single partition of str. As each part of a partition may be subsequently partitioned, for each part, a row also stores the position of a row that defines the subsequent partition of the part on its corresponding table. Once the bottom-up process calculates the tables for each pair (str, NT) of the parse forest, the k-best tree is extracted by a top-down process. This process begins by extracting the partition of the row at position k of the table of the start symbol and the complete set of strokes pair. Then, the subsequent partitions are extracted recursively from the tables corresponding to each part. The last step is repeated until obtaining partitions that correspond to terminal symbols.

In the case of recognition of mathematical expressions, evaluating all parse trees for complex expressions is infeasible. We then propose a pruning strategy applied in the bottom-up step of the k-best tree extraction method. The pruning technique consists on avoiding parse subtrees that have low likelihood scores: for each table of a pair (str, NT), only rows whose partitions have a likelihood score above a predefined threshold are considered. The threshold is calculated experimentally.

6.2.1 Ranking functions

Geometric averaging

In MacLean and Labahn (2013); Zhang et al. (2005) the authors proposed fuzzy-based score functions. The set of all possible interpretations is defined as a fuzzy set and a ranking function is interpreted as a membership function. The membership function is defined recursively as the geometric mean of the stroke partitions and relations of a parse tree.

Considering graph grammars, the membership score of a parse tree node x is given by:

$$mScore_n(x) = \left(\left(\prod_{v \in V_{g(x)}} mScore_{vert}(v) \right) \left(\prod_{e_{i,j}=(v_i,v_j) \in E_{g(x)}} mScore_{edge}(e_{i,j}) \right) \right)^{\frac{1}{|V_{g(x)}|+|E_{g(x)}|}}, \tag{6.2}$$

where $mScore_{vert}(v)$ and $mScore_{edge}(e_{i,j})$ give the membership score of vertex v_i and edge $e_{i,j}$, respectively. The score $mScore_{vert}(v_i)$ is calculated as:

$$mScore_{vert}(v) = \begin{cases} mScore_c(label(v), str(v)), & \text{if } label(v) \in T \\ mScore_n(child(v)), & \text{if } label(v) \in N, \end{cases} \tag{6.3}$$

where $mScore_c(label(v), str(v))$ gives the confidence value of the strokes $str(v)$ representing a terminal symbol $label(v)$. This value is given by the best symbol classifier described in Section 4.3.2.

The score $mScore_{edge}(e_{i,j})$ gives the confidence value of having a relation $e_{i,j}$ between subexpressions composed of strokes $str(v_i)$ and $str(v_j)$. The value is calculated using the best relation classifier described in Section 5.3.2.

Using the above equations, the membership score or confidence value of a parse tree t is given by $mScore_n(x_r)$, where x_r is the root of t.

The calculation of the confidence value of a parse tree is illustrated using the tree of Figure 6.4. The elements of the three are indexed in order to identify the cost at each tree part. Then, the fuzzy membership score for the tree is calculated as:

$$\begin{aligned} mScore_n(x_1) &= mScore_{vert}(v_1) \\ &= mScore_n(child(v_1)) \\ &= mScore_n(x_2) \\ &= ((mScore_{vert}(v_2) \times mScore_{vert}(v_3) \times mScore_{vert}(v_4)) \\ &\quad \times (mScore_{edge}(e_1) \times mScore_{edge}(e_2)))^{\frac{1}{3+2}}, \end{aligned} \tag{6.4}$$

where the score $mScore_{vert}(v_2)$ is given by:

$$\begin{aligned} mScore_{vert}(v_2) &= mScore_n(x_3) \\ &= ((mScore_{vert}(v_5) \times mScore_{vert}(v_6)) \times mScore_{edge}(e_3))^{\frac{1}{2+1}} \\ &= ((mScore_c(\text{“}P\text{”}, str(v_8)) \times mScore_c(\text{“}P\text{”}, str(v_9))) \times mScore_{edge}(e_3))^{\frac{1}{3}}, \end{aligned} \tag{6.5}$$

similarly, the score $mScore_{vert}(v_3)$ and $mScore_{vert}(v_4)$ are calculated as:

$$mScore_{vert}(v_3) = mScore_c(\text{“}<\text{”}, str(v_{10})), \tag{6.6}$$

$$mScore_{vert}(v_4) = mScore_c(\text{“1”}, str(v_{12})), \tag{6.7}$$

From the above equations, we can see that while the scores of vertices v_3 and v_4 are given by the score of the classification of two symbols, the score of vertex v_2 is given by the score of the classification of two symbols and one relation. As the score of v_2 is calculated as a geometric average, a variance in a factor its score ($mScore_c(\text{“}P\text{”}, str(v_8))$, $mScore_c(\text{“}P\text{”}, str(v_9))$, or $mScore_{edge}(e_3)$) is not as influential as one variance in a factor of the score of v_3 ($mScore_c(\text{“}<\text{”}, str(v_{10}))$) or v_4 ($mScore_c(\text{“1”}, str(v_{12}))$). Similarly, a variance in a factor of score v_2 is not as influential as a variance in the score of relations e_1 or e_2. The cost-based score described in the next section assigns equal *influence* or *weight* to each symbol or relation classification score.

Energy minimization

The energy minimization score assigns costs to symbols and subexpression relations that compose a parse tree and combines both costs with a linear function. This score does not define a recursive function, but considers all relations and symbols that are generated by a subtree rooted at a particular tree node.

Formally, the cost of a parse tree node x is defined as:

$$cost(x) = \alpha \left(\frac{\sum_{v \in term(t'_x)} cost_c(label(v), str(v))}{|term(t'_x)|} \right) + (1-\alpha) \left(\frac{\sum_{e \in rel(x)} cost_r(e)}{|rel(t'_x)|} \right), \tag{6.8}$$

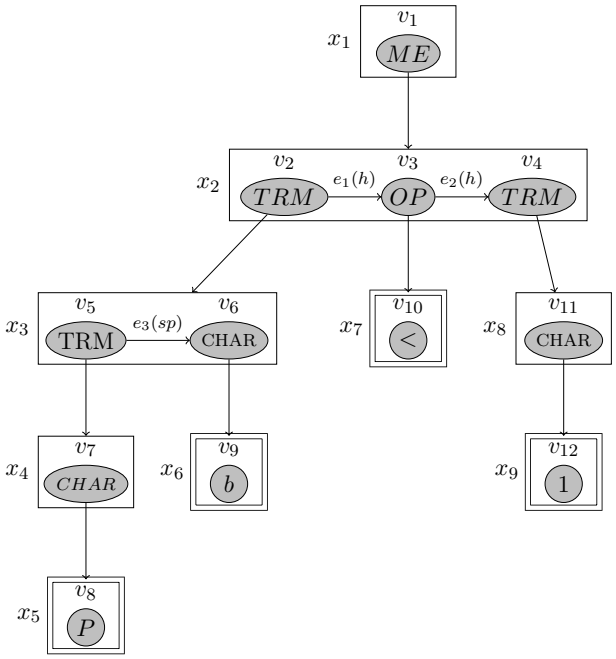


Figure 6.4: *Parse tree of expression $P^b < 1$, extracted from the parse forest of Figure 6.1(c). Nodes are indexed as x_i, for $i = 1, \ldots, 9$. Similarly, vertices and edges of the instantiated graphs are respectively indexed as v_j, for $j = 1, \ldots, 12$, and e_k, for $k = 1, \ldots, 3$. Nodes with terminal symbols are depicted with double line borders.*

where t'_x is the subtree of t that has x as root, $term(t'_x)$ is the set of all vertices of t'_x that have terminal symbols as labels, $cost_c(label(v), str(v))$ is the cost of considering $str(v)$ as a symbol $label(v)$, and $cost_r(e)$ is the cost of considering e as a relation. The cost scores are calculated by applying a logarithm function to the symbol and relation classifier outputs. That is, given a symbol or relation classifier output z, the corresponding cost score is given by $-log(z)$.

Similarly to the geometric averaging score, the cost of a parse tree t is given by $cost(x_r)$, where x_r is the root of t. In the case of the tree of Figure 6.4, the cost is calculated as:

$$\begin{aligned} cost(x_1) &= \alpha \left(\frac{\sum\limits_{v \in term(t'_{x_1})} cost_c(label(v), str(v))}{|term(t'_{x_1})|} \right) + (1-\alpha) \left(\frac{\sum\limits_{e \in rel(t'_{x_1})} cost_r(e)}{|rel(t'_{x_1})|} \right) \\ &= \alpha \left(\frac{cost_c(\text{“}P\text{”}, str(v_8)) + cost_c(\text{“}b\text{”}, str(v_9)) + cost_c(\text{“}<\text{”}, str(v_{10})) + cost_c(\text{“}1\text{”}, str(v_{12}))}{4} \right) \\ &\quad + (1-\alpha) \left(\frac{cost_r(e_1) + cost_r(e_2) + cost_r(e_3)}{3} \right), \end{aligned} \tag{6.9}$$

Chapter 7

Experimentation

In this chapter, we evaluate the performance of the complete recognition system. The evaluation is done over the CROHME-2014 dataset (Mouchère et al., 2014).

7.1 Experimental setup

The CROHME-2014 dataset consists of mathematical expressions that have been handwritten by hundreds of people from different countries. It is divided into training and test sets, with 9,507 and 986 mathematical expressions, respectively. From the training set, we randomly selected 950 expressions (about 10%) to optimize our system parameters (validation set).

The organizers of the CROHME competition provided a string grammar that defines a language of the mathematical expressions included in the dataset [1]. In order to evaluate our system on the dataset, we implemented a graph grammar version of the string grammar. Our graph grammar is formally defined as described in Chapter 3, and is composed of 205 production rules, including rules to generate the 101 symbol classes (terminals) considered in the competition.

To comprehensively evaluate a mathematical expression recognition system, performance metrics should measure recognition rates at different levels (Lapointe, 2008; Lapointe and Blostein, 2009). One of the most common metrics is the expression recognition rate (expression level). This metric measures the percentage of correctly recognized expressions from the total ones, where an expression is considered correctly recognized when all its symbols are correctly segmented and classified and all spatial relations between its symbols have been correctly identified. However, this metric does not provide an insight about *how much* of an expression has not been correctly recognized, neither distinguish between symbol and relation errors. For example, according to this metric, it does not matter if an expression $\sum_{i=0}^{n} x^i$ is missrecognized as $\sum_{i=0}^{n} x^1$ or $\sum_{i=0} mX$; in both cases the result would be counted as one error, although the second one contains more symbol and relations missrecognized.

In the CROHME competition (Mouchère et al., 2014), to complement the expression recognition rate, the systems have also been evaluated using object level metrics. These metrics include recall and precision of symbol segmentation and recognition and spatial relation detection. Recall and precision of symbol segmentation and recognition are defined as usual. Spatial relation detection measures the percentage of correctly identified relations between pairs of symbols, where a relation

[1] http://www.iapr-tc11.org/dataset/CROHME/CROHME_full_v2.zip

is considered well identified if the symbols at both sides of the relation are correctly segmented and the right relation class is identified. In our evaluation, we consider both object and expression level metrics, and report results using the evaluation tools provided by the organizers of the competition[2].

Seven systems participated in the CROHME-2014 competition. We identify the systems as: System I, System II, ... , System VII. These names correspond to those given in the competition. The systems implement a variety of approaches, including parsing techniques with string grammars, baseline extraction, and statistical models. A brief description of each system is given in Mouchère et al. (2014).

As explained in Chapter 3, our recognition method comprises two stages: hypotheses graph generation and graph parsing. To optimize our recognition method, we need to set the values of parameters in both stages in such a way that the method provides high accuracy and is efficient enough to be used in a real scenario.

The hypotheses graph generation stage is configured according to a pair of symbol and relation classifier thresholds. Recall that to compute a hypotheses graph, we train symbol and relation classifiers (described in Chapters 4 and 5, respectively) to identify symbol and relation hypotheses with their corresponding label sets. The label set of a particular hypothesis (symbol or relation) is calculated using likelihood scores given by the classifiers. Given a hypothesis h and a label set $(l_1, \ldots, l_m)$, sorted in descending order by their likelihood scores $score(l_i)$, for $i = 1, \ldots, m$, a set of k labels is assigned to h, such that:

$$k = \arg\min_{k} \sum_{i=1}^{k} score(l_i) > tr, \tag{7.1}$$

where tr defines a minimum confidence value to select a label set or to reject a hypothesis. A hypothesis is rejected if its *junk* label score is the biggest one and it is greater than the threshold. Thus, the higher a symbol/relation threshold value, the more symbol/relation hypotheses are accepted (stored in the graph), and the more labels are assigned to them. It is important to note that for a symbol classifier threshold value 1, all the stroke groups generated by the stroke grouping algorithm (described in Chapter 4) are accepted as symbol hypotheses and all possible labels are attributed to each of them. In the case of relations, for a threshold value of 1, a relation hypothesis, attributed with all relation labels, is created for each pair of symbol hypotheses.

The graph parsing stage is configured through a threshold that defines a minimum cost that a tree should have in order to be considered as a possible result. When using the cost function, an additional parameter is a weighting α between symbol and relation costs (value in the range [0-1]). The larger the value of α, the more weight is given to the symbol cost and the less to the relation cost (see Section 6.2.1 for more details).

To select the optimal parameter values, we used the training and validation sets as follows. We trained the symbol and relation classifiers with symbols and relations extracted from the training set. We evaluated several feature sets for each classifier and selected the best ones using the validation set. The results regarding the symbol and relation classifiers' performance are detailed in Chapters 4 and 5, respectively. Once the classifiers were optimized, we used them to select the hypotheses graph generation and graph parsing parameter values. First, we selected the best symbol and relation classifier threshold values. Then, using the selected values, we determined the tree extraction

[2] https://www.cs.rit.edu/~dprl/CROHMELib_LgEval_Doc.html

threshold and the weighting between symbol and relation costs. This evaluation was also done over the validation set, and the corresponding results are described in Section 7.2.

After setting the optimal parameter values, we evaluated the resulting system in the test set and compared its performance with systems that participated in the CROHME-2014 competition. The corresponding results are described in Section 7.3.

7.2 Parameter setting

This section describes the selection of parameter values for the hypotheses graph generation and the graph parsing techniques. This evaluation is done using the validation set.

7.2.1 Hypotheses graph generation

To evaluate the hypotheses graph generation step over a combination of symbol and relation threshold values, we calculate the recall of symbols, relations and complete expressions contained in the hypotheses graphs generated for such combination. Figure 7.1 shows the results relative to this evaluation, over symbol and relation threshold values in the range [0-1]. Note that these results also indicate the maximum recognition rates achievable by the parsing technique (or the missrecognition rates induced by the hypotheses graph generation stage).

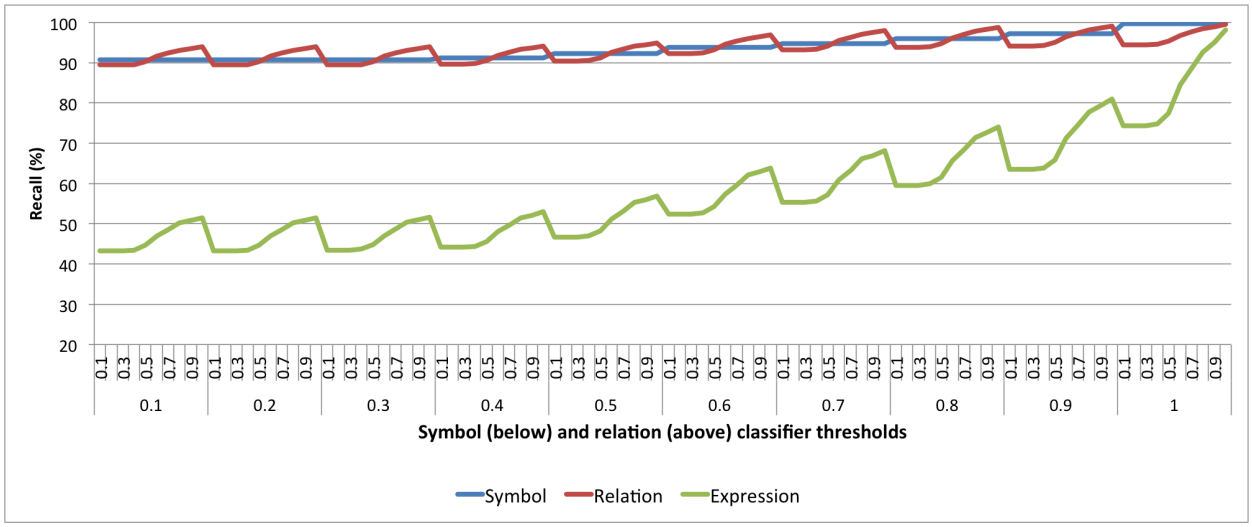


Figure 7.1: *Symbol, relation and expression level recall of the hypotheses graph generation step, for thresholds in the range* $[0.1 - 1.0]$. *For each symbol threshold value, values for the relation threshold are varied from* 0.1 *to* 1.0.

We can see in Figure 7.1 that even for the lowest threshold values about 90% of symbols and relations, and 40% of expressions are detected. We can also see that no considerable improvement is obtained, neither at symbol nor relation levels, for threshold values below 0.5. Using the maximum threshold values, almost 100% of the symbols are correctly identified (specifically, $99,75\%$). As for a symbol threshold value of 1 no stroke group is rejected, that recall value also corresponds to the percentage of symbols identified by the stroke grouping method. Similarly, for the maximum threshold values, almost all relations and expressions are identified ($99,45\%$ and 98.11%, respectively). This means that the expected symbols and relations of each expression are mostly present in its corresponding hypotheses graph. In an ideal case, we would like to set threshold values as

high as possible to guarantee the highest possible recall values. However, higher threshold values also mean a more complex parsing process. The best threshold values are then selected considering their applicability during parsing.

To gain insight about the size of the hypotheses graph relative to an input expression, we measured the ratio between the number of symbol and relation hypothesis labels and the number of symbols and relations of the expression, respectively. For instance, for an expression composed of 5 symbols and 4 relations, a ratio of 2 for both symbols and relations means that its corresponding hypotheses graph contains about $5 \times 2 = 10$ symbol labels and $4 \times 2 = 8$ relation labels. Figure 7.2 shows the mean of the ratios calculated over threshold values in the range [0.1-0.9]. We avoid showing the ratios for threshold values of 1, as they are so high that make difficult the analysis of the ratios for lower values. Nonetheless, a chart that includes the ratios relative to threshold values of 1 can be seen in Appendix B.

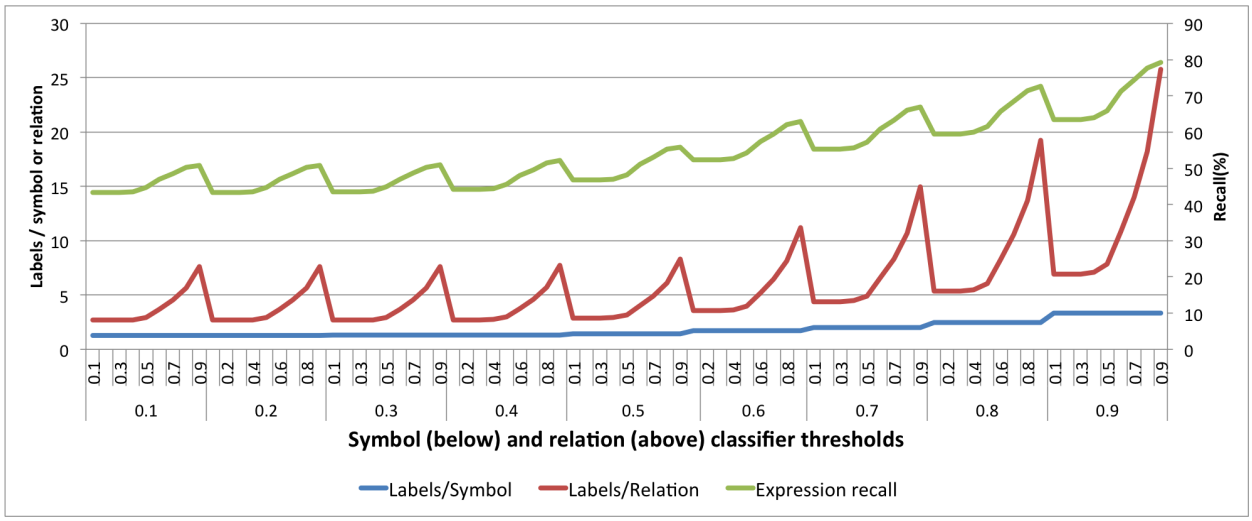


Figure 7.2: *Mean of symbol and relation hypothesis labels per symbol and relation, respectively. The left hand axis indicates the number of labels per symbol and relation. The right hand axis indicates the percentage of recognized expressions.*

We can see in Figure 7.2 that for thresholds in the range $[0.1 - 0.4]$ the number of symbol and relation labels is almost constant. This indicates that, for that range of values, the hypotheses graphs contain almost the same hypotheses, which also explains the almost constant symbol and relation recall (see Figure 7.1) for that range of values. For thresholds above 0.4, we see a slow increment in the **Labels/Symbol** ratio, but a high increment in the **Labels/Relation** ratio. This difference in the ratio increments is natural, as in the relation hypotheses generation step we evaluate all pairs of symbol hypotheses to determine the existence of relations between them (generating a potential quadratic relation between the number of relations and symbols of the graph).

Figure 7.2 also gives hints about how to select the best threshold configuration. For instance, we see that for symbol threshold $t_{symb} = 0.7$ and relation threshold $t_{rel} = 0.9$, the recall is about 65%. We also see that a similar recall is obtained for $t_{symb} = 0.9$ and $t_{rel} = 0.5$. A comparison of the two configurations suggests that the second configuration would often produce a more efficient recognition method because, in average, it generates less labels per expression. In general, we could say that configurations with high relation thresholds can be replaced with configurations with lower relation thresholds and higher symbol thresholds to obtain similar recall rates but more efficient parsing.

7.2.2 Graph parsing

Similarly to the hypotheses graph evaluation, we used the validation set to analyze the parsing algorithm recall rates over a range of symbol and relation thresholds. We defined the maximum symbol and relation thresholds as 0.98 and 0.85, respectively; as parsing large expressions with thresholds larger than those takes much time to be considered in a real application. Along this evaluation, we set the same weight for the symbol and relation costs, that is, $\alpha = 0.5$ and the tree extraction threshold to 0.1. Figure 7.3 shows the expression recall obtained by the graph parsing method and the corresponding recall obtained by the hypotheses graph generation step (recall that they indicate the maximum recall rates that the parsing algorithm could achieve). The figure shows that for thresholds higher than $t_{symb} = 0.9$ and $t_{rel} = 0.8$, no considerable improvements are obtained. For experimentations on the test set, we nonetheless set $t_{symb} = 0.98$ and $t_{rel} = 0.85$, as those threshold values allow to keep more hypotheses that could be useful during parsing of unseen data (better generalization). We can also see that for the best threshold values there is a gap of about 40% between the hypotheses graph and the graph parsing recall. This difference indicate that the current recognition technique has potential to obtain much better recognition rates just by improving the graph parsing technique.

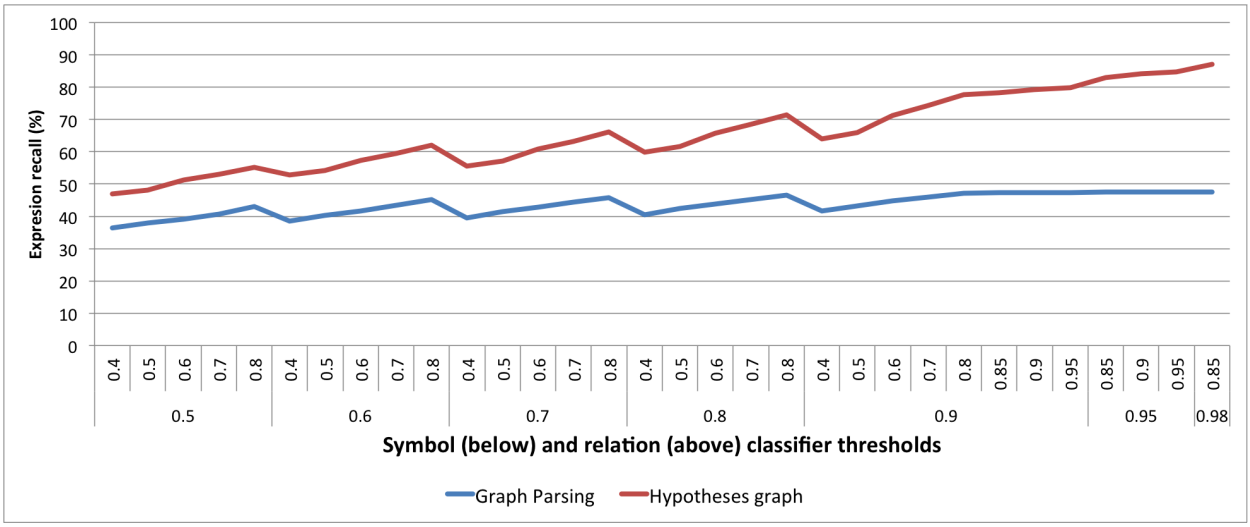


Figure 7.3: *Symbol, relation and expression recall obtained at graph parsing and hypotheses graph generation stages, for different symbol and relations thresholds values.*

Using the best symbol and relation thresholds, we evaluated the cost function with different tree extraction threshold and cost weighting values (the higher the symbol cost weight, the higher the influence of symbol classification interpretations). Through this evaluation, we set the tree extraction threshold to 0.1. For the cost weighting, we compared the obtained cost function rates with the rates obtained by the fuzzy score. Table 7.1 shows the results regarding this experimentation. Results show that to obtain the best recall rates, both costs should be considered (avoiding weights of zero or one). We set the cost wighting value to $\alpha = 0.4$, as the best expression recall was obtained for such value. We can see that recall rates at symbol level are stable for symbol cost weights above 0.2. The relation recall presents a variation similar to the symbols, probably because it depends directly on the symbol segmentation recall.

Table 7.1: *Comparison of scoring function using the validation set of CROHME-2014.*

Score function	segmentation recall	classification recall	tree rel. recall	expr. recognition recall
cost ($\alpha = 0.0$)	48.21	37.47	17.84	6.11
cost ($\alpha = 0.2$)	91.75	86.21	74.22	46.42
cost ($\alpha = 0.4$)	92.96	87.37	76.69	47.68
cost ($\alpha = 0.5$)	93.83	88.17	77.27	47.58
cost ($\alpha = 0.6$)	93.36	87.66	76.84	46.84
cost ($\alpha = 0.8$)	93.26	87.49	76.19	46.32
cost ($\alpha = 1.0$)	92.69	86.94	38.36	17.89
fuzzy	91.96	85.82	64.54	40.63

7.3 Results

For better understanding of the recognition performance over the test set, we measure the hypotheses graph generation performance using the same range of threshold values used for the validation set. Figures 7.4 and 7.5 show the results regarding that evaluation. As the symbol and relation classifiers were optimized using the validation set, it is natural that the recall rates obtained on the test set are lower than the recall rates obtained on the validation set. Specifically, we see that for the lowest threshold values, the recall rates decreased about 5% for symbol and relation levels and 12% for expression level. This difference is less significant as the threshold values increase. Regarding the number of labels per symbols and relations, the ratios obtained on the test set do not present considerable differences in relation to those obtained in the validation set, which suggests stable graph complexities on these data.

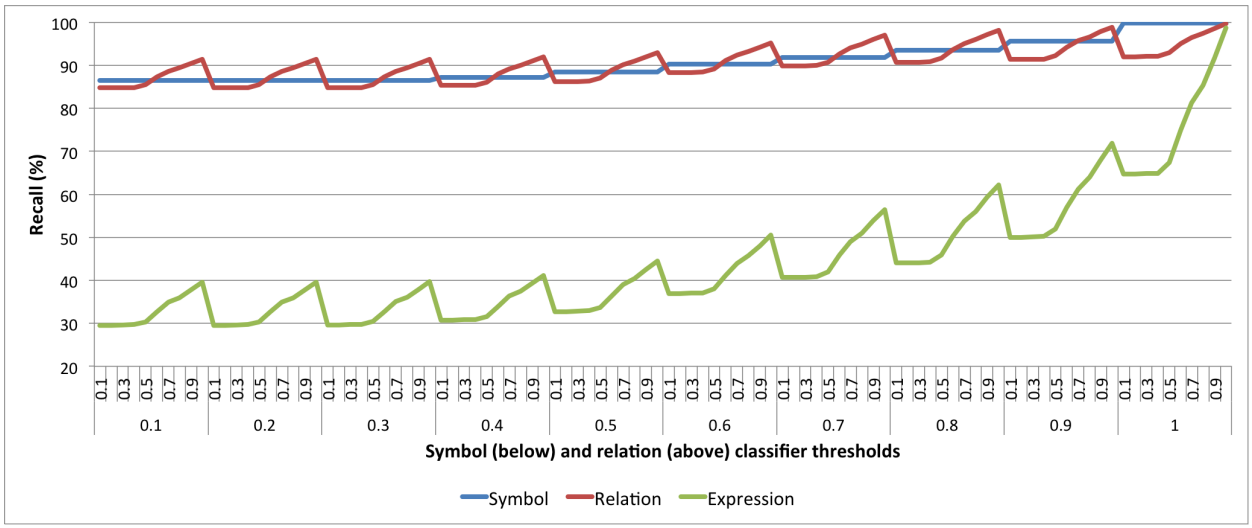


Figure 7.4: *Symbol, relation and expression level recall of the hypotheses graph generation step, for thresholds in the range* $[0.1 - 1.0]$. *This results are calculated over the test set expressions.*

Table 7.2 compares the expression level recognition rates of our system and systems that participated in the CROHME competition. It includes the percentage of expressions that were recognized with up to three errors (in symbol or relation levels). Considering that for the selected parameter values the hypotheses graph generation method identified 78.40% of the expressions correctly, much better recognition rates are still possible by improving the graph parsing method (this observation is coherent with the comparison of the corresponding hypotheses graph and graph parsing rates

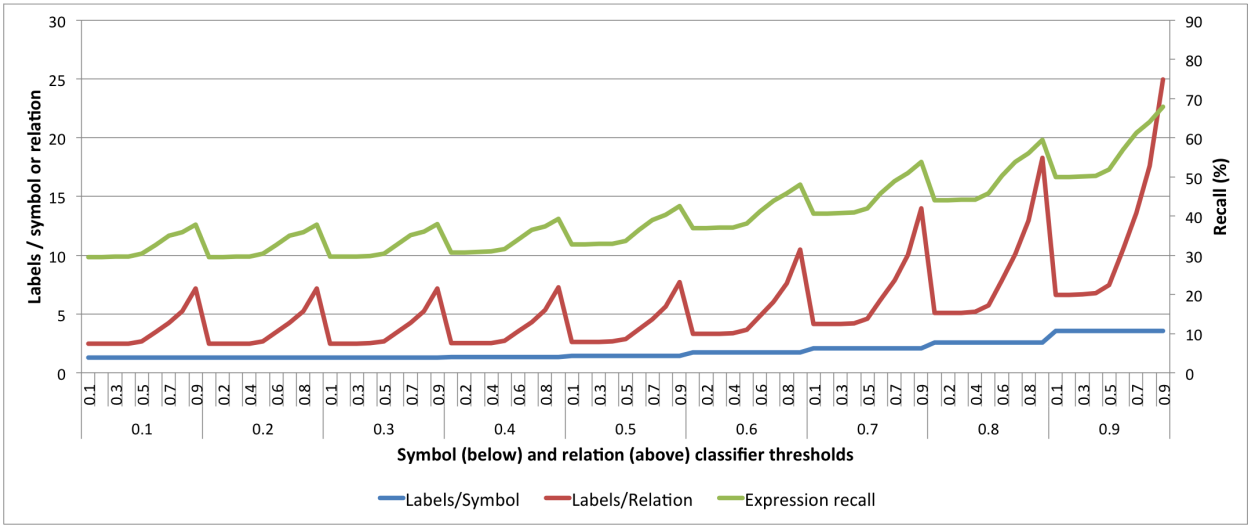


Figure 7.5: *Mean of symbol and relation hypothesis labels per symbol and relation, respectively, over the test set expressions. The left hand axis indicates the number of labels per symbol and relation. The right hand axis indicates the percentage of recognized expressions.*

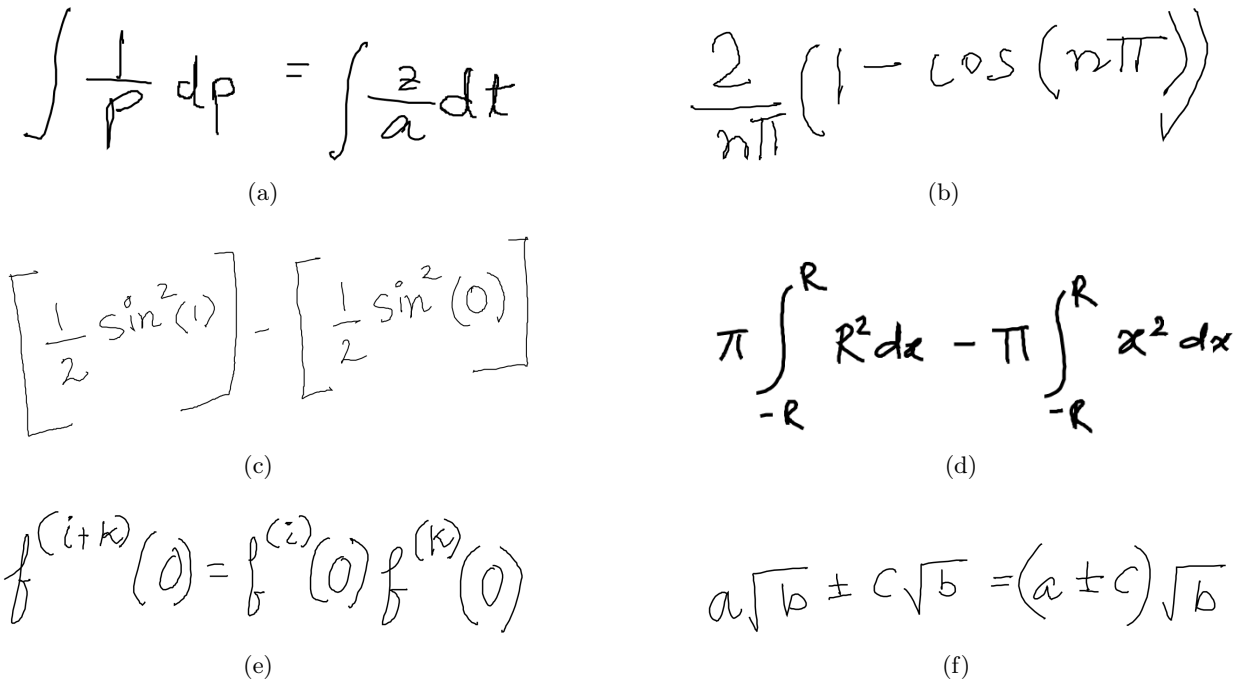
obtained for the validation set).

Table 7.2: *Expression level comparison of our method with systems of the CROHME-2014 competition.*

System	correct	#errors ≤ 1	#errors ≤ 2	#errors ≤ 3
I	37.22	44.22	47.26	50.20
II	15.01	22.31	26.57	27.69
III	62.68	72.31	75.15	76.88
IV	18.97	28.19	32.35	33.37
V	18.97	26.37	30.83	32.96
VI	25.66	33.16	35.90	37.32
VII	26.06	33.87	38.54	39.96
ours	**33.98**	**43.10**	**47.56**	**49.29**

To have a better understanding of the potential of our method, we analyzed the expressions that have been recognized correctly. The expressions include samples that have up to 39 strokes, including all the six relation types. Figure 7.6 shows examples of those expressions. It can be seen that even expressions that have ambiguous symbols and irregular arrangements of symbols are correctly recognized. We also found that some symbols or relations that were missrecognized in isolated experiments, were correctly recognized at expression level. For instance, in isolated relations classification (Section 5.3), the relation between the subexpressions "$\frac{1}{2}$" and "$\sin^2(1)$" of Figure 7.6(c) was missclassified as *superscript*, but when recognizing the whole expression, the best interpretation includes the right relation (*horizontal*).

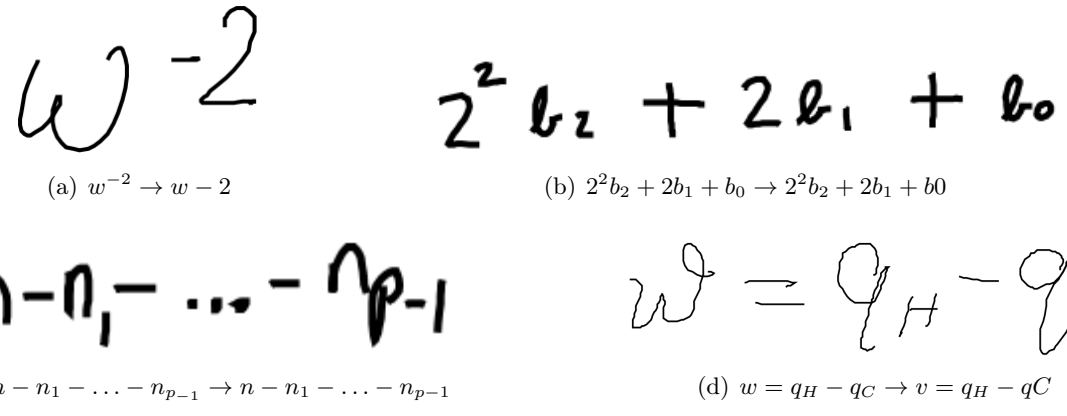
Table 7.2 indicates that 15% of the expressions would be correctly recognized if we disregard up to three errors. Figure 7.7 shows some of those expressions. Most of the error could probably be corrected by optimizing the symbol and relation classifiers as described in Sections 4.4 and 5.4, respectively. However, some cases are difficult to solve, even for humans. For instance, in the expression of Figure 7.7, the system interprets the relation between p and -1 as *horizontal*, that is, $p-1$, but the relation is rotulated as *subscript* in the ground-truth, that is p_{-1}. Given the handwritten expression, we could consider that the system's output is a more likely interpretation than



(a) (b) (c) (d) (e) (f)

Figure 7.6: *Examples of handwritten mathematical expressions that have been correctly recognized by our method.*

the ground-truth's.



(a) $w^{-2} \rightarrow w - 2$ (b) $2^2b_2 + 2b_1 + b_0 \rightarrow 2^2b_2 + 2b_1 + b0$

(c) $n - n_1 - \ldots - n_{p_{-1}} \rightarrow n - n_1 - \ldots - n_{p-1}$ (d) $w = q_H - q_C \rightarrow v = q_H - qC$

(e) $bag_1 \rightarrow bay1$ (f) $a_0 + 3a_1 + 9a_2 + 27a_3 = 0 \rightarrow a_0 + 3a_1 + ga_2 + 27a_3 = 0$

Figure 7.7: *Expressions recognized with a few errors. For each expression, its ground truth and the system's output is showed as: ground truth → system's output.*

Table 7.3 compares our system's symbol segmentation and classification recall and spatial relation detection rates with those of the CROHME competition. At symbol classification, we make a parallel between our system's performance and the best system of the competition. For this purpose, we show the symbol hypotheses classification results of both systems in table Table 7.4 (this results were introduced in Section 4.3.2). It is interesting to note that while in the symbol hypotheses classification task our system performed almost as good as System III, in the expression recognition

task, System III outperforms our system by about 13%. Furthermore, System III outperforms all the systems with a considerable difference when recognizing complete expressions.

Table 7.3: *Object level comparison of our method with systems of the CROHME-2014 competition.*

	segmentation		classification		tree rel.	
System	recall	precision	recall	precision	recall	precision
I	93.31	90.72	86.59	84.18	84.23	81.96
II	76.63	80.28	66.97	70.16	60.31	63.74
III	98.42	98.13	93.91	93.63	94.26	94.01
IV	85.52	86.09	76.64	77.15	70.78	71.51
V	88.23	84.20	78.45	74.87	61.38	72.70
VI	83.05	85.36	69.72	71.66	66.83	74.81
VII	89.43	86.13	76.53	73.71	71.77	71.65
ours	**88.04**	**92.91**	**80.92**	**85.39**	**61.20**	**65.18**

Table 7.4: *Comparison of our method and System III for symbol hypothesis classification, using the CROHME-2014 test set.*

	Without Junk	With Junk		
System	Rec. rate	Rec. rate	TAR	FAR
Ours	**89.21**	**90.50**	**92.53**	**3.93**
System III	91.04	85.54	87.12	10.39

From the systems that participated in the competition, two of the best, System I and System III, use parsing techniques as part of the recognition process. System I corresponds to the work of Álvaro et al. (2012), described in Section 2.3 of this thesis. It uses a string grammar and a CYK-based parsing algorithm. The best system, System III, corresponds to the commercial system *MyScript*[3]. According to Mouchère et al. (2014), *MyScript* has been optimized over hundreds of thousands of equations, property of the company and not publicly available. The optimization of the whole system using such dataset should explain part of the difference between the *MyScript's* performance and the rest of the systems. Additional information that System I and III included in their systems, and that our system did not considered, consists of statistics about the application of the rules of their grammar.

To gain insight about the most difficult structures, we extracted the spatial relations for which our system obtained more recognition errors. In this case, we consider symbol-to-symbol spatial relations, where a relation is identified by the relation identity, and the identity of symbols at both ends. Table 7.5 shows the ten relations that presented more errors, sorted by the number of errors, in decreasing order. The table also includes the corresponding total number of samples of each relation and the percentage of the samples that were missrecognized. Some of the structures are particularly difficult due to the ambiguity at symbol level. For instance, our system often missrecognizes "$\times$" as "x" and the trigonometric function "sin" as tree letters, like "s", "i" and "n", related by an *horizontal* relation.

[3]http://www.myscript.com/

Table 7.5: *Spatial relations between symbols (of the test set) that presented more errors. Each relation is identified with a printed representation. The relation identity is implicit by the relative symbol positions.*

Relation	# errors	# samples	% errors
$\frac{1}{-}$	28	133	21.05
$= -$	26	91	28.57
$x+$	26	120	21.67
$x\times$	24	24	100
$\times x$	24	24	100
$(x$	21	108	19.44
$\sin($	20	42	47.62
$\frac{-}{2}$	19	81	23.46
$\sqrt{-}$	19	29	65.52
$\frac{-}{n}$	18	37	48.65

7.4 Discussion

The similar ratios between the number of hypotheses graph labels and symbol and relations of the test and validation sets suggest that the proposed method maintains a consistent computational cost on both sets. Furthermore, in terms of computational cost, the method is flexible as the stroke partitions search space can be tuned through the symbol and relation thresholds. In terms of effectiveness, the method obtained 4% less segmentation recall in the test set, relative to the validation set. That difference is propagated to the symbol classification, relation and complete expression recall values. This explains, at some extent, the 14% decrease in terms of complete expression recall in the test set (a single error at symbol or relation levels is translated into a missrecognized expression).

The proposed recognition technique obtained better recognition rates than several state of the art techniques, but it still has room for improvement. The hypotheses graph generation method detected 78% of the expressions. The current parsing algorithm was able to correctly recognize 33.98% of the expressions, which leaves a gap of almost 45% in relation to the hypotheses graph rate. This suggests that further work should be directed to improve the parsing method.

A main limitation of the current graph parsing method is the lack of a statistical model. Statistical models have been successfully applied for strings parsing (Collins, 2003). They basically associate probabilities (calculated from parameters optimized over training data) to the application of production rules and rank trees according to the combination of the probabilities of their component rules. In the case of mathematical expression recognition, statistical models could indicate the probability of recognizing certain symbols or structures in particular subexpressions. Such information may help to solve ambiguities when handwritten data (like shape of symbols, relative positions between symbols or time related information) is not enough. For instance, statistical models could help to solve the problems of missrecognizing a symbol "$\times$" as "x" and the trigonometric function "sin" as tree letters, like "s", "i" and "n", related by an *horizontal* relation. In the first case, the symbol "$\times$" probably appears more frequently between a pair of numbers (or letters) and almost never alone; in the second case, the three symbols would probably appear more often as the trigonometric function "sin", rather than as three different letters. The statistical models of System III, optimized

over hundreds of thousands of expressions, and System I, optimized over the CROHME dataset, should partially explain the difference between the performance of those systems and ours.

Machine learning techniques work best when there is large quantities of data for parameter optimization. Having 101 symbol classes, and six relation types, the possible symbol-to-symbol relation types is $O(101^2 * 6)$. Although not all possible relations between symbols are allowed (for example, a symbol + hardly ever would be related with another symbol by a *superscript* relation), this number gives an idea of the large quantity of data required for training. The availability of larger publicly available datasets would allow to better optimize academic systems and have a more homogeneous comparison between them and commercial systems.

Chapter 8

A framework to build online handwritten mathematical expression datasets

8.1 Introduction

It is generally acknowledged that publicly available datasets with ground-truth data are essential when evaluating the performance of pattern recognition methods. First, weakness and strengths of different methods can be determined by testing them on a common dataset (Valveny et al., 2007). Second, public datasets allow reproduction of experiments to validate or negate the results (Lapointe, 2008). Third, the accessibility to such data enables contests which have proven useful for several fields (MacLean et al., 2011).

In the domain of recognition of handwritten mathematical expressions, the lack of publicly available datasets has been a main issue (Lapointe, 2008). Most of the systems developed to solve this problem have been evaluated on private expression sets (e.g. Awal et al., 2009; Matsakis, 1999; Rhee and Kim, 2009; Vuong et al., 2010). To cope with this problem, the CROHME dataset (Mouchère et al., 2014) has recently been built by merging parts of six datasets from different research groups. Although this dataset has about $10,000$ expressions, larger datasets are still needed.

The process of building handwritten mathematical expression datasets with ground-truth data is labor-intensive and error-prone (Wenyin and Dori, 1998). A large and expressive dataset should comprise a large variety and number of sample expressions, and ground-truthing them implies labeling thousands of symbols as well as their relationships individually. To avoid manual labeling of thousands of individual symbols in the sample expressions, part of the dataset creation process should be automated. A possible approach is to consider a set of model expressions annotated with ground-truth data and then automatically annotate samples that are obtained by transcribing the models. Such approach is used, for instance, in Hirata and Honda (2011); MacLean et al. (2011).

Part of the work of this project was devoted to develop techniques to automatize the creation of mathematical expression datasets. This chapter describes this effort and their corresponding results. Section 8.2 describes the main qualities a dataset must have in order to allow an effective evaluation. Section 8.3 outlines our framework to build mathematical expression datasets. The framework is based on an expression matching technique to automatically label mathematical symbols (Hirata and Honda, 2011). We also developed a system, called *ExpressMatch*, that implements

the proposed framework and manages the creation process. The system is described in Section 8.4. We used the implemented *ExpressMatch* system to built a dataset and released it as open-source. The results regarding such experimentation are given in Section 8.5.

8.2 Desired qualities of mathematical expression datasets

Several important qualities that should be fulfilled by testing datasets in order to allow effective performance evaluation of mathematical expression recognition methods are pointed out in the literature (Awal et al., 2010a; Lapointe, 2008; Lapointe and Blostein, 2009):

- **different levels of labeling** (Awal et al., 2010a): datasets should have labelings at stroke, symbol and expression levels, in order to allow evaluation of different faces of systems. For example, to evaluate symbol recognition rate (number of correctly recognized symbols over the total number of symbols), ground-truth of each symbol in the expression is needed, while for evaluating segmentation techniques, labeling at stroke level is needed;
- **multiple ground-truths at expression level** (Lapointe, 2008; Lapointe and Blostein, 2009): in some cases there are several equivalent ways to represent a given mathematical expression within a computer mathematical format. For example, considering LaTeX, x_0^2 can be represented as "x^2_0", "x_0^2","x^{2}_{0}", and "x_{0}^{2}"; all these representations should be accepted as correct;
- **subsets of mathematical expressions that meet some constraints**: many systems are designed to work within specific constraints (limited number of symbol classes, limited number of symbols in each mathematical expression, specific field of mathematics, and so on). Thus, mathematical expressions should be organized and classified under some established criteria. For each set of constraints, selecting only mathematical expressions satisfying the constraints should be possible;
- **statistical representativity** (Lapointe, 2008; Lapointe and Blostein, 2009): distribution of mathematical expression types within a specific domain should be considered in a dataset, in order to allow a good approximation of the performance of systems in real scenarios;
- **public availability** Lapointe (2008); Lapointe and Blostein (2009): evaluation and comparison of different methods on a common dataset would facilitate assessment of weakness and strengths of each system.

These qualities were considered to define our framework and the *ExpressMatch* functionalities.

8.3 The dataset creation procedure

Hirata and Honda (2011) proposed an expression matching technique to automatically label handwritten mathematical symbols. In this technique, the labels (ground-truth) of symbols of a mathematical expression (model) is transferred to their corresponding symbols on a second expression (transcribed expression). Figure 8.1 shows an expression matching example.

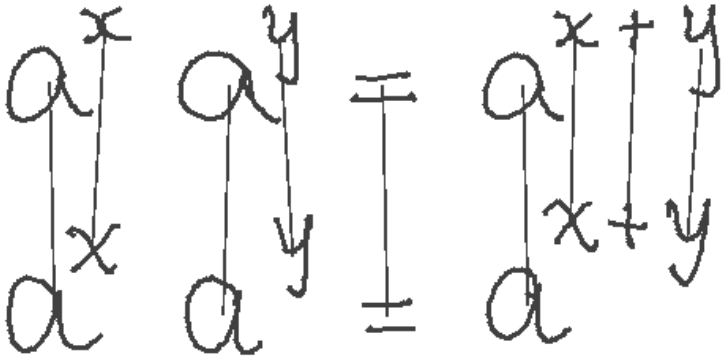


Figure 8.1: *Expression matching example. Vertical lines indicate the matched symbols.*

In this thesis, we improved the expression matching technique proposed in Hirata and Honda (2011) by including new features to calculate the matching cost, and by making a thorough analysis of the matching results on a larger dataset. With these improvements, we increased the correct symbol assignment rate from 92% to 99%. A detailed description of the improved expression matching method is given in Hirata and Julca-Aguilar (2015).

Considering the expression matching technique, we propose the following general procedure to build mathematical expression datasets:

- Creation of model expressions. This step consists on handwriting mathematical expressions and attaching their corresponding ground-truth. In this process, all symbols of each expression must be segmented.
- Transcription of model expressions. In this step, volunteers are invited to transcribe the model expressions. As a result, we have a number of expression samples of each model expression.
- Symbol segmentation of the transcribed expressions. As in the case of model expressions, all symbols of the transcribed expressions must be segmented.
- Expression matching. Symbols of each transcribed expression are mapped to their corresponding symbols on the model.
- Ground truth transferring. Once the symbols are matched, the ground-truth may be transferred from the model expressions to the transcribed ones. This transferring process may include ground-truth at different levels of granularity.

8.4 The *ExpressMatch* system

The *ExpressMatch* system implements the methodology described in Section 8.3. The main features of this system are highlighted next:

- the set of model expressions define a corpus. Since models are input manually, the system is very flexible with respect to types of corpora that can be created;
- at expression level, more than one ground-truth can be attached to each model expression, via textual input;

- expressions can be associated to user-defined categories. This feature is useful to select only expressions of a given category;
- user registration and management controls which and how many times a model expression has been transcribed by each `writer`;
- time gap between strokes is taken into consideration to perform segmentation at writing time. Whenever the time gap between two strokes is larger than a given threshold, the system considers that a new symbols is being written. Although this mechanism adds some restriction to the writing style, it has been observed empirically that `writers` have no difficult to adapt themselves to the rule;
- symbols in transcribed expressions are automatically labeled by assigning them to the corresponding symbols in the model expression, based on the expression matching approach proposed in Hirata and Julca-Aguilar (2015);
- matching between symbols in model and transcribed expressions can be visually verified and interactively corrected if necessary;
- both model and transcribed expressions can be added incrementally. In addition, data gathered in different machines can be combined into a same database. These features make possible an incremental creation of large datasets;
- subsets of expressions can be selected and exported as XML format files. For instance, it is possible to select only expressions of a specific category, or expressions with the number of symbols within a given interval, or expressions written by a specific group of writers;
- it is possible to extract symbol samples in order to create symbol datasets. The set of symbols obtained from the expressions will better resemble the way they are naturally written within mathematical expressions than when they are written in a isolated way, and this fact may be relevant for the development of symbol recognizers for mathematical expression recognizers.

8.4.1 *ExpressMatch* architecture

ExpressMatch consists of six main components, as shown in Fig. 8.2: (1) time-based segmentator (**TBS**) for segmentation of symbols at writing time; (2) model expression capturer (**MC**) for capturing model expressions and annotating them with ground-truth data; (3) instance capturer (**IC**) for capturing transcribed expression (instances); (4) expression matching-based labeler (**EMBL**) for labeling of instance expressions by matching them to the respective model expression; (5) labeling editor (**LE**) for interactive verification and correction of labelings; and (6) importer/exporter of dataset (**IED**) for importing/exporting data.

Two kinds of users are allowed: `administrators` and `writers`. `Administrators` can define models, evaluate labeling results and use the import/export functionality by interacting respectively with the MC, LE, and IED user interfaces. `Writers` can only write instance expressions by interacting with the IC interface. `Administrators` are also `writers`.

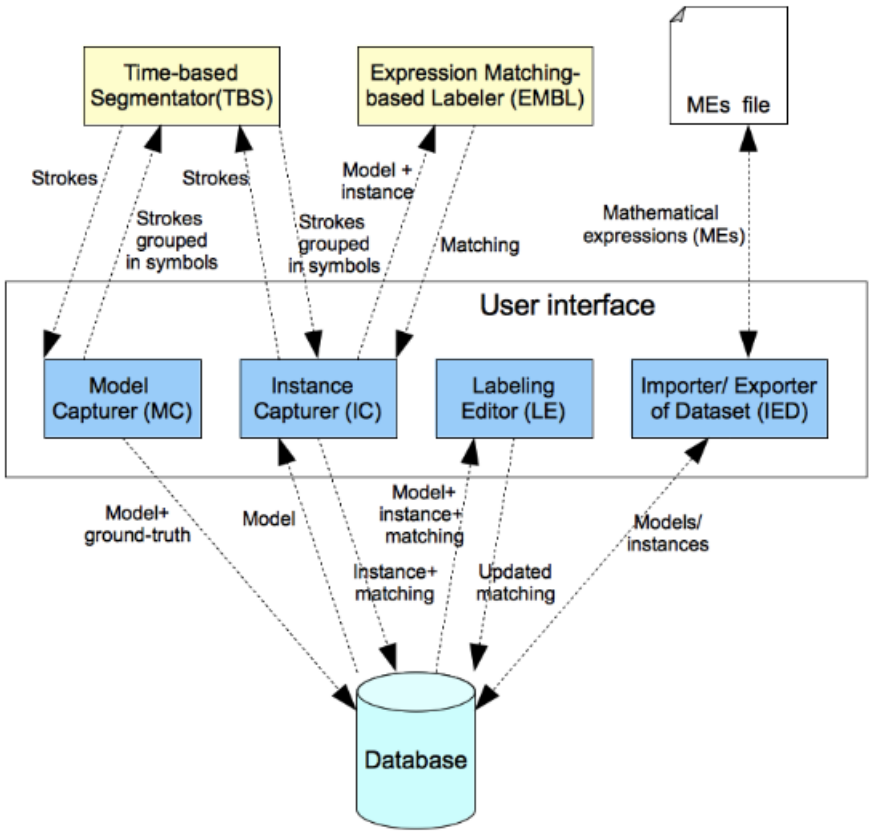


Figure 8.2: ExpressMatch *architecture.*

Time based segmentator (TBS)

Given a pair of consecutive strokes, TBS considers them as being part of the same symbol if the temporal gap between the end of the first and the beginning of the second stroke is no longer than a predefined threshold.

Model capturer (MC)

Model expressions and their corresponding ground-truth data can be input through the MC interface. Figure 8.3 shows a snapshot of the interface when the model expression $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ is being input. Symbols are segmented during writing: MC sends strokes and their timing information to TBS that returns the strokes grouped as symbols (see Section 8.4.1). MC indicates which strokes are being considered as belonging to a particular symbol by displaying a bounding box around the set of strokes. The `undo` and `delete` functionality allow correction of possible wrong segmentations. `Undo` eliminates the last written stroke, while `delete` removes an entire symbol (any set of strokes related to a bounding box).

To help organization of mathematical expressions, expression classes can be defined and each model expression can be assigned to any of those classes. Figure 8.3 shows that the written model is being assigned to `Arithmetic` category.

Ground-truth data at expression level can be input as textual information through the text area above the written expression. Figure 8.3 shows ground-truth data in LaTeX format. Additional ground-truth can be assigned using the `append` button, placed below the text area. At symbol level, the π button allows assignment of ground-truth data: labels for each of the symbols can be

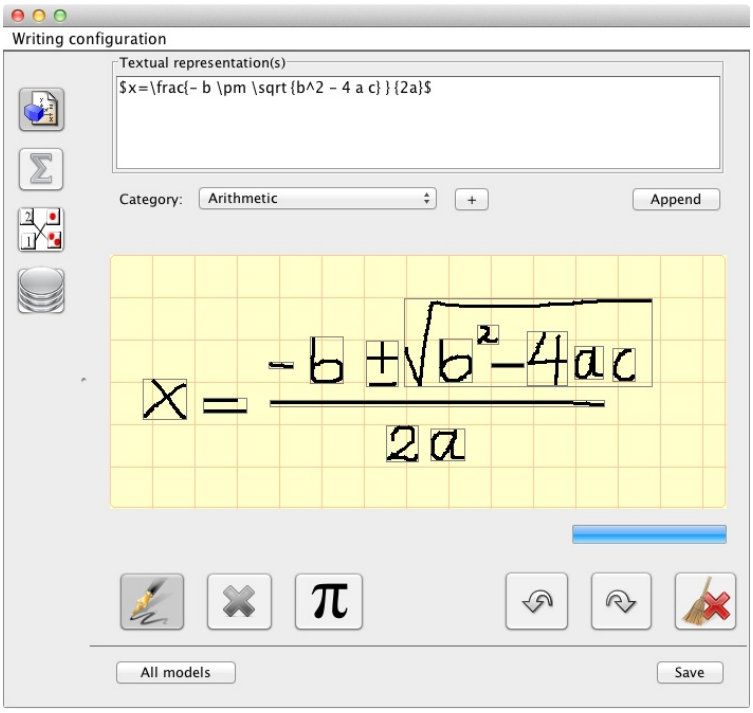


Figure 8.3: *Model collector: expression* $x = \frac{-b \pm \sqrt{b^2-4ac}}{2a}$ *is being defined as a model.*

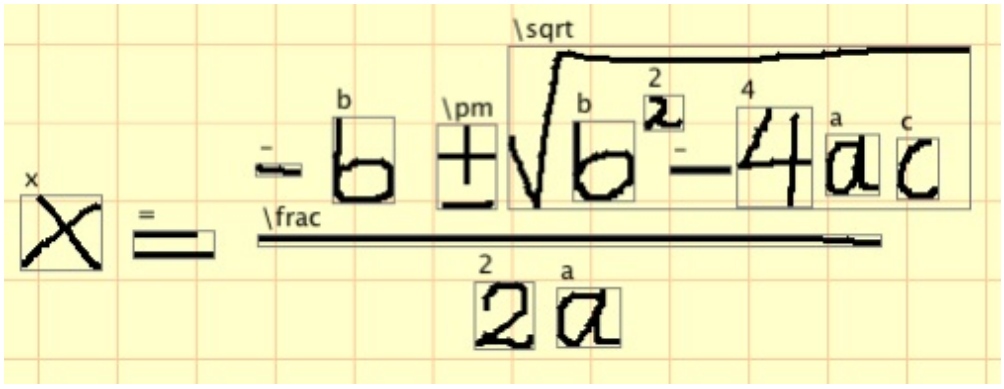


Figure 8.4: *Model expression of Figure 8.3 with all its symbols labeled.*

manually entered, being subsequently shown in the superior corner of each symbol, as shown in Figure 8.4.

Instance capturer (IC)

This module is the interface used to capture instances of model expressions. Model expressions are randomly selected from the set of predefined model expressions and displayed in the superior part of the interface. The system controls which expressions have already been transcribed by each registered user. Figure 8.5 shows the interface displaying the model expression $cos\theta = \frac{x}{\sqrt{x^2+y^2}}$ and its transcription below it. As the MC component, IC also interacts with the TBS component to get the symbols segmented at writing time. Segmentation is indicated with a bounding box around each symbol and can be corrected with `undo` and `delete` buttons in a similar way to MC interface. The system issues a warning message if the number of symbols in the model and transcribed expressions differs.

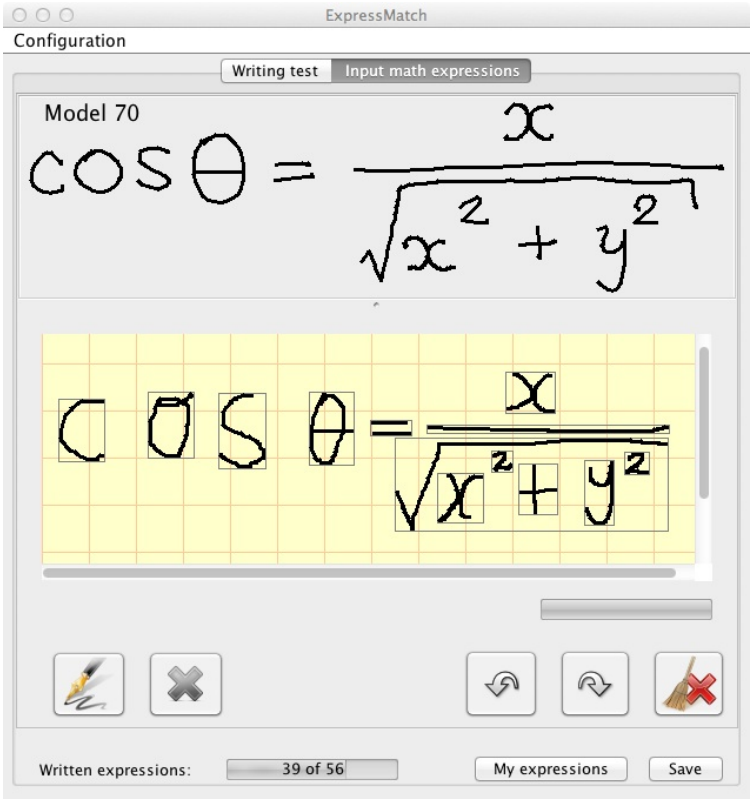


Figure 8.5: *Instance capturer: model expression is shown at the top, to indicate users what instance they have to write.*

Expression matching based labeler (EMBL)

Symbols of an instance expression are labeled automatically, based on the method described in Hirata and Honda (2011). Each time an instance expression is input, EMBL computes a matching between that instance and its corresponding model, finding a one-to-one correspondence between unlabeled symbols in the instance and labeled symbols in the model expression. The correspondence determines the label of the symbols in the instance expression.

Labeling editor (LE)

This interface allows `administrators` to interactively evaluate and correct symbol labeling. Model expressions and the list of instances for the selected model are shown on the left side of the interface, and the matching between the selected model-instance pair is displayed on the main panel (see Figure 8.6). The computed correspondence between symbols in a model and instance expressions are displayed graphically by line segments linking them. To correct a matching, an extremity of the line segment can be interactively placed over the correct matching symbol.

For expressions with a large number of symbols, line segments may appear cluttered, making visual verification difficult. To facilitate visual inspection in such cases, it is possible to display groups of non intersecting line segments (see Figure 8.7).

Importer/exporter (IED)

This component allows model and instance expressions to be imported/exported. Thus, expressions collected in different machines can be joined into a single central dataset. In addition, the

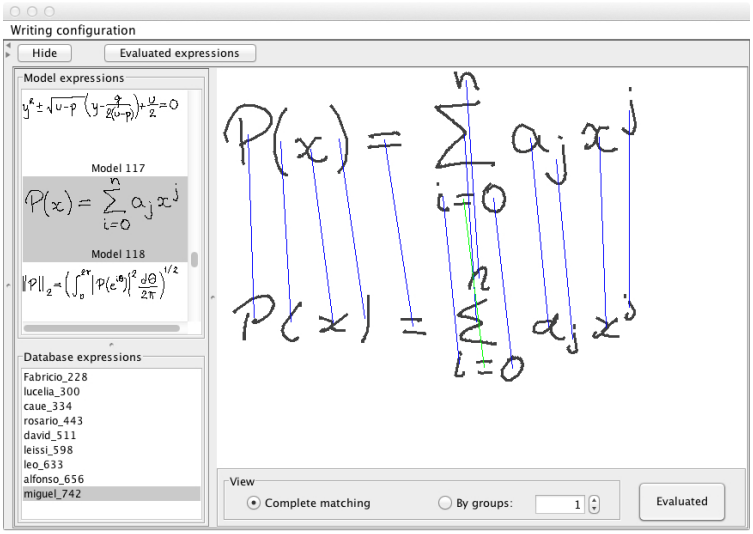


Figure 8.6: *Labeling editor: labeling result is shown as a matching between labeled symbols of model expression (at the top) and unlabeled symbols of the instance expression (at the bottom).*

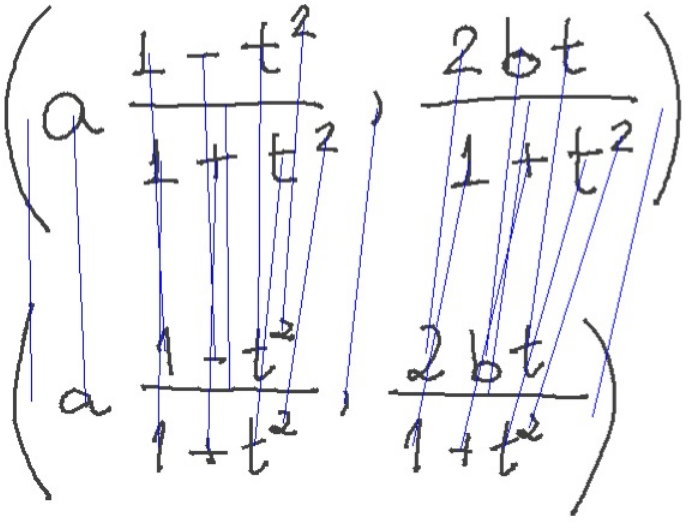
whole dataset or a subset of it can be exported as XML format files. Subsets with expressions belonging to specific categories, or written by specific users, or having a specific number of symbols or classes of symbols can be selected for exportation. Datasets with isolated symbol samples can also be exported.

8.5 Results

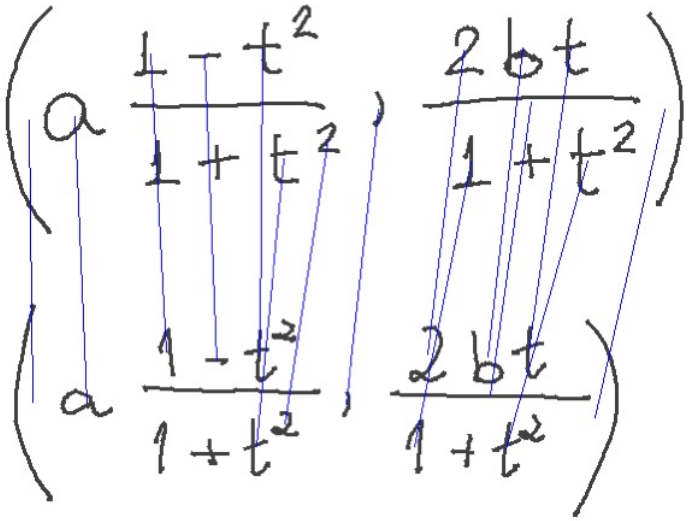
A large dataset consisting of fifty six model expressions has been created in order to evaluate the functionalities and usability of *ExpressMatch*. A total of 25 writers, with background in Engineering or Computer Science, volunteered to transcribe the expressions. Since writing may be a time consuming task, users were asked to write as many instances as their time constraint allowed. For inputting the expressions, an HP tablet PC was used. The average time spent by a user to enter all 56 expressions was about one hour. Through all the collecting process, the threshold used in TBS (for segmentation of symbols) was set to 500 milliseconds, a value that was determined experimentally.

A total of 926 instances were collected. From these, 16 (or 1.7%) were discarded due to segmentation or semantic error. After automatic matching between each of the 910 instances and respective models, they have been visually verified, and incorrect symbol assignments have been manually corrected. The verification and label correction process took about 5 hours.

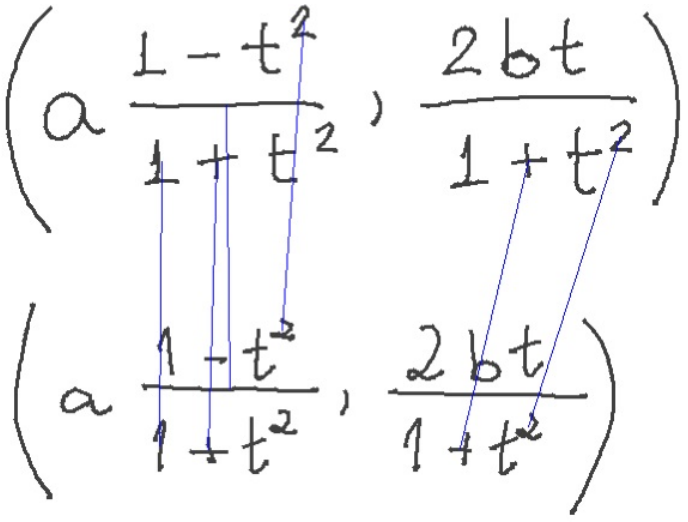
In addition, writers were asked to give feedback on how much writing is affected by using the rule described in Section 8.4.1. Overall, the evaluation is that users had rapidly adapted themselves to the writing rules, which was not considered a severe restriction.



(a)



(b)



(c)

Figure 8.7: *Example of matching with many line segments: (a) with all segments, (b) and (c) two groups of non intersecting segments.*

Chapter 9

Conclusions and future work

In this thesis, we have studied the online handwritten mathematical expressions recognition problem and proposed a new recognition technique. The problem involves three main tasks: symbol segmentation, symbol classification and structural analysis. We have developed techniques to treat each of the tasks and integrated the techniques into a single parsing process. We evaluated our approach over the CROHME-2014 dataset (Mouchère et al., 2014).

In addition to the new recognition technique, we developed a framework to automatize the creation of handwritten mathematical expression datasets. We have developed a system that implements the framework and built a dataset using such system. Both the system and the dataset have been released as open source.

9.1 Symbol segmentation and classification

Chapter 4 describes our work devoted to treat the symbol segmentation and classification problems. In our mathematical expression recognition approach, we do not compute an explicit symbol segmentation, nor symbol classification. Instead, we compute a set of stroke groups that may represent symbols and attach labels that indicate likely identities to each group. The calculated symbol hypotheses are then used by our parsing algorithm to compute the final segmentation and classification result.

We proposed a method to generate the stroke groups based on the proximity of strokes. A limitation of the method is the assumption that symbols contain a maximum number of strokes. Future work could include taking off that constraint without decreasing the current recall and precision rates. Also, it would be interesting to apply machine learning techniques that allow to extract scores to measure the likelihood of considering a stroke group as a symbol (without considering a specific class) based on spatial and time features. The scores could be combined with those generated by symbol and relation classifiers during the parsing process.

When recognizing symbols of mathematical expressions, the neighborhood of an evaluated symbol provides useful information to solve ambiguities. Symbols of an expression are not isolated, which means that strokes that do not belong to an evaluated symbol may cross over it, or be placed close to it. The sizes of symbols of a same expression generally maintain a relative scale that help to solve ambiguities. For instance, the symbol "." in an expression "2.3" could look like a "0", or "o" if we had a *close-up* of it, but it would almost always look as a point if we have a view of the complete expression. While most approaches treat the classification problem considering symbols

as isolated objects, we make use of the existent information of the context of each symbol:

- We proposed features that capture relative scale information and neighboring strokes information of an evaluated symbol hypothesis. The features help to differentiate symbols with similar shape as well as to filter out false symbols.
- When considering only true symbol hypotheses, the proposed features obtained recognition rates comparable to the best results of the CROHME competition. These results suggest that, although the features extracted from a symbol's neighborhood have a high variance (as they depend on where the symbol is placed), the variance does not affect the effectiveness.
- When true and false symbol hypotheses are considered, the proposed features overcame other approaches. A main advantage of the good discrimination between true and false symbol hypotheses is that it enables the development of a more efficient parsing, as more wrong interpretations are filtered out by the classifier.

9.2 Relation classification

Chapter 5 described our work related to the relation classification problem:

- Category vectors, that describe the position of symbols relative to a baseline, increment the recognition rate of geometric and histogram features in about 3% in both cases. Although we calculated the category vectors manually, they could also be calculated using machine learning techniques. For instance, a mathematical expression could be divided into baselines, and, for each baseline, the positions of the symbols relative to its baseline could be extracted. Using the extracted positions, the maximum-likelihood estimation could then be used to calculate the mean of the position of each symbol class relative to a baseline. The automatic calculation of category vectors is an interesting topic for future work.
- A subexpression may consists of a single symbol or several symbols arranged in arbitrary structures. Most previous works used only bounding boxes of subexpressions to classify relations. We showed that including features extracted from adjacent dominant symbols of the subexpressions improves the recognition rate. Further work will include exploring other ways to exploit information extracted from the evaluated subexpressions.

In the symbol and relation classification problems we have proposed features that aim to capture the shape of symbols and relations, respectively, as images (or histograms). We have seen that these features obtained performance comparable to other features proposed in the literature. An advantage of the image-based features is the fact that they provide *complete* shape descriptions of objects, which may provide information not captured by handcrafted features[1]. At this regard, deep learning techniques, as convolutional neural networks, have shown to take more advantage of those features than other types of neural networks (Bengio et al., 2013; Lecun et al., 1998). The use of deep learning techniques to process the image-based features proposed in this thesis may provide considerable improvements on the symbol and relation classification problems.

[1] This point is also a motivation of deep learning approaches for patter recognition (Bengio et al., 2013; Lecun et al., 1998)

9.3 Graph parsing

A parsing technique is composed of a grammar, that defines a language to be recognized and a parsing algorithm that determines whether a given input belongs to the language. Constraints on the grammar often allows more efficient parsing, but reduce its generality. Chapter 6 described our graph parsing technique for recognition of mathematical expressions and Chapter 7 reported the evaluation of the technique.

The graph grammar proposed in this thesis provides a more natural model to represent mathematical expressions than other grammars reported in the literature. Structures that include multiple relation types are represented as labeled edges, without the need of defining compositions of rules, as it is done in string grammars. Our graph grammar does not assume specific structures on graphs of its production rules (as it is the case in the approach of Celik and Yanikoglu (2011)) which allows the grammar to be easily extended to recognize a variety of structures.

The main problem of using graph parsing techniques is the computational cost (Flasiński and Jurek, 2014). This issue has made several authors to introduce constraints into their proposed grammars and parsing algorithms (Flasiński and Jurek, 2014). To cope with the parsing complexity, our parsing algorithm only considers stroke partitions derived by a hypotheses graph calculated using symbol and relation classifiers optimized over training data. In Chapter 7, we saw that the hypotheses graph generation method identifies 78% of the test set expressions. Better rates can be obtained by improving the symbol and relation classifiers as described above.

A major topic for future research is how to improve the parsing algorithm effectiveness. The expression recall obtained by the current parsing algorithm is 33.98%, which leaves a gap of almost 45% in relation to the recall obtained by the hypotheses graph generation stage. Introducing a statistical model is a first method to explore in order to improve the parsing algorithm effectiveness. An additional topic for future research is the optimization of the system through a global learning scheme. The current system has been optimized by training the symbol and relation classifiers independently and then integrating them into a cost function. However, the literature suggests that for systems that are composed of several recognition modules, a training scheme that optimizes all modules over a single loss function provides better results (Lecun et al., 1998).

The proposed graph grammar and parsing algorithm are general enough to be adapted to other two-dimensional object recognition problems. An eventual adaptation would involve to retrain the symbol and relation classifiers and to write on a graph grammar according to the recognized language. As the parsing algorithm finds stroke partitions by searching for an isomorphism between a graph of a production rule and another derived by the hypotheses graph, the parsing algorithm would remain applicable.

9.4 Automatic annotation of mathematical expression datasets

The need for large ground truthed datasets of two-dimensional handwritten data has been emphasized in the literature (Lapointe, 2008; Lapointe and Blostein, 2009). In the case of mathematical expressions, although the CROHME dataset has helped to establish a common framework to evaluate different recognition methods, more data is still required. Let's consider that, for instance, for digit classification (only 10 symbol classes), the MNIST dataset has about $70,000$ samples; and, for string parsing, the Wall Street Journal portion of the Penn Treebank has about $40,000$ sentences.

In contrast, the CROHME dataset has about $10,000$ mathematical expressions with 101 symbol classes, that combined by the 6 spatial relation classes defined in the dataset, form $O(101^2 * 6)$ symbol-to-symbol relation types.

Future work will include the improvement of the technique for annotating ground-truth into mathematical expressions, as described in Chapter 8. This improvement will include taking off the constraint of writing symbols with consecutive strokes and the integration of recognition techniques developed in this thesis. The method will be eventually used to build a publicly available large dataset for further experimentations.

Chapter 10

Résumé étendu en français

Reconnaissance d'Expressions Mathématiques Manuscrites Guidée par le Contexte

Contexte de l'étude

Cette thèse s'attaque au problème de la reconnaissance d'expressions mathématiques manuscrites en proposant de nouvelles approches pour aborder ce type de langage.

Les méthodes traditionnelles permettant la saisie d'expressions mathématiques dans un document numérique sont relativement complexes et peu efficaces. Elles sont basées soit sur la maitrise d'un langage de mise en forme, tel que LATEX, ou bien alors sur une navigation dans des menus déroulants, tel que l'outil MathType associé à l'environnement de MS-Word. Dans le premier cas, l'utilisateur doit mémoriser un grand nombre de codes spécifiques qui permettent la mise en forme de l'expression. Un exemple de tel langage est donné ci-dessous. Dans le second cas, l'usage de la souris et du clavier permet de composer son expression au prix de beaucoup d'allers et retours dans tous les sous-menus.

$ x_{1,2} = \frac{-b\pm\sqrt{b^2 – 4ac}}{2a} $

$$x_{1,2} = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

Fig. 1 : Chaine Latex et son rendu sous forme image

La reconnaissance automatique de l'expression donnée directement sous forme manuscrite par l'utilisateur est une solution très intéressante pour faciliter l'intégration d'une telle expression dans un document numérique. On distinguera deux types de modalité pour enregistrer la forme manuscrite, soit une image provenant d'un scan de la formule déjà écrite (domaine hors-ligne), soit une trajectoire temporelle résultant d'une acquisition en temps-réel à l'aide d'une interface tactile (domaine en-ligne).

Les expressions mathématiques font partie des langages bidimensionnels au même titre que les formules chimiques, les partitions musicales ou encore les diagrammes. Il s'agit dans ces langages de positionner dans le plan des symboles de telle sorte que la nature du symbole et sa position sont porteurs de sens. Par exemple, l'expression 2^x n'a pas le même sens que l'expression $2x$. Les symboles sont les mêmes, mais les relations spatiales sont différentes. Il est légitime d'espérer que

l'approche que l'on pourra proposer pour les expressions mathématiques puisse être facilement transposable pour les autres langages bidimensionnels.

D'un point de vue scientifique, beaucoup de verrous sont à lever. Les difficultés se situent au niveau des tâches de segmentation, de classification, d'analyse de structures, d'apprentissage automatique. De plus, il ne s'agit pas de résoudre ces tâches indépendamment les unes des autres mais de concert. Ce problème général est considéré comme un excellent sujet d'intérêt pour la communauté de reconnaissance des formes qui a consacré de nombreux travaux pour l'attaquer de multiples façons, Zanibbi and Blostein (2012). En particulier, il est intéressant de mentionner l'existence d'une compétition internationale depuis 2011 qui fait le point régulièrement sur les avancées dans ce domaine (Mouchère et al., 2011, 2012, 2013, 2014).

Nous considérerons dans ce travail des données d'entrée en-ligne captées à l'aide d'un stylo électronique ou de toutes surfaces tactiles permettant d'enregistrer la trajectoire de l'instrument d'écriture, celui-ci pouvant se limiter à l'extrémité du doigt. Dans ce cas, les données en-ligne constituées des coordonnées (x, y) des points échantillonnées le long de la trajectoire forment des séquences temporelles regroupées en traits (stroke). Un trait étant délimité par un posé et un levé.

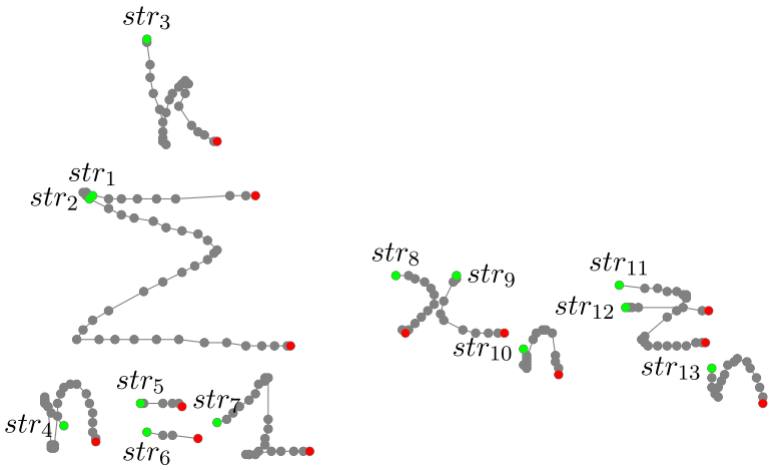


Fig. 2 : Exemple d'expression manuscrite extraite de la base CROHME-2014

L'exemple représenté sur la Figure 2 correspond à l'expression $\sum_{n=1}^{k} x_n z_n$. Elle est composée d'une séquence de 13 traits, noté *str* = *{str*$_1$ *, . . . , str*$_{13}$ *}*, où *str*$_i$ représente le i^{ème} trait en considérant l'ordre temporel de tracé des traits. Chaque trait est lui-même composé d'une séquence de points 2-D. Sur la figure, le premier point (posé) est affiché en vert, tandis que le dernier point (levé) est représenté en rouge. Comme on peut le voir sur cette figure, le pas d'échantillonnage spatial n'est pas constant, il est lié à la vitesse d'écriture. Il sera donc plus distant lorsque le tracé est rapide et plus serré lorsque celui-ci est lent. On cherche en général à s'affranchir de ces variations non significatives pour la reconnaissance en appliquant des prétraitements pour normaliser l'écriture.

Structure générale d'un système de reconnaissance d'expressions mathématiques

Trois tâches fondamentales sont à mettre en place. Elles consistent en 1) la segmentation du tracé, 2) la classification des symboles et 3) l'analyse structurelle.

1. La segmentation du tracé

Cette première tâche cherche à regrouper les traits élémentaires formant un même symbole. Il s'agit de réaliser une partition de l'ensemble des traits. Avec l'exemple de la Figure 2, la segmentation attendue devrait fournir la partition contenant 8 sous-ensembles, telle que P = {{str_1, str_2}, {str_3}, {str_4}, {str_5, str_6}, {str_7}, {str_8, str_9}, {str_{10}}, {str_{11}, str_{12}}, {str_{13}}}.

Le nombre de partitions possibles croit très rapidement avec le nombre de traits. Il est défini par le nombre de Bell qui satisfait la règle suivante :

$$B_{n+1} = \sum_{k=0}^{n} \binom{n}{k} B_k \quad \text{avec } B_0 = B_1 = 1.$$

En considérant l'exemple précédent où n = 13 traits, nous obtenons une valeur de B_{13} = 27 644 437 partitions distinctes. Dès lors, il n'est pas envisageable de toutes les considérer, il faudra introduire des contraintes spatiales et temporelles pour limiter cette explosion combinatoire. Certains travaux imposent la continuité temporelle dans la production des traits à l'intérieur des symboles. Dans ce cas, il est possible de représenter toutes les partitions de l'ensemble des traits par un graphe de segmentation sous la forme d'un treillis tel que celui représenté sur la figure 3. Le nombre de chemins dans ce treillis, soit le nombre de partitions, n'est plus alors que de 2^{n-1}, soit avec l'exemple précédent 4 096 partitions distinctes. Le chemin affiché en rouge sur la figure 3 représente la partition *P* = {{str_1, str_2}, {str_3, str_4}, {str_5}}. Dans nos travaux, nous avons relâché cette contrainte en acceptant des traits retardés. Cela donne à l'utilisateur la possibilité de corriger des symboles après la saisie des symboles suivants, par exemple changer l'expression « *x - 1* » en l'expression « *x + 1* » en rajoutant une barre verticale pour transformer le symbole '-' en '+'. La gestion des hypothèses de segmentation devient alors beaucoup plus complexe.

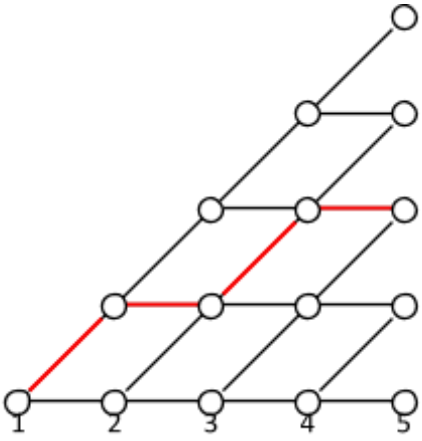


Fig. 3 : Treillis de segmentation avec contrainte temporelle.

2. La classification des symboles

La difficulté de cette tâche réside pour partie dans le nombre élevé de classes de symboles à considérer et à la confusion naturelle existant entre un certain nombre de ces classes. Ainsi, la compétition CROHME définit 101 classes de symboles mathématiques. La figure 4 présente quelques cas d'ambigüités dans la reconnaissance des formes. L'échantillon (a) peut être interprété comme le symbole '+' ou 't', pour la forme (b), on peut hésiter entre '6', 'G' ou 'σ' (sigma), la forme (c) peut être un '9' ou un 'q', tandis que la forme (d) peut représenter un 'p', un 'l', un 'ρ' (rho) ou un 'P'. Ces ambigüités devront être levées en utilisant autant que possible le contexte.

(a) (b) (c) (d)

Fig. 4 : Formes manuscrites ambigües. Symboles provenant de la base CROHME-2014.

Pour cette tâche, un classifieur de formes est utilisé. Dans une grande majorité des cas, on trouvera un système par apprentissage automatique de type réseau de neurones (NN : Neural Network), ou systèmes à vastes marges (SVM : Support Vector Machine). Un des points clés, outre le choix des caractéristiques et la dimension de l'espace de représentation, réside dans la stratégie d'apprentissage de ces machines. En effet, ce classifieur interagit avec les hypothèses de segmentation résultant de l'étape précédente. Il est donc important qu'il puisse réagir efficacement quelque soit la situation proposée, y compris vis-à-vis d'une segmentation invalide. Dans notre travail, nous avons privilégié un réseau de neurones de type perceptron multicouches (1 couche cachée), utilisant des caractéristiques d'histogrammes de contextes de formes (Shape context).

3. L'analyse structurelle

Cette étape d'analyse structurelle est spécifique au cas des langages graphiques tels que les expressions mathématiques et ne se pose pas en reconnaissance de l'écriture standard où les symboles sont simplement alignés de gauche à droite pour former du texte. Ici, il s'agit d'identifier les différentes relations spatiales existant entre les symboles ou les sous-expressions pour permettre de donner son sens à l'expression. Il est utile de guider cette analyse en s'appuyant sur une grammaire qui permet de contrôler ou de faciliter l'interprétation d'une expression. Par exemple, il est possible d'imposer la présence d'une parenthèse fermante si une parenthèse ouvrante a été préalablement reconnue. Dans le cadre de ces travaux, six relations spatiales sont considérées. En plus de la relation élémentaire *gauche-droite*, on distinguera : *exposant*, *indice*, *au-dessus*, *en-dessous*, et à *l'intérieur* (typiquement pour une racine).

Ainsi, même des expressions très courtes comme celle illustrée sur la figure 5, peuvent être difficiles à traiter à chacun des trois niveaux évoqués ci-dessus. Par exemple, il est possible de regrouper les traits str_4 et str_5 et alors reconnaitre un seul symbole, par exemple un '4'. A l'inverse, s'ils sont séparés, il s'agirait alors de la séquence « < 1 ». De même la relation spatiale entre les traits (str_1+str_2) et str_3 est difficilement décidable entre *gauche-droite* et *exposant*.

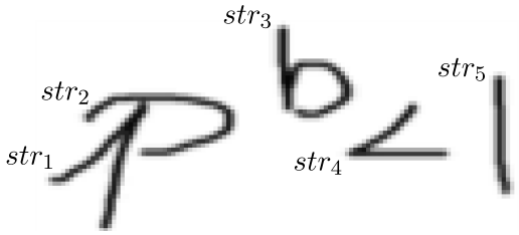


Fig. 5 : Une expression mathématique manuscrite difficile à interpréter.

Il en résulte toutes ces interprétations possibles : « P^b <1 », « p^b < 1 », « P^b4 » ou encore « p^b4 ».

Les points-clés de notre approche

Notre approche est fondée sur une méthode d'analyse de l'ensemble des traits contenus dans l'expression pour produire comme résultat un arbre d'analyse qui correspond à une interprétation particulière de l'expression. Pour gérer les ambigüités évoquées, la technique d'analyse va produire de multiples interprétations qui seront mises en concurrence. Nous appelons *T(str)* l'ensemble de tous les arbres d'analyse produits à partir de l'ensemble *str* des traits. De plus, nous définissons une fonction de coût *cost(t, str)* qui évalue la qualité de l'interprétation d'un arbre *t* élément de *T*. Le problème posé peut alors s'exprimer comme la recherche de l'arbre $t_{best}(str)$ tel que :

$$t_{best}(str) = \arg\min_{t \in T(str)} cost(t, str)$$

Le système global que l'on propose est résumé par la figure 6. Il est composé de deux fonctions principales : 1) un générateur de graphe des hypothèses et 2) un analyseur de graphe. Le générateur construit l'espace de recherche sous la forme d'un graphe dont l'exploration est contrôlée par l'outil d'analyse.

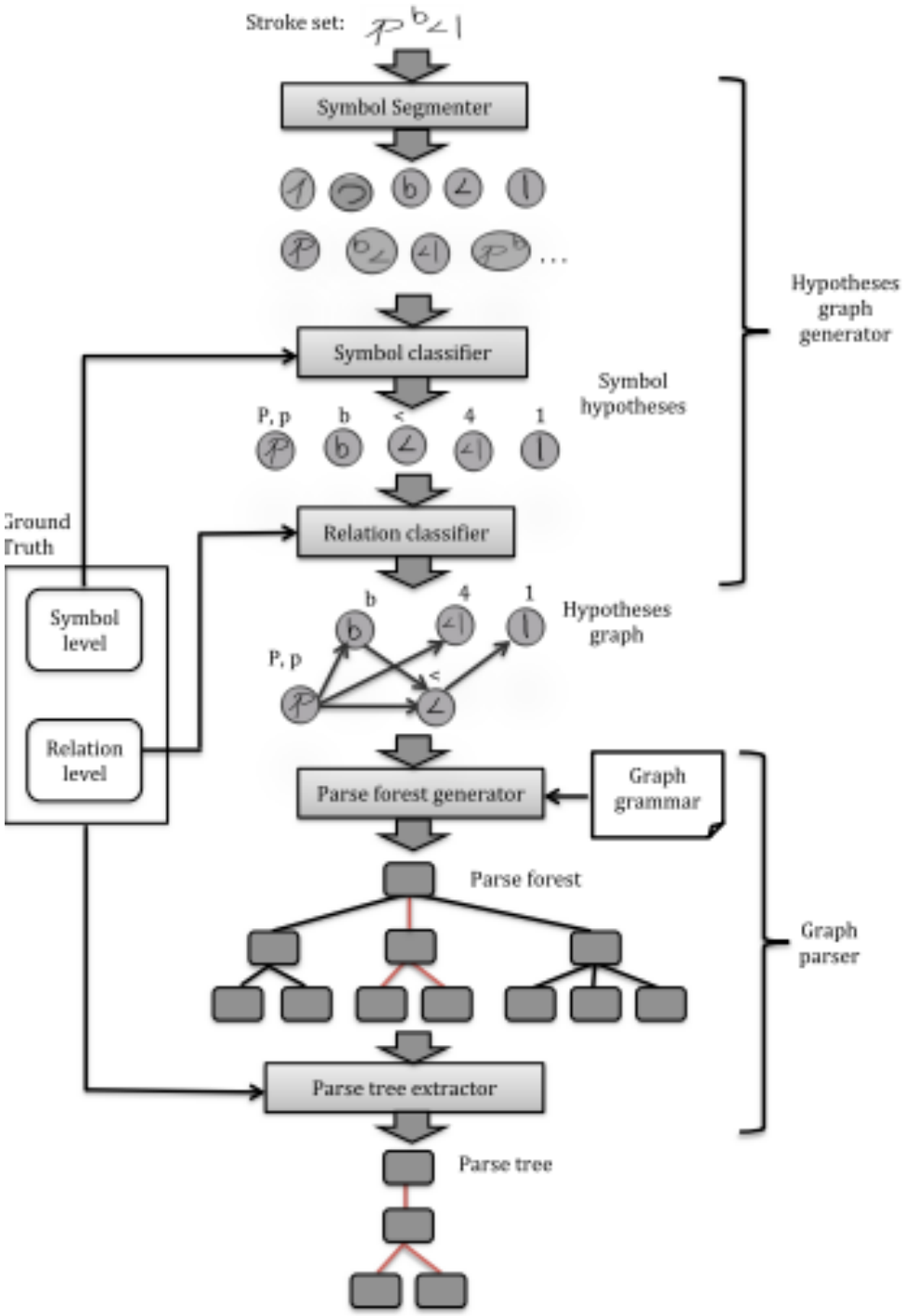


Fig. 6 : Schéma général du système de reconnaissance proposée.

En premier lieu, l'étape de segmentation propose les regroupements de traits qui peuvent représenter des symboles. Ces regroupements sont soumis à des contraintes de nombre de traits et de distance entre traits, ce qui interdit la fusion de traits trop éloignés. Ils sont ensuite soumis au classifieur de symbole (NN) qui produit pour chaque hypothèse de symbole, une distribution de probabilités sur l'ensemble des classes de symboles. A partir de ces groupements étiquetés au niveau symbole, un classifieur de relations identifie les relations existant entre une paire donnée de ces groupements. Ces résultats sont stockés dans le graphe des hypothèses dont les nœuds sont étiquetés par les labels des symboles et les arcs par les labels des relations.

Ainsi que le montre la figure 6, le graphe des hypothèses contient plusieurs alternatives concurrentes. L'outil d'analyse du graphe va générer plusieurs arbres correspondants à différentes interprétations de l'ensemble des traits d'entrée. Par ailleurs, une grammaire définit les règles de construction des graphes correspondants à des interprétations possibles des expressions. La construction des arbres est faite par recherche de structures dans le graphe qui correspondent aux motifs admissibles par la grammaire de graphes. Tous les arbres valides poussent dans une forêt permettant une gestion efficace de l'espace de stockage. Pour finir, la sélection du plus beau sujet de la forêt est réalisée. Celle-ci utilise la fonction de coût évoquée précédemment et une technique de recherche sous-optimale qui évite un examen exhaustif de toute le forêt en évitant l'évaluation d'arbres dont le coût est prohibitif.

Cette stratégie basée sur une grammaire de graphes fournit une approche générale pour modéliser des langages bidimensionnels. Les phrases (dans ce cas, les expressions mathématiques) de notre grammaire sont représentées par des graphes étiquetés dont les nœuds sont les terminaux (symboles) et les arcs les relations entre les symboles. Dans la mesure où les arcs représentent des relations quelconques et où les nœuds sont dans un ordre quelconque, une structure 2D quelconque peut être générée par ce formalisme.

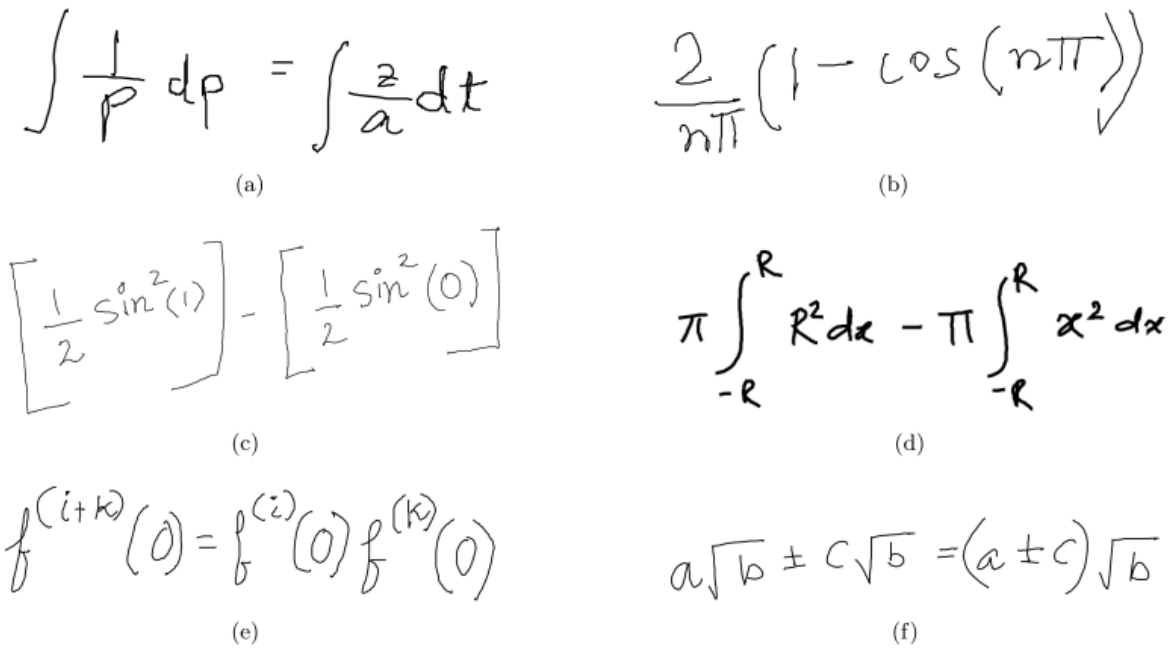
L'inconvénient d'une telle approche en reconnaissance des formes réside dans la complexité calculatoire (Flasiński and Jurek, 2014). Pour pallier cet inconvénient, nous limitons l'exploration de l'espace de recherche à la fois dans la construction du graphes des hypothèses et dans l'exploration de la forêt.

Résultats expérimentaux

Nous présentons ici à la fois des résultats quantitatifs et qualitatifs obtenus sur la base de test de CROHME-2014. Le tableau I, montre les taux de reconnaissance au niveau expression complète des 7 systèmes ayant participé à la compétition et les nôtres en dernière ligne. Nous sommes classés 3ème sans erreur et 2nd en acceptant 2 erreurs par expression. Les exemples présentés sur la figure 7 montrent un ensemble d'expressions complètement bien reconnues.

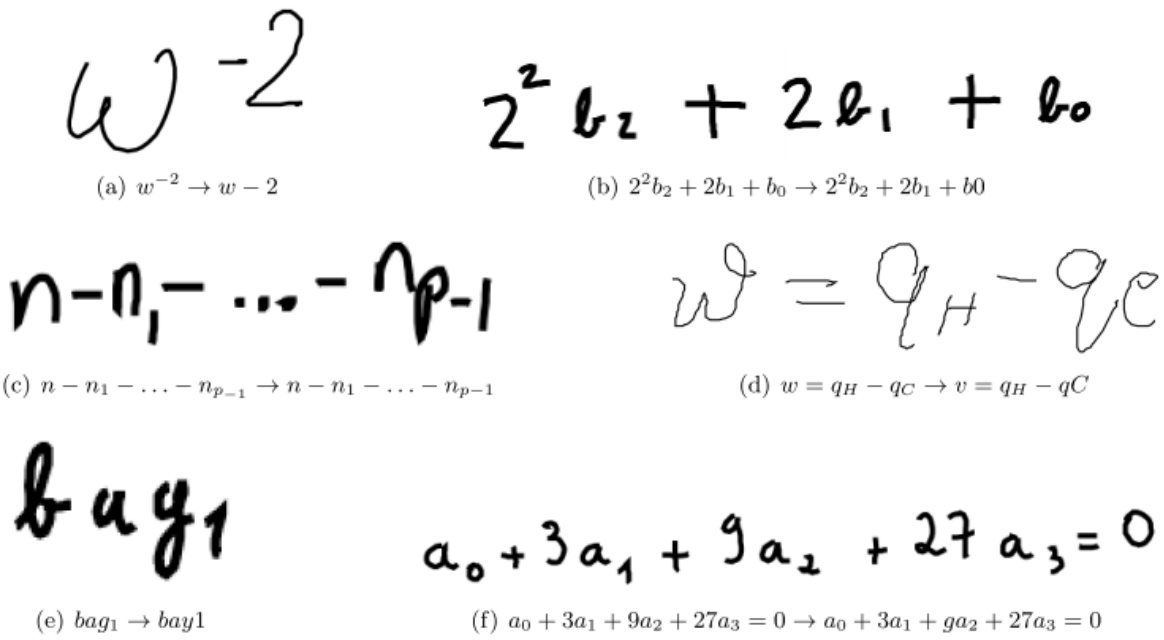
Tableau I. Pourcentages de reconnaissance au niveau expression complète

System	correct	#errors ≤ 1	#errors ≤ 2	#errors ≤ 3
I	37.22	44.22	47.26	50.20
II	15.01	22.31	26.57	27.69
III	62.68	72.31	75.15	76.88
IV	18.97	28.19	32.35	33.37
V	18.97	26.37	30.83	32.96
VI	25.66	33.16	35.90	37.32
VII	26.06	33.87	38.54	39.96
ours	**33.98**	**43.10**	**47.56**	**49.29**



(a) (b) (c) (d) (e) (f)

Fig. 7 : Exemples d'expressions bien reconnues.



(a) $w^{-2} \rightarrow w - 2$ (b) $2^2b_2 + 2b_1 + b_0 \rightarrow 2^2b_2 + 2b_1 + b0$

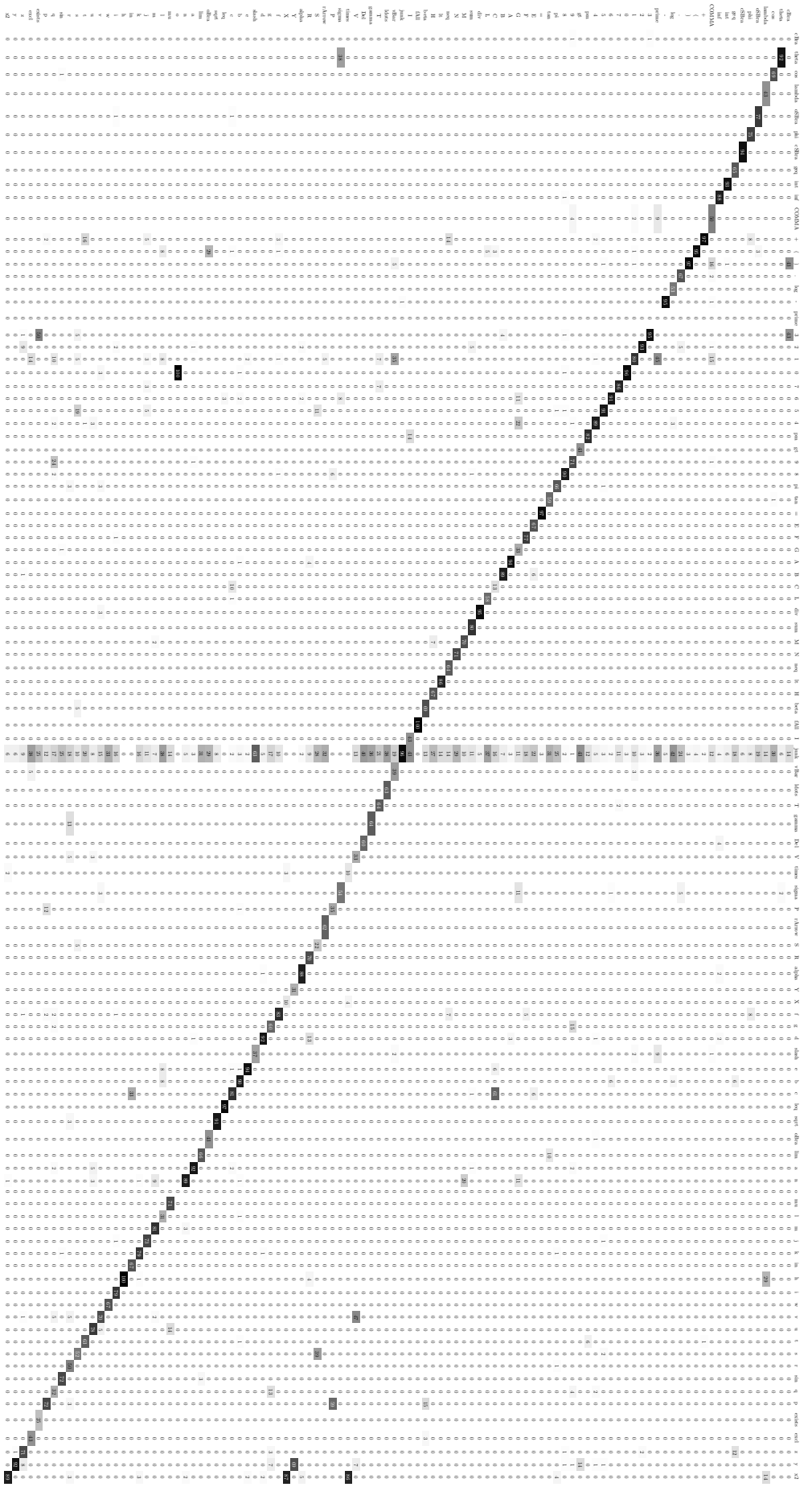
(c) $n - n_1 - \ldots - n_{p-1} \rightarrow n - n_1 - \ldots - n_{p-1}$ (d) $w = q_H - q_C \rightarrow v = q_H - qC$

(e) $bag_1 \rightarrow bay1$ (f) $a_0 + 3a_1 + 9a_2 + 27a_3 = 0 \rightarrow a_0 + 3a_1 + ga_2 + 27a_3 = 0$

Fig. 8 : Exemples d'expressions avec quelques erreurs de reconnaissance Vérité terrain -> Expression reconnue.

Appendix A

CROHME-2014 Symbol hypothesis classification



CROHME-2014 Symbol hypothesis classification confusion matrix

Appendix B

Hypothesis graph labels per symbol and relation

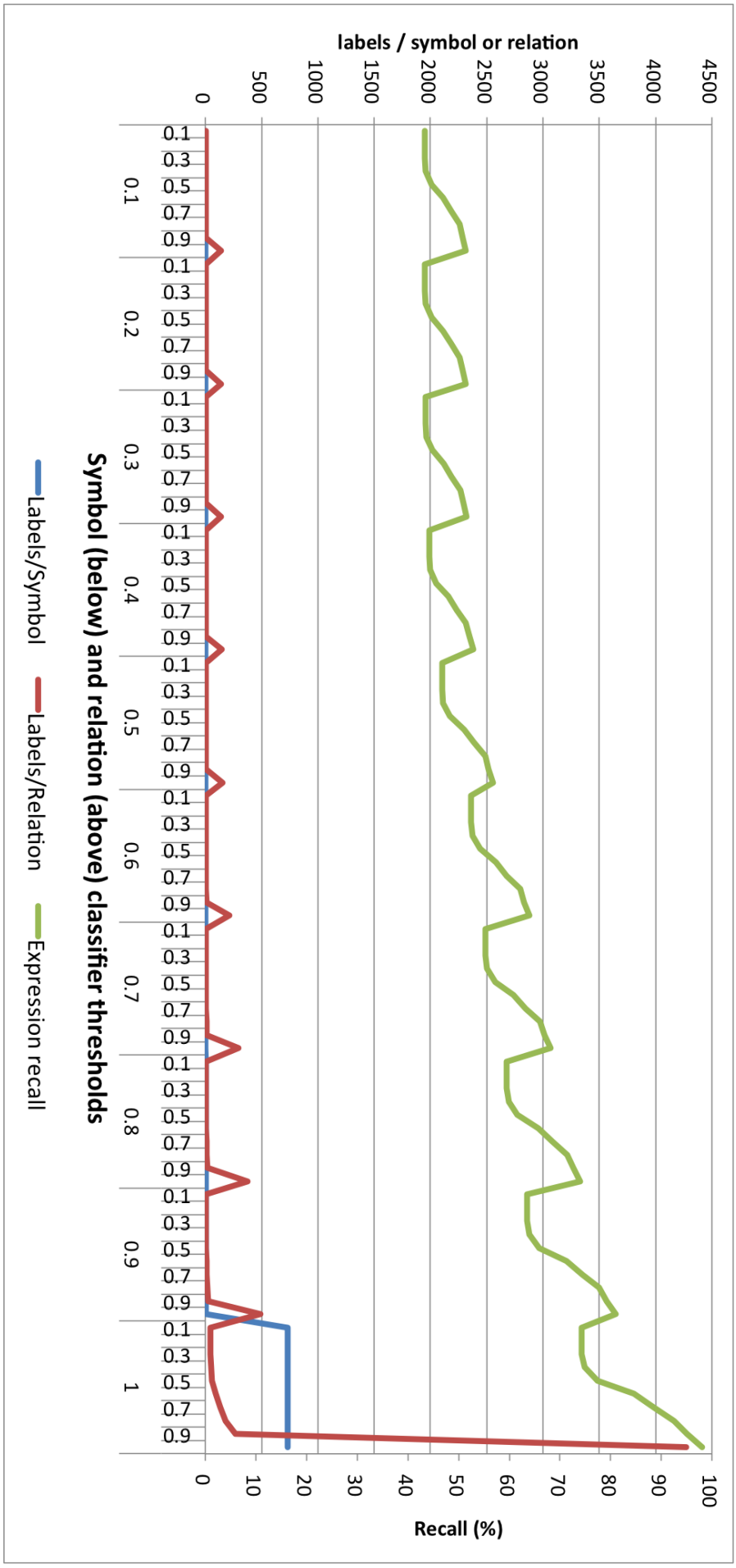


Figure B.1: *Mean of symbol and relation hypothesis labels per symbol and relation, respectively; calculates over the test set. The left hand axis indicates the number of labels per symbol and relation. The right hand axis indicates the percentage of recognized expressions.*

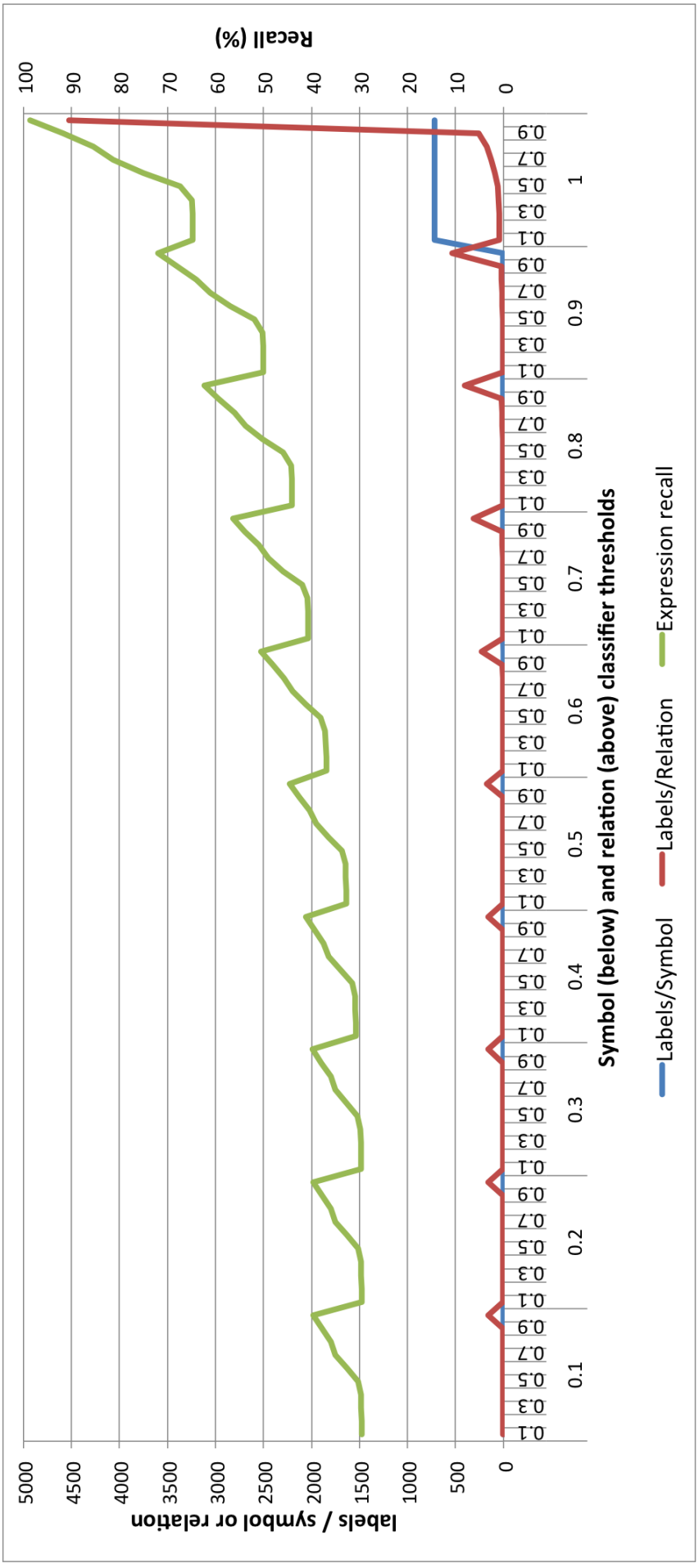


Figure B.2: *Mean of symbol and relation hypothesis labels per symbol and relation, respectively; calculates over the test set. The left hand axis indicates the number of labels per symbol and relation. The right hand axis indicates the percentage of recognized expressions.*

Bibliography

Frank D. J. Aguilar and Nina S. T. Hirata. Expressmatch: A system for creating ground-truthed datasets of online mathematical expressions. In *Proceedings of the 2012 10th IAPR International Workshop on Document Analysis Systems*, DAS '12, pages 155–159, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4661-2. doi: 10.1109/DAS.2012.38. 2, 9

J. Almazan, A. Fornes, and E. Valveny. A non-rigid feature extraction method for shape recognition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 987–991, Sept 2011. doi: 10.1109/ICDAR.2011.200. 46

F. Álvaro, J.-A. Sanchez, and J.-M. Benedi. Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1225–1229, Sept 2011. doi: 10.1109/ICDAR.2011.247. 18

F. Álvaro, J.-A. Sanchez, and J.-M. Benedi. Offline features for classifying handwritten math symbols with recurrent neural networks. In *Pattern Recognition (ICPR), 2014 22nd International Conference on*, pages 2944–2949, Aug 2014. doi: 10.1109/ICPR.2014.507. 40

Francisco Álvaro and Richard Zanibbi. A shape-based layout descriptor for classifying spatial relationships in handwritten math. In *Proceedings of the 2013 ACM Symposium on Document Engineering*, DocEng '13, pages 123–126, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1789-4. doi: 10.1145/2494266.2494315. URL http://doi.acm.org/10.1145/2494266.2494315. ix, 2, 34, 35, 51, 53, 54, 57

Francisco Álvaro, Joan-Andreu Sanchez, and José-Miguel Benedí. Recognition of on-line handwritten mathematical expressions using 2d stochastic context-free grammars and hidden markov models. *Pattern Recognition Letters*, 2012. 18, 21, 23, 27, 29, 37, 39, 79

Robert H. Anderson. *Syntax-directed recognition of hand-printed two-dimensional mathematics*. PhD thesis, Dept. Eng. Appl. Phys., Harvard University, 1968. 2, 11, 21, 23, 51

Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. Towards handwritten mathematical expression recognition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 1046–1050, 2009. 18, 23, 83

Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. The problem of handwritten mathematical expression recognition evaluation. In *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 646–651, 2010a. 2, 84

Ahmad-Montaser Awal, Harold Mouchère, and Christian Viard-Gaudin. Improving online handwritten mathematical expressions recognition with contextual modeling. In *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, pages 427–432, 2010b. 3, 18, 23

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(0):68 – 77, 2012. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2012.10.024. 18, 34, 39

S. Belongie, J. Malik, and J. Puzicha. Shape matching and object recognition using shape contexts. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:509–522, April 2002. 40, 41, 43, 44, 53

Y. Bengio, A. Courville, and P. Vincent. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828, 2013. ISSN 0162-8828. doi: http://doi.ieeecomputersociety.org/10.1109/TPAMI.2013.50. 94

D. Blostein and A. Grbavec. Recognition of mathematical notation. In H. Bunke and P. Wang, editors, *Handbook of Character Recognition and Document Image Analysis*, pages 557–582. World Scientific, 1997. 11

Pierre Boullier, Alexis Nasr, and Benoît Sagot. Constructing parse forests that include exactly the n-best pcfg trees. In *Proceedings of the 11th International Conference on Parsing Technologies*, IWPT '09, pages 117–128, Stroudsburg, PA, USA, 2009. Association for Computational Linguistics. URL http://dl.acm.org/citation.cfm?id=1697236.1697259. 67

Bruno Buchberger. Mathematica: A system for doing mathematics by computer? In Alfonso Miola, editor, *Design and Implementation of Symbolic Computation Systems*, volume 722 of *Lecture Notes in Computer Science*, pages 1–1. Springer Berlin Heidelberg, 1993. ISBN 978-3-540-57235-0. 16

H. Bunke. Attributed programmed graph grammars and their application to schematic diagram interpretation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, PAMI-4(6): 574–582, Nov 1982. ISSN 0162-8828. doi: 10.1109/TPAMI.1982.4767310. 29

M. Celik and B. Yanikoglu. Probabilistic mathematical formula recognition using a 2d context-free graph grammar. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 161–166, Sept 2011. doi: 10.1109/ICDAR.2011.41. viii, 21, 22, 23, 27, 29, 30, 37, 64, 95

Kam-Fai Chan and Dit-Yan Yeung. Mathematical expression recognition: a survey. *International Journal on Document Analysis and Recognition*, 3:3–15, 2000. 11

N. Chomsky. Three models for the description of language. *Information Theory, IRE Transactions on*, 2(3):113–124, September 1956. ISSN 0096-1000. doi: 10.1109/TIT.1956.1056813. 13

P. A. Chou. Recognition of equations using a two-dimensional stochastic context-free grammar. *Proc. SPIE 1199, Visual Communications and Image Processing IV, vol. 1199*, pages 852–865, 1989. doi: 10.1117/12.970095. 15, 23

Michael Collins. Head-driven statistical models for natural language parsing. *Comput. Linguist.*, 29(4):589–637, December 2003. ISSN 0891-2017. doi: 10.1162/089120103322753356. URL http://dx.doi.org/10.1162/089120103322753356. 80

Adrien Delaye and Eric Anquetil. Hbf49 feature set: A first unified baseline for online symbol recognition. *Pattern Recogn.*, 46(1):117–130, January 2013. ISSN 0031-3203. 40

Mariusz Flasiński and Janusz Jurek. Fundamental methodological issues of syntactic pattern recognition. *Pattern Analysis and Applications*, 17(3):465–480, 2014. ISSN 1433-7541. doi: 10.1007/s10044-013-0322-1. URL http://dx.doi.org/10.1007/s10044-013-0322-1. 8, 23, 29, 36, 95

Dick Grune and Jacobs Ceriel J.H. *Parsing Techniques: A Practical Guide*. Springer, 2ed edition, 2008. vii, 12, 13, 23, 30, 63

Feng Han and Song-Chun Zhu. Bottom-up/top-down image parsing with attribute grammar. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(1):59–73, Jan 2009. ISSN 0162-8828. doi: 10.1109/TPAMI.2008.65. 29, 64

Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2009. 43, 44

Nina S. T. Hirata and Willian Y. Honda. Automatic labeling of handwritten mathematical symbols via expression matching. In *Proceedings of the 8th International Conference on Graph-based Representations in Pattern Recognition*, pages 295–304, 2011. 5, 83, 84, 85, 89

Nina S.T. Hirata and Frank D. Julca-Aguilar. Matching based ground-truth annotation for online handwritten mathematical expressions. *Pattern Recognition*, 48(3):837 – 848, 2015. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2014.09.015. URL http://www.sciencedirect.com/science/article/pii/S0031320314003768. 2, 9, 85, 86

B.Q Huang and M-T Kechadi. A structural analysis approach for online handwritten mathematical expressions. In *International Journal of Computer Science and Network Security,*, 2007. 6, 8, 34, 39

Frank Julca-Aguilar, Christian Viard-Gaudin, Harold Mouchère, Sofiane Medjkoune, and Nina Hirata. Mathematical symbol hypothesis recognition with rejection option. In *14th International Conference on Frontiers in Handwriting Recognition*, 2014a. 9

Frank Julca-Aguilar, Christian Viard-Gaudin, Harold Mouchère, Sofiane Medjkoune, and Nina Hirata. Integration of shape context and neural networks for symbol recognition. In *Semaine du Document Numérique et de la Recherche d'Information 2014 (SDNRI)*, 2014b. 9, 44

Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina S. T. Hirata. *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 20th Iberoamerican Congress, CIARP 2015, Montevideo, Uruguay, November 9-12, 2015, Proceedings*, chapter Top-Down Online Handwritten Mathematical Expression Parsing with Graph Grammar, pages 444–451. Springer International Publishing, Cham, 2015. ISBN 978-3-319-25751-8. doi: 10.1007/978-3-319-25751-8_53. URL http://dx.doi.org/10.1007/978-3-319-25751-8_53. 9

M. Koschinski, H.-J. Winkler, and M. Lang. Segmentation and recognition of symbols within handwritten mathematical expressions. In *Proc. IEEE Int. Conf. on Acoustics, Speech, and Signal Process.*, volume 4, pages 2439–2442, 1995. vii, 16

Martin Lange and Hans Leiß. To cnf or not to cnf? an efficient yet presentable version of the cyk algorithm. *Informatica Didactica*, 8, 2009. 14

Adrien Lapointe. Issues in performance evaluation of mathematical notation recognition systems. Master's thesis, Queen's Univ., 2008. 2, 5, 71, 83, 84, 95

Adrien Lapointe and Dorothea Blostein. Issues in performance evaluation: A case study of math recognition. In *Proceedings of the 10th International Conference on Document Analysis and Recognition*, pages 1355–1359, 2009. 5, 71, 84, 95

S. Lavirotte and L. Pottier. Optical formula recognition. In *Document Analysis and Recognition, 1997., Proceedings of the Fourth International Conference on*, volume 1, pages 357–361 vol.1, Aug 1997. doi: 10.1109/ICDAR.1997.619871. 29, 30, 37, 64

Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998. ISSN 0018-9219. doi: 10.1109/5.726791. 94, 95

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, 1996. ICASSP-96. Conference Proceedings, 1996 IEEE*, volume 6, pages 3434–3437 vol. 6, May 1996. 6, 8, 34, 39

Liang Lin, Tianfu Wu, Jake Porway, and Zijian Xu. A stochastic graph grammar for compositional object representation and recognition. *Pattern Recognition*, 42(7):1297 – 1307, 2009. ISSN 0031-3203. doi: http://dx.doi.org/10.1016/j.patcog.2008.10.033. URL http://www.sciencedirect.com/science/article/pii/S0031320308004603. 29

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 (IJDAR)*, 16(2):139–163, 2013. ISSN 1433-2833. viii, 2, 8, 18, 19, 20, 21, 23, 29, 34, 36, 39, 61, 65, 67

Scott MacLean, George Labahn, Edward Lank, Mirette Marzouk, and David Tausky. Grammar-based techniques for creating ground-truthed sketch corpora. *Int. J. Doc. Anal. Recognit.*, 14: 65–74, 2011. 2, 5, 53, 83

Nicholas E Matsakis. Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge, 1999. 16, 34, 39, 83

G. Mori, S. Belongie, and J. Malik. Efficient shape matching using shape contexts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(11):1832–1837, 2005. ISSN 0162-8828. 44

H. Mouchère, C. Viard-Gaudin, D.H. Kim, J.H. Kim, and U. Garain. Icfhr 2012 competition on recognition of on-line mathematical expressions (crohme 2012). In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 811–816, 2012. 2

H. Mouchère, C. Viard-Gaudin, U. Garain, D. H. Kim, Kim J. H., and Zanibbi R. Icdar 2013 crohme: Third international competition on recognition of online handwritten mathematical expressions. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 1428–1432, Aug 2013. doi: 10.1109/ICDAR.2013.288. 2

H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain. Icfhr 2014 competition on recognition of on-line handwritten mathematical expressions (crohme 2014). In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 791–796, Sept 2014. doi: 10.1109/ICFHR.2014.138. vii, 2, 3, 4, 5, 9, 27, 39, 47, 71, 72, 79, 83, 93

Harold Mouchère, Christian Viard-Gaudin, Dae Hwan Kim, Jin Hyung Kim, and Utpal Garain. Crohme2011: Competition on recognition of online handwritten mathematical expressions. In *ICDAR'11*, pages 1497–1500, 2011. 2

J. Pflatz and Rosenfeld A. Web grammars. In *Proc. First International Joint Conference on Artificial Intelligence*, pages 193–220, 1969. 23, 25, 26

Réjean Plamondon and Sargur N. Srihari. On-line and off-line handwriting recognition: A comprehensive survey. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22:63–84, January 2000. ISSN 0162-8828. 11

L. Prasanth, V. Babu, R. Sharma, G. V. Rao, and Dinesh M. Elastic Matching of Online Handwritten Tamil and Telugu Scripts Using Local Features. In *Proceedings of the Ninth International Conference on Document Analysis and Recognition - Volume 02*, pages 1028–1032, 2007. 41, 44

S. Quiniou, H. Mouchère, S.P. Saldarriaga, C. Viard-Gaudin, E. Morin, S. Petitrenaud, and S. Medjkoune. Hamex - a handwritten and audio dataset of mathematical expressions. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 452–456, Sept 2011. doi: 10.1109/ICDAR.2011.97. 2, 5

Taik Heon Rhee and Jin Hyung Kim. Efficient search strategy in structural analysis for handwritten mathematical expression recognition. *Pattern Recognition*, 42:3192–3201, 2009. 83

G. Sanchez and J. Llados. A graph grammar to recognize textured symbols. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 465–469, 2001. doi: 10.1109/ICDAR.2001.953833. 29

F. Simistira, V. Papavassiliou, V. Katsouros, and G. Carayannis. Recognition of spatial relations in mathematical formulas. In *Frontiers in Handwriting Recognition (ICFHR), 2014 14th International Conference on*, pages 164–168, Sept 2014. doi: 10.1109/ICFHR.2014.35. 51, 53

Fotini Simistira, Vassilis Katsouros, and George Carayannis. Recognition of online handwritten mathematical formulas using probabilistic {SVMs} and stochastic context free grammars. *Pattern Recognition Letters*, 53(0):85 – 92, 2015. ISSN 0167-8655. doi: http://dx.doi.org/10.1016/j.patrec.2014.11.015. URL http://www.sciencedirect.com/science/article/pii/S0167865514003651. 2, 23, 27, 29, 37

Ernesto Tapia. *Understanding Mathematics: A System for the Recognition of On-Line Handwritten Mathematical Expression.* PhD thesis, Freie Universität Berlin, 2004. 17, 34, 39, 40, 51

Ernesto Tapia and Raul Rojas. Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In *Graphics Recognition. Recent Advances and Perspectives*, volume 3088, pages 329–340. 2004. 6, 8, 17

Ernesto Tapia and Raúl Rojas. A survey on recognition of on-line handwritten mathematical notation, 2007. 11

Gy Turan. On the complexity of graph grammars. *Rep. Automata theory Research Group*, 1982. 36, 64

Stephen H. Unger. A global parser for context-free phrase structure grammars. *Commun. ACM*, 11(4):240–247, April 1968. ISSN 0001-0782. doi: 10.1145/362991.363001. 19, 23, 35

E. Valveny, P. Dosch, Adam Winstanley, Yu Zhou, Su Yang, Luo Yan, Liu Wenyin, Dave Elliman, Mathieu Delalandre, Eric Trupin, Sébastien Adam, and Jean-Marc Ogier. A general framework for the evaluation of symbol recognition methods. *Int. J. Doc. Anal. Recognit.*, 9:59–74, February 2007. 83

Ba-Quy Vuong, Yulan He, and Siu Cheung Hui. Towards a web-based progressive handwriting recognition environment for mathematical problem solving. *Expert Systems with Applications*, 37 (1):886–893, January 2010. 83

K. Wagner. Über eine eigenschaft der ebenen komplexe. *Mathematische Annalen*, 114(1):570–590, 1937. ISSN 0025-5831. URL http://dx.doi.org/10.1007/BF01594196. 66

Zhiyong Wang, Bin Lu, Zheru Chi, and Dagan Feng. Leaf image classification with shape context and sift descriptors. In *Digital Image Computing Techniques and Applications (DICTA), 2011 International Conference on*, pages 650–654, 2011. doi: 10.1109/DICTA.2011.115. 41

Liu Wenyin and D. Dori. Performance evaluation of graphics recognition algorithms: principles and applications. In *Pattern Recognition, 1998. Proceedings. Fourteenth International Conference on*, volume 2, pages 1180–1182 vol.2, Aug 1998. doi: 10.1109/ICPR.1998.711907. 83

H.-J. Winkler, H. Fahrner, and M. Lang. A soft-decision approach for structural analysis of handwritten mathematical expressions. In *Acoustics, Speech, and Signal Processing, 1995. ICASSP-95., 1995 International Conference on*, volume 4, pages 2459–2462 vol.4, 1995. 16

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*, 2006. vii, 2, 13, 14, 15, 18, 23, 29, 36, 37, 39, 65

Daniel H. Younger. Recognition and parsing of context-free languages in time n3. *Information and Control*, 10(2):189 – 208, 1967. ISSN 0019-9958. doi: http://dx.doi.org/10.1016/S0019-9958(67)80007-X. URL http://www.sciencedirect.com/science/article/pii/S001999586780007X. 13

Richard Zanibbi and Dorothea Blostein. Recognition and retrieval of mathematical expressions. *International Journal on Document Analysis and Recognition (IJDAR)*, 2012. ISSN 1433-2833. 2, 11

Richard Zanibbi, Dorothea Blostein, and James R. Cordy. Recognizing mathematical expressions using tree transformation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24:1455–1467, 2002. vii, ix, 8, 17, 18, 27, 34, 36, 51, 52

Richard Zanibbi, Harold Mouchère, and Christian Viard-Gaudin. Evaluating structural pattern recognition for handwritten math via primitive label graphs. In *Proceeding of Document Recognition and Retrieval XX, DRR 2013, USA*, 2013. 2

Ling Zhang, Dorothea Blostein, and Richard Zanibbi. Using fuzzy logic to analyze superscript and subscript relations in handwritten mathematical expressions. In *Proceedings of the Eighth International Conference on Document Analysis and Recognition*, pages 972–976, 2005. ix, 8, 17, 52, 67

Publications

1. Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina S. T. Hirata. Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 20th Iberoamerican Congress, CIARP 2015, Montevideo, Uruguay, November 9-12, 2015, Proceedings, chapter Top-Down Online Handwritten Mathematical Expression Parsing with Graph Grammar, pages 444-451.

2. Nina S.T. Hirata and Frank D. Julca-Aguilar. Matching based ground-truth annotation for online handwritten mathematical expressions. Pattern Recognition, 48(3):837 - 848, 2015.

3. Frank Julca-Aguilar, Christian Viard-Gaudin, Harold Mouchère, Sofiane Medjkoune, and Nina Hirata. Mathematical symbol hypothesis recognition with rejection option. In 14th International Conference on Frontiers in Handwriting Recognition.

4. Frank Julca-Aguilar, Christian Viard-Gaudin, Harold Mouchère, Sofiane Medjkoune, and Nina Hirata. Integration of shape context and neural networks for symbol recognition. In Semaine du Document Numérique et de la Recherche d'Information (SDNRI) 2014 .

5. Alexandre Noma, Frank D. Julca-Aguilar, Nina S. T. Hirata. Matching Expressions by using Structural Belief Propagation: First Results. 26th Conference on Graphics, Patterns, and Images, 2013 (SIBGRAPI 2013).

6. Frank D. J. Aguilar and Nina S. T. Hirata. Expressmatch: A system for creating ground-truthed datasets of online mathematical expressions. In Proceedings of the 2012 10th IAPR International Workshop on Document Analysis Systems, DAS'12, pages 155-159, Washington, DC, USA, 2012. IEEE Computer Society.

UNIVERSITE BRETAGNE LOIRE

IME-USP

UNIVERSITÉ DE NANTES

Thèse de Doctorat

Frank D. JULCA-AGUILAR

Recognition of Online Handwritten Mathematical Expressions using Contextual Information

Résumé

Les expressions mathématiques manuscrites en-ligne sont constituées d'une séquence de traces. Leur reconnaissance nécessite de résoudre trois problèmes fondamentaux: la segmentation de ses symboles, leur reconnaissance et l'analyse structurelle de l'expression (i.e. l'identification des relations spatiales inter-symboles). La nature ambiguë des expressions manuscrites et leur structure spatiale sont les problèmes principaux du processus de reconnaissance. Dans cette thèse, nous proposons une modélisation du problème par une analyse syntaxique de graphes. La description des règles de production grammaticales par des graphes permet de modéliser directement la nature non linéaire des structures. Notre algorithme d'analyse détermine récursivement les partitions des traces en respectant les graphes des règles de production. Pour diminuer le coût de calcul, graphes issus des partitions sont limitées à un ensemble d'hypothèses de symboles et de relations pré-calculées grâce à classifieur entraîné. Ce classifieur donne à chaque hypothèse un ensemble d'étiquettes associées à leur vraisemblance. C'est l'analyse syntaxique qui sélectionnera la meilleure interprétation globale. Cette analyse produit de plusieurs arbres syntaxiques pour représenter plusieurs interprétations, leur associe un coût et choisit l'arbre avec le coût le plus faible pour l'interprétation finale. Les évaluations effectuées sur une base d'expressions conséquente et publique ont permis de montrer que notre approche est plus performantes que plusieurs méthodes de l'état de l'art et que l'utilisation du graphe d'hypothèses de symboles et de relations permet de contrôler la complexité de l'analyse. L'adaptation à d'autres langages graphiques est possible.

Mots clés
Reconnaissance d'expressions mathématiques, reconnaissance de symboles, reconnaissance de relations spatiales, analyse syntaxique de graphes.

Abstract

Online handwritten mathematical expressions consist of sequences of strokes. Automatic recognition these data requires solving three subproblems: symbol segmentation, symbol classification, and structural analysis (i.e. identification of spatial relations between symbols). Ambiguity, that often leads to several likely interpretations, and the non-linear structure of the expressions are main issues of the recognition process. In this thesis, we model the recognition problem as a graph parsing problem. The graph-based description of relations in production rules allows direct modeling of non-linear structures. Our parsing algorithm determines recursive partitions of the input strokes that induce graphs matching the production rule graphs. To mitigate the computational cost, we constrain the partitions to graphs derived from sets of symbol and relation hypotheses, calculated using previously trained classifiers. A set of labels that indicate likely interpretations is associated to each hypothesis, and the selection of the best interpretation is driven by the parsing algorithm. The parsing method computes multiple parse trees to represent alternative interpretations, assigns a cost to each tree and selects a tree with minimum cost as result. The evaluations show that the proposed method is more accurate than several state of the art methods; the use of symbol and relation hypotheses to constrain the search space effectively reduces the parsing complexity; and adaptation to other two-dimensional object recognition problems is possible. As a secondary contribution, we developed a framework to automatize the handwritten mathematical expression datasets building process.

Key Words
Mathematical expression recognition, symbol classification, spatial relation classification, graph parsing.