



**HAL**  
open science

## A general framework for the recognition of online handwritten graphics

Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, Nina S. T.  
Hirata

► **To cite this version:**

Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, Nina S. T. Hirata. A general framework for the recognition of online handwritten graphics. *International Journal on Document Analysis and Recognition*, 2020, 10.1007/s10032-019-00349-6 . hal-02474242

**HAL Id: hal-02474242**

**<https://hal.science/hal-02474242>**

Submitted on 2 Jan 2023

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# A General Framework for the Recognition of Online Handwritten Graphics

Frank Julca-Aguilar, Harold Mouchère, Christian Viard-Gaudin, and Nina S. T. Hirata

**Abstract**—We propose a new framework for the recognition of online handwritten graphics. Three main features of the framework are its ability to treat symbol and structural level information in an integrated way, its flexibility with respect to different families of graphics, and means to control the tradeoff between recognition effectiveness and computational cost. We model a graphic as a labeled graph generated from a graph grammar. Non-terminal vertices represent subcomponents, terminal vertices represent symbols, and edges represent relations between subcomponents or symbols. We then model the recognition problem as a graph parsing problem: given an input stroke set, we search for a parse tree that represents the best interpretation of the input. Our graph parsing algorithm generates multiple interpretations (consistent with the grammar) and then we extract an optimal interpretation according to a cost function that takes into consideration the likelihood scores of symbols and structures. The parsing algorithm consists in recursively partitioning the stroke set according to structures defined in the grammar and it does not impose constraints present in some previous works (e.g. stroke ordering). By avoiding such constraints and thanks to the powerful representativeness of graphs, our approach can be adapted to the recognition of different graphic notations. We show applications to the recognition of mathematical expressions and flowcharts. Experimentation shows that our method obtains state-of-the-art accuracy in both applications.

**Index Terms**—Graphics recognition, online handwriting recognition, graph parsing, mathematical expression, flowchart.

## 1 INTRODUCTION

RECOGNITION of online handwriting aims at finding the best interpretation of a sequence of input strokes [1]. Roughly speaking, handwriting data can be divided into two broad categories: text and graphics. In text notation, symbols are usually composed of strokes that are consecutive relative to a time or spatial order; and symbols themselves are also arranged according to a specific order, for example, from left to right. The ordering of symbols defines a single adjacency, or relation type, between consecutive symbols. By contrast, graphics encompass a variety of object types such as mathematical or chemical expressions, diagrams, and tables. Symbols in graphics notation are often composed of strokes that are not consecutive with respect to neither time nor spatial order. Furthermore, a diversified set of relations is possible between arbitrary pairs of symbols. See Figure 1, for instance, where a handwritten mathematical expression illustrates some characteristics of graphics notation.

Due to the linear arrangement of symbols, text recognition can be modeled as a parsing of one-dimensional (1D) data. On the other hand, graphics are intrinsically two-dimensional (2D) data, requiring a structural analysis, and there are no standard parsing methods as in the 1D case. Parsing depends on symbol segmentation (or, stroke grouping), symbol identification, and analysis of structural relationship among constituent elements. Stroke grouping

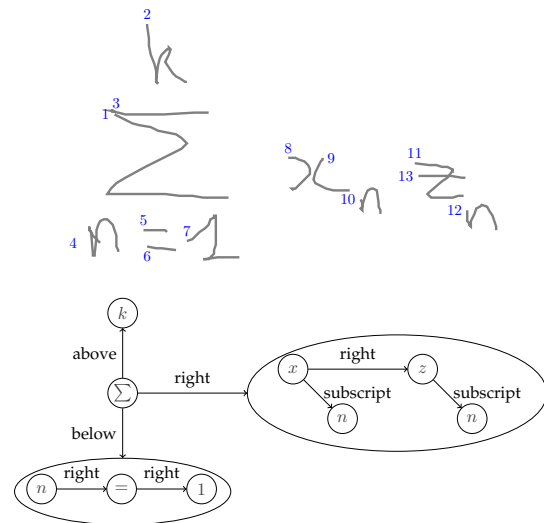


Fig. 1. Handwritten mathematical expression example. Top: A sequence of strokes where the order (indicated by numbers in blue) is given by the input time. Symbols  $\Sigma$  and  $z$  are composed of non-consecutive strokes. Bottom: The expression  $\Sigma$  is composed of symbols and several types of spatial relations between them.

in texts is relatively simpler than in graphics as already mentioned. Identification of segmented symbols include challenges such as the possibly large number of symbol classes, shape similarity between symbols in distinct classes, and shape variability within a same class (e.g. arrows in flowcharts might include arbitrary curves, and be directed towards any orientation). Structural analysis involves the identification of relations between symbols and a coherent integrated interpretation. The large variety of relations

- Frank Julca-Aguilar and Nina S. T. Hirata are with the Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, Brazil.  
E-mail: {faguilar, nina}@ime.usp.br
- Harold Mouchère and Christian Viard-Gaudin are with Institut de Recherche en Communications et Cybernétique of Nantes, University of Nantes.  
E-mail: {christian.viard-gaudin, harold.mouchere}@univ-nantes.fr

might define complex hierarchical structures that increments the difficulty in terms of efficiency and accuracy. There is a strong dependency among the three tasks since symbol segmentation and classification algorithms must often rely on structural or contextual information to solve ambiguities, and structural analysis algorithms depend on symbol identification to build coherent structures.

Although recognition of 2D objects is a subject of study since long ago [2], many of the efforts are still focused on solving specific aspects of the recognition process (e.g., detection of constituent parts or classification of components and their relations). A large number of works that tackle the entire recognition problem is clearly emerging, but they are often restricted to specific application domains and have limitations [3], [4], [5].

Motivated by the problem of online handwritten mathematical expression recognition, we have examined issues related to the recognition process and identified three features that are desirable. The first feature is *multilevel information integration*. By multilevel information integration we mean integrating symbol and structural level information to find the best interpretation of a set of strokes. In mathematical expression recognition, methods that seek information integration have already been the concern of several works [6], [7], [8], but it is still one of the most challenging problems. The second feature is related to *model generalization*. Existing methods often limit the type of expressions to be recognized (for instance, do not include matrices), consider a fixed notation (for instance, it adopts either  $\sum_{i=1}^n x_i$  or  $\sum_{i=1}^n x_i$ ), or limit the set of mathematical symbols to be recognized. Any extensions regarding these limitations may require major changes in the recognition algorithms. The third feature is *computational complexity management*. A general model often results in exponential time algorithms, making its application unfeasible. Existing models handle time complexity issues by adopting constraints that limit the recognizable structures [8], [9].

To deal with the issues described above, we have elaborated a general framework for the recognition of online handwritten mathematical expressions and then show its generality by building a flowchart recognition system using the same framework. We model a mathematical expression as a graph, and represent the recognition problem as a graph parsing problem. The recognition process is divided into three stages: (1) hypotheses identification, (2) graph parsing, and (3) optimal interpretation retrieval. The first stage computes a graph, called hypotheses graph, that encodes plausible symbol interpretations and relations between pairs of such symbols. The second stage parses the set of strokes to find all interpretations that are valid according to a pre-defined graph grammar, using the hypotheses graph to constrain the search space. The parsing method is based on a recursive search of isomorphisms between a labeled graph defined in the graph grammar and the ones derived from the hypotheses graph. The last stage retrieves the most likely interpretation based on a cost function that models symbol segmentation, classification and structural information jointly.

Conceptually, the valid structures are defined through a

graph grammar and likely structures in the input stroke set are captured in the hypotheses graph. Thus, the proposed framework enhances independence of the parsing step with respect to specificities of the mathematical notation considered. As a consequence, we have a flexible framework with respect to different mathematical notations. For instance, new expression structures can be included in the family of expressions to be recognized by just including the structures in the grammar rules. Similarly, the class of mathematical symbols to be recognized can be extended by just including new symbol labels in the grammar and in the hypotheses graph building procedure.

With respect to graphics in general, among them there is large difference in the set of symbols and relations between symbols. Thus, recognition techniques are often developed for a specific family of graphics, introducing constraints that not only limit their effectiveness, but also their adaptation to recognize different families of graphics. In spite of these differences, graphic notations share common concepts – a set of interrelated symbols spread over a bidimensional space, organized in hierarchical structures that are decisive to the interpretation. We argue that the flexibility of the proposed framework encompasses other families of graphics. This argument is supported by the fact that graphs has already proven adequate to model graphics in general. In addition, there are examples that show that families of graphics can be specified by means of a graph grammar [5], [10], [11]. Moreover, hypotheses graphs can be built based on data-driven approaches.

The main contributions of this work are thus twofold. First, we present a general framework in which the parsing process is independent of the family of graphics to be recognized and a control of the computational time is possible by means of a hypotheses graph. Second, we demonstrate an effective application of the framework to the recognition of mathematical expressions and flowcharts.

The remaining of this text is organized as follows. In Section 2 we review some methods and concerns in previous works related to the recognition of mathematical expression and flowcharts, as these types of graphics served as the ground for the development of the method described in this manuscript. We also briefly comment on some works that proposed graph grammars for the recognition of 2D data and influenced our work. In Section 3 we detail the proposed framework. Then in Section 4 we describe how the elements and parameters required by the framework have been defined for the recognition of mathematical expressions and flowcharts. In Section 5 we present and discuss the experimental results for both applications, and in Section 6 the conclusions and future works.

## 2 RELATED WORK

In this section, we review some characteristics of the recognition process in previous works, with emphasis on methods for mathematical expression [12], [13], [14] and flowchart recognition [15], [16], [17].

Early works related to the recognition of mathematical expressions were predominantly based on a sequential recognition process consisting of the symbol segmentation, symbol identification and structural analysis steps [18], [19],

[20]. However, a weakness of sequential methods is the fact that errors in early steps are propagated to subsequent steps. For instance, it might be difficult to determine if two handwritten strokes with shape “)” and “(”, close to each other, form a single symbol “x”, or are the opening and the closing parentheses, respectively. To solve this type of ambiguity, it may be necessary to examine relations of the strokes with other nearby symbols or even with respect to the global structure of the whole expression. This type of observation has motivated more recent works to consider methods that integrate symbol and structural level interpretations into a single process. Most of them are based on parsing methods as described below.

Given an input stroke set, the goal of parsing is to find a parse tree that “explains” the structure of the stroke set, relative to a predefined grammar. From a high-level perspective, parsing-based techniques avoid sequential processing by generating several symbol and relation interpretations, combining them to form multiple interpretations of the whole input stroke set, and selecting the best one according to a score (based on the whole structure).

An important element in parsing based approaches is the grammar. A grammar defines how we model a (graphics) *language*. For mathematical expressions, most approaches [21], [22], [23], [24], [25] use modifications of context-free string grammars in Chomsky Normal Form<sup>1</sup> (CNF). Such grammars define production rules of the form  $A \xrightarrow{r} BC$ , where  $r$  indicates a relation between adjacent elements of the right hand side (RHS) of the rule. For instance, expression  $4^2$  can be modeled through a rule  $TERM \xrightarrow{\text{superscript}} NUMBER NUMBER$ . However, as such grammars impose the restriction of having at most two elements on the RHS of a rule, structures with more than two components, like  $2 + 4$ , or  $\sum_i^n x_i$ , must be modeled as a recursive composition of pairs of components. MacLean *et. al.* [8] proposed *fuzzy relational context free grammars* to overcome this limitation. They included 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. However, the model assumes that the relation can only be of vertical or horizontal types. *Celik and Yanikoglu* [9] use graph grammars with production rules of the form  $A \rightarrow B$ , where both  $A$  and  $B$  are graphs, and  $B$  represents the components of a subexpression as vertices and their relations as edges. Graph grammar models offer more powerful representativeness compared to string grammars. However, the authors limit the grammars to have specific structures (each graph in a rule is either a single vertex graph, or a *star* graph – a graph with a single central vertex and surrounding vertices that are connected only to the central one), largely restricting the set of recognizable expressions.

With respect to parsing, most algorithms proposed in the literature for mathematical expressions are based on the CYK algorithm [26]. The CYK algorithm assumes that the input (in our case) strokes form a sequence and the grammar is in CNF. Those based on bottom-up approaches build a parse tree by first identifying symbols (leaves) from

single or groups of consecutive strokes, and then combining the symbols recursively to form subcomponents (subtrees), until obtaining a component that covers the whole input set. To adapt the CYK algorithm to the recognition of mathematical expressions, *Yamamoto et. al.* [24] introduced an ordering of the strokes based on the input time. Other approaches avoid the stroke ordering assumption, but introduce different constraints to satisfy the decomposition of the input into pairs of components [21], [22], [23], [25]. MacLean *et. al.* [8] proposed a top-down parsing algorithm that does not assume grammars in CNF, but assumes that the input follows either a vertical or horizontal ordering (the *fuzzy relational context free grammars* mentioned above). Methods that use the CYK algorithm or others borrowed from the context of string grammars must decompose the 2D input into a set of 1D inputs. As there is no guarantee that such decomposition is possible, these methods may present strong limitations with respect to parsable 2D structures and be completely inappropriate for other types of 2D data.

On the other hand, methods that consider graph grammars face computational complexity issues. A key step of any parsing algorithm is the definition of how a stroke set can be partitioned according to the RHS of a rule. Let us consider a set of  $n$  strokes. Assuming stroke ordering and a CYK-based algorithm as in [21], [22], [23], [24], [25], rules have at most two components in the RHS and therefore the number of meaningful partitions is  $O(n)$  – we can assign the first  $i$  strokes to the first component and the rest for the second, with  $i \in \{1, \dots, n - 1\}$ . On the other hand, if we do not impose CNF, but keep the stroke ordering assumption as in [8], then a rule may have  $k$  symbols on its RHS, and the number of meaningful partitions is  $O\binom{n}{k}$ , corresponding to  $k - 1$  split points on the sequence of  $n$  strokes. In graph grammars, without any restriction and a rule with  $k$  vertices in the RHS, the number of partitions is  $O(n^k)$  – any non-empty stroke subset can be mapped to any vertex. Restricting the graph structures in the grammar, for instance to star graph structures as done by *Celik and Yanikoglu* [9], is a way to manage the parsing complexity. Note, however, that in this case the set of recognizable expressions is constrained not only by the parsing algorithm but also by the grammar.

Flowcharts in general have a smaller symbol set than mathematical expressions. However, their structure presents higher variance. For instance, the flowchart in Figure 2 includes two loops, and adjacent symbols can be located at any (vertical, horizontal, or diagonal) position relative to each other, regardless the relation type. In contrast, in mathematical expressions, for a given relation type between two symbols (e.g. superscript) it is expected that one symbol is located at some specific area relative to the other (e.g. top-right). Thus, for flowcharts it may be difficult to establish a spatial ordering of the input strokes.

To cope with the structural variance of diagrams, some approaches introduce strong constraints in the input, as requiring all symbols to have only one stroke [27], or loop-like symbols to be written by consecutive strokes [15]. With respect to symbol recognition, detection of texts (or text box) and arrow symbols are regarded as more difficult, as they do not present a fixed shape. For instance, *Carton et. al.* [16] determine box symbols (like *decision*, and *data* structure)

1. In a CNF, all production 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

and then select the best interpretations using a deformation metric. Text symbols are recognized only after box symbols. Bresler *et. al.* [17] also first recognize possible box and arrow symbols, and leave text recognition as a last step. After symbol candidates are identified, the best symbol combination is selected through a max-sum optimization process.

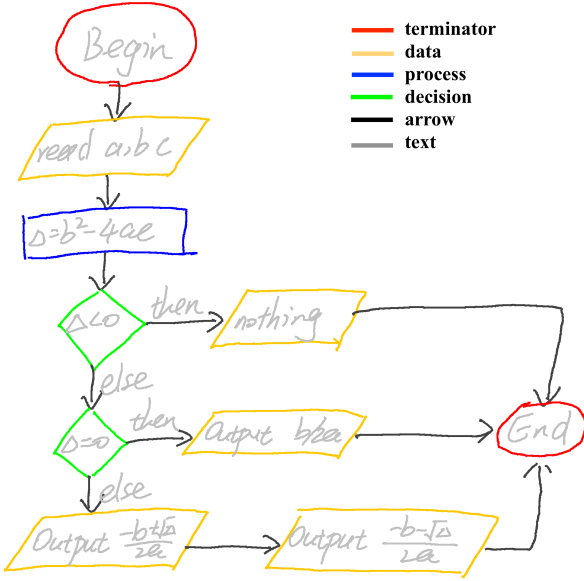


Fig. 2. Flowchart example. Strokes are colored according to the symbol type they belong to.

An interesting example of graph grammar use is described in [11]. The authors propose an attributed graph grammar that allow attributes to be passed from node to node in the grammar, both vertically and horizontally, to describe a scene of man-made objects. Projection of rectangles are used as primitives. However, passage of attributes must be evaluated during parsing, making the parsing algorithm be context-dependent. In [10] entity-relationship diagrams are modeled by a context-sensitive graph grammar with the “left-hand side of every production being lexicographically smaller than its right-hand side”. A critical part of the parsing algorithm is to find matchings of the right-hand side of a rule to replace the left-hand-side, making it very complex.

The above review on some characteristics related to the recognition of 2D data illustrates that existing methods present several restrictions and limitations and clearly can not be easily transposed to the recognition of other families of graphics.

In the method proposed in this work, instead a CYK-based algorithm (that assumes a grammar in CNF), we define a graph grammar and use a top-down parsing algorithm, similar to the one of [8], but without assuming any ordering of the input strokes. To avoid context-aware algorithms during parsing, we consider stroke partitions drawn from a previously built hypotheses graph (see Section 3.4) to match the right-hand side of the rules. By doing this, we decouple the parsing algorithm from the particularities of the family of graphics, and achieve independence of the target notation. In addition, it is important to note that target domain knowledge can be fully exploited in the graph

grammar definition and hypotheses graph building. This characteristic makes the proposed method general enough to be applied to the recognition of a variety of graphic notations.

### 3 THE PROPOSED RECOGNITION FRAMEWORK

The proposed recognition framework is composed of three main parts: (1) **hypotheses graph generation**, (2) **graph parsing**, and (3) **optimal tree extraction**. In the first part, stroke groups that are likely to represent symbols, and a set of possible relations between these stroke groups are identified and stored as a graph, called hypotheses graph. In the second part, valid interpretations (potentially multiple of them) are built from the hypotheses graph by parsing it according to a graph grammar. The interpretations found are stored in a parse forest. Then, in the third part an optimal tree is extracted from the parse forest, based on a scoring function.

We first discuss the two main input data of the framework, a handwritten input graphic to be recognized (a set of strokes) and a graph grammar, and then detail the three parts, keeping an abstraction level suitable for the recognition of a variety of graphics in general. Concepts are illustrated using mathematical expressions as examples. Implementation related details regarding the application of the framework to the recognition of mathematical expressions and flowcharts are presented in Section 4.

#### 3.1 Stroke set

Online handwriting consists of a set of strokes. Each stroke is, typically, a sequence of point coordinates sampled from the moment a writing device (such as a stylus) touches the screen up to the moment it is released. We assume that each stroke belongs to only one symbol (this assumption is common when dealing with handwritten graphics). Otherwise, a preprocessing step could be applied to split a stroke that is part of two or more symbols. These concepts are illustrated in Figures 3a and 3b.

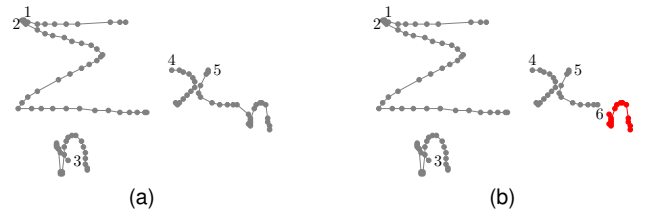


Fig. 3. Handwritten expressions representing  $\sum_n x_n$ . Each expression is composed of a set of strokes, where each stroke is a sequence of bidimensional coordinates (dots in gray). In (a), stroke 5 belongs to two symbols. In (b), each stroke belongs to only one symbol.

#### 3.2 Graph grammar model

A graph grammar [28] defines a *language* of graphs. We denote a graph  $G$  as a pair  $(V_G, E_G)$ , where  $V_G$  represents the set of vertices of  $G$  and  $E_G$  represents the set of edges of  $G$ . A labeled graph is a graph with labels in its vertices and edges. Hereafter we assume labeled graphs, with labels

defined by a function  $l$  that assigns symbol labels (in a set  $SL$ ) to vertices and relation labels (in a set  $RL$ ) to edges. We define a family of graph grammars, called *Graphic grammars*, to model graphics as labeled graphs.

**Definition 1.** A *graphic grammar* is a tuple  $M = (N, T, I, R)$  where:

- $N$  is a set of non-terminal nodes (or non-terminals);
- $T$  is a set of terminal nodes (or terminals), such that  $N \cap T = \emptyset$  (for convenience we denote elements in  $T$  using the same names used for the labels in  $SL$ );
- $I$  is a non-terminal, called initial node;
- $R$  is a set of production (or rewriting) rules of the form  $A := B$  where  $A$  is a non-terminal node and  $B = (V_B, E_B)$  is a connected graph with label  $l(v) \in N \cup T$  for each  $v \in V_B$ , and label  $l(e) \in RL$  for each  $e \in E_B$ .

Note that  $M$  is a context-free graph grammar [28]. The language defined by a graphic grammar  $M = (N, T, I, R)$  is a (possibly infinite) set of connected labeled graphs and is denoted  $\mathcal{L}(M)$ . Similarly to string grammars, a labeled graph  $G$  belongs to  $\mathcal{L}(M)$  if  $G$  can be derived (or generated) from the initial non-terminal node  $I$  by successively applying production rules in  $R$ , until obtaining a graph with only terminal nodes.

Figure 4 shows a graphic grammar that models simple arithmetic and logical expressions. Each production rule defines the replacement of a non-terminal, a single vertex graph  $G_l$  at the left hand side (LHS) of the rule, with a graph  $G_r$  at the right hand side (RHS).

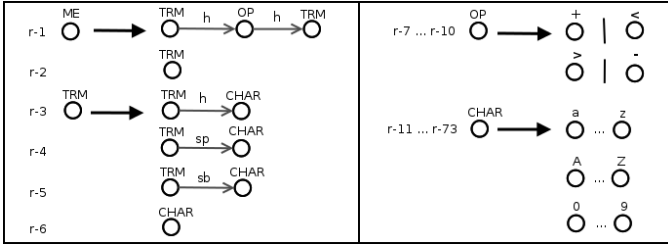


Fig. 4. Graph grammar that models basic mathematical expressions. The grammar is defined by non-terminals  $N = \{ME, TRM, OP, CHAR\}$ , relation labels  $RL = \{sp, sb, h\}$ , terminals  $T = \{+, -, <, >, a, \dots, z, A, \dots, Z, 0, \dots, 9\}$ , rules  $R = \{r-1, \dots, r-73\}$ , and  $ME$  at the left hand side graph of rule  $r-1$  is the initial node. Abbreviations:  $ME$  = mathematical expression,  $sp$  = superscript,  $sb$  = subscript,  $h$  = horizontal,  $TRM$  = term,  $OP$  = operator,  $CHAR$  = character.

Figure 5 shows a graph generation process using the grammar of Figure 4. Rules are applied sequentially, starting with non-terminal  $ME$ , until all elements in the generated graph are terminals. Dashed arrows correspond to edges that link the replacing graphs with the host graph.

The definition of how a replacing graph should be linked to a host graph  $G$  is called *embedding* [28], and it should be specified for each production rule. Formally, given a production rule  $G_l := G_r$ , its application consists in replacing a subgraph  $G_l$  of  $G$  with  $G_r$  and the embedding defines how  $G_r$  will be attached to  $G \setminus G_l$ . The *attachment* may be defined by a set of edges that link the replacing graph  $G_r$  to  $G \setminus G_l$ . For instance, Figure 6 shows two different embeddings for

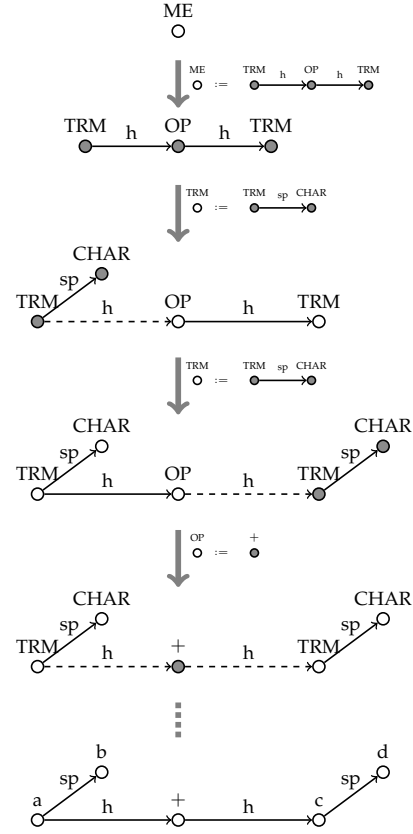


Fig. 5. Generation of a graph that represents the expression  $a^b + c^d$ . At each rule application, the replacing graph nodes are depicted in dark gray. Edges that link the replacing graph with the host graph are depicted with dashed arrows. Rule applications after the fourth one are not shown.

a same production rule, and the graphs generated for each embedding.

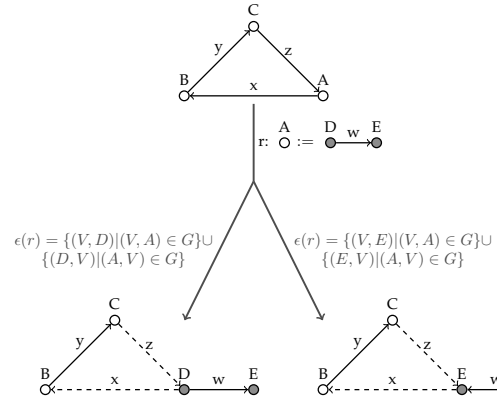


Fig. 6. Graph transformation with two different embeddings. The top graph is transformed through rule  $r$ . Each embedding defines edges between vertices that are linked to vertex  $A$  of the top graph with vertex  $D$  (left hand side embedding) or  $E$  (right hand side embedding) of the replacing graph. Dashed arrows represent the edges defined by each embedding.

The embedding specification depends on the desired language. It is possible to define a same embedding specification for all rules, as we do for mathematical expressions (see Section 4). An embedding can also take spatial



information into consideration, for example by including edges only between spatially close vertices. More detailed examples of embeddings are provided in Section 4, through applications to the recognition of mathematical expressions and flowcharts. To ensure that the generated graphs are connected, we assume that every embedding is specified in such a way that its application generates connected graphs.

### 3.3 Hypotheses graph generation

Given a set of strokes  $S$ , we define a hypotheses graph as an attributed graph  $H = (V_H, E_H)$ , where  $V_H$  is a set of symbol hypotheses and  $E_H$  is a set of relation hypotheses computed from  $S$ . Each symbol hypothesis  $v \in V_H$  corresponds to a subset of  $S$ , denoted as  $stk(v)$ , and has as an attribute a list  $L(v) = \{(l_i, s_i), i = 1, \dots, k_v\}$  of likely interpretations. Each of these interpretations  $(l_i, s_i)$  consists of a symbol label  $l_i \in SL$  and its respective likelihood score  $s_i \in [0, 1]$ . Note that a stroke may be shared by multiple symbol hypotheses. Relation hypotheses (edges in  $E_H$ ) are defined over pairs of disjoint symbol hypotheses (i.e., hypotheses such that their stroke sets are disjoint), and also have as an attribute a list of likely relation interpretations denoted  $L(e)$ . Relation labels are in  $RL$ . Figure 7 shows a handwritten mathematical expression and a hypotheses graph calculated from it.

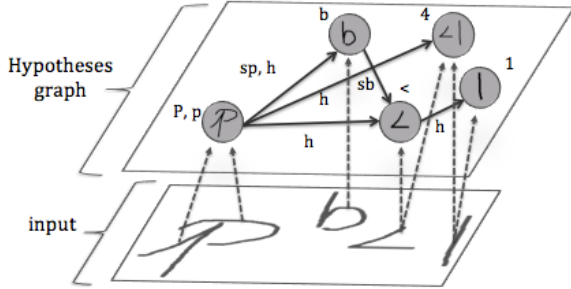


Fig. 7. Hypotheses graph example. Vertices represent symbol hypotheses and edges represent relations between symbols. The labels associated to symbols and relations indicate their most likely interpretations.

To build a hypotheses graph, machine learning methods are effective in identifying groups of strokes that may form symbols and, similarly, relations among them (see application example in Section 4). Since many stroke groups do not correspond to an actual symbol and many pairs of symbols are not directly related each other within a graphic, rather than training classifiers to identify only true hypotheses, those that do not represent any symbol or relation can be included as elements of an additional class, called *junk*. Training data can be extracted from within the graphic, together with surrounding context, in order to improve rejection of false hypotheses. As will become clear later, hypotheses graphs play an important role to constrain the search space during the parsing process. A high precision and recall in the identification of symbol hypotheses and relations is thus desirable to efficiently constrain the search space.

#### 3.3.1 Label list pruning

To define the labels and respective likelihood scores of symbol and relation hypotheses, we could use the confi-

dence scores returned by the respective classifiers. However, to manage complexity, only class labels that present high confidence scores should be kept. Selecting the labels to be kept based on a fixed global confidence threshold value is not adequate since label distributions vary greatly among symbols and relations. An effective method to select the most likely labels for each hypothesis  $h$  is described next.

Let  $\{(l_i, s_i), i = 1, \dots, n_h\}$  be the pairs of labels and respective scores initially attributed to  $h$ , sorted in descending order according to the likelihood scores  $s_i$ . Then, given a distribution threshold  $tr$  (between 0 and 1), we define the minimum number of  $k$  top ranked labels whose confidences sum up to at least  $tr$ :

$$k = \arg \min_x \sum_{i=1}^x s_i > tr \quad (1)$$

Hypothesis  $h$  is rejected if it presents highest score for the *junk* class label and if that score is above the threshold  $tr$ . Otherwise, we set  $L(h) = \{(l_i, s_i) : i = 1, \dots, k\}$ . We define label pruning thresholds  $t_{\text{syimb}}$  for symbols and  $t_{\text{rel}}$  for relations.

### 3.4 Graph parsing

The goal of the parsing process is to build a parsing tree that explains the set of input strokes  $S_{\text{input}}$ , according to a grammar. Since there might be more than one interpretation, multiple trees might be generated, possibly sharing subtrees each other. Thus, they will be stored in a parse forest.

Figure 8 shows a parse forest calculated from the hypotheses graph of Figure 7, using the graph grammar of Figure 4. As can be seen, the root node (top of the figure) corresponds to the starting non-terminal  $ME$ . Two branches are generated from rules associated to  $ME$ . The left branch is generated by applying rule 2 and the right branch by applying rule 1. Note that, for each rule, any of the resulting partition of the strokes induces a graph that is isomorphic to the RHS graph of the respective rule. The same principle holds for the remaining of the nodes.

The parsing process follows a top-down approach. To understand the parsing process, a key step is to understand how a stroke set is partitioned when a rule is applied. More specifically, given a set of strokes  $S$  and a non-terminal  $NT$ , for each rule  $A := B$  associated to  $NT$ , we must find every partition of  $S$  that is a valid matching to  $B$ . A partition of  $S$  is a matching to  $B$  if its number of parts is equal to the number of vertices of  $B$ , so that each part can be assigned to one vertex in  $B$ . A matching is valid if the following two conditions hold: (1) the partition of  $S$  induces a graph that is isomorphic to  $B$ , and (2) each subset of strokes assigned to a vertex of  $B$  must be parsable according to the grammar.

Supposing the number of vertices in  $B$  is  $k$  and the number of strokes in  $S$  is  $n$ , without any constraint, the total number of possible stroke partitions to be examined to generate the valid matchings would be  $O(k^n)$ . Exhaustively examining each of these partitions is not computationally practical.

A main strategy of our method is to constrain the number of partitions to be examined with the aid of the hypotheses graph. We assume that all meaningful interpretations are present in the hypotheses graph as a subgraph. Thus, before

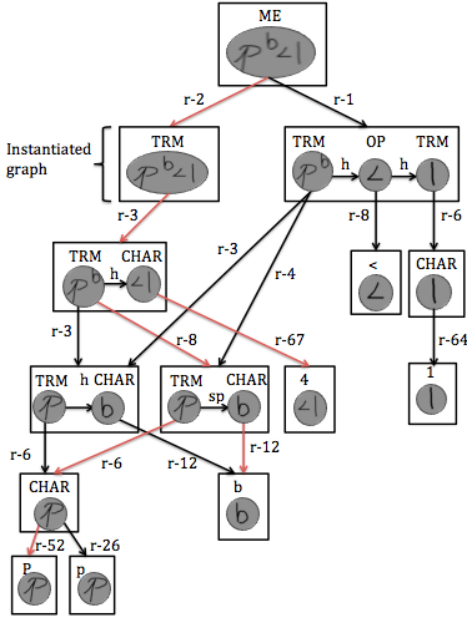


Fig. 8. A parse forest representing multiple interpretations of a mathematical expression. Labels on arrows refer to grammar rules of Figure 4. Red arrows represent a parse tree that corresponds to the interpretation “ $P^b 4$ ”.

starting the parsing process, we build the set of all stroke groups, denoted hereafter as  $STK$ , underlying any valid connected subgraph of  $H$ . Note that these stroke groups must not contain repeated strokes, i.e., a valid subgraph is one in which a same stroke is not present twice. Furthermore, not all stroke groups will be necessarily parsable. The relation between two stroke groups is also recorded in  $STK$  as being the same between the corresponding subgraphs. Hence, during parsing, the search space of valid matchings will be restricted to those present in  $STK$ . Once a valid matching is found, an instance of  $B$ , which we call *instantiated graph*, will be recursively parsed and will become a *parsed graph* when each of its vertices is successfully parsed.

The complete algorithm is described next. For the sake of simplification, we will assume that the input grammar contains only two types of rules: *terminal* and *non-terminal*. Terminal rules are productions of the form  $A := b$ , where the RHS graph  $b$  is a single vertex graph, with labels in the terminal set, such as rules from r-7 to r-73 of the grammar of Figure 4. Non-terminal rules are productions of the form  $A := B$ , where  $B$  is a graph containing one or more vertices, each of them with non-terminal labels, such as rules r-1 to r-6 of the grammar of Figure 4. Thus, Algorithm 1 considers only these two types of rules. Its extension to treat rules that contain both terminals and non-terminals in its right-hand side is a straightforward combination of the previous two cases.

Algorithm 1 receives as inputs a stroke set  $S = \{stk_1, \dots, stk_n\}$  and a non-terminal  $NT$ . Initially, the set of strokes is the whole input set  $S_{input}$  and the non-terminal is the starting node  $I$ . Then, it applies each of the production rules that have  $NT$  as the LHS graph and returns a set (*parsedG*) containing all parsed graphs, together with the respective rules that “generated” them.

---

**Algorithm 1 : parseGraphic( $S, NT$ )**

Parses a set of strokes  $S$  from a non-terminal  $NT$

---

**Input:** ( $S, NT$ )

**Output:** *parsedG* =  $\{(G_1, r_1), \dots, (G_q, r_q)\}$

```

1: parsedG  $\leftarrow \emptyset$ 
2: if parsed[( $S, NT$ )] then
3:   parsedG  $\leftarrow TBL[(S, NT)]$ 
4: else
5:   for all rule in rulesWithLHS( $NT$ ) do
6:     if rule is  $A \rightarrow b$  then
7:       if  $l(b) \in L(S)$  then
8:          $G \leftarrow buildGraph(S, l(b))$ 
9:         parsedG  $\leftarrow parsedG \cup \{(G, rule)\}$ 
10:      end if
11:    else
12:      for all  $G$  in validMatchingInstances( $S, B = RHS(rule)$ ) do
13:        if  $\forall v \in V_G, parseGraphic(stk(v), l(v)) \neq \emptyset$  then
14:          parsedG  $\leftarrow parsedG \cup \{(G, rule)\}$ 
15:        end if
16:      end for
17:    end if
18:  end for
19:   $TBL[(S, NT)] \leftarrow parsedG$ 
20:  parsed[( $S, NT$ )]  $\leftarrow True$ 
21: end if
22: return parsedG

```

---

To avoid recomputation, a global table  $TBL$  indexed by pairs ( $S = \{stk_1, \dots, stk_n\}, NT$ ) is used. An entry in  $TBL$  is of the form  $TBL[(S, NT)] = \{(G_1, r_1), \dots, (G_q, r_q)\}$  where  $G_i$  is a parsed graph and  $r_i$  is the rule that “generated”  $G_i$ . At the end of the algorithm, if the pair ( $S, NT$ ) is not parsable, the corresponding entry in  $TBL$  is empty.

Lines 2-3 verify if the pair ( $S, NT$ ) has already been processed. If so, results are retrieved from  $TBL$  and returned. Otherwise, lines 5-18 iterate over the rules that have  $NT$  in its LHS graph. If the rule is of terminal type (lines 6-10), it suffices to check if the RHS vertex label,  $l(b)$ , is contained in the set of labels  $L(S)$  attributed to the underlying stroke set. This verification is done by checking if the stroke set  $S$  corresponds to a vertex in the hypotheses graph and if the label set of the corresponding vertex includes  $l(b)$ . Then a single vertex graph is built and stored together with the rule in *parsedG*. If the rule is of non-terminal type (lines 11-17), for each valid matching between  $S$  and  $B$  (line 12) we verify if the instantiated graph is parsable. The parsing result, either a list of parsed graphs, or an empty list (in case of parsing failure), is added to  $TBL$ . As already mentioned, table  $TBL$  is used to avoid parsing recomputation of pairs ( $S, NT$ ). At the end of the parsing process, the parse forest can be extracted from  $TBL$  by traversing it starting from index ( $S_{input}, I$ ).

### 3.4.1 Pruning strategies

Besides constraining the partitions to be examined to only those formed by stroke groups that underlie a subgraph of  $H$ , there are other strategies that can be used to speed up computation. For example, determining the maximum and minimum size of non-terminal nodes is a strategy that has been previously used in text parsing [29]. The sizes, in terms of graphic symbols or strokes, can be computed directly from the grammar. Based on these numbers, during parsing any stroke subsets that are out of the min-max ranges do not need to be evaluated. This information can be calculated when building  $STK$ . Moreover, to find valid



matching partitions, the minimum and maximum sizes of the stroke subsets already matched to some vertices can be used to determine the minimum and maximum size of the stroke groups that still can be matched to the rest of the nodes.

Another useful information is to explore the knowledge that a non-terminal can generate only a specific subgroup of the terminals. For instance, in the grammar of Figure 4, non-terminal  $OP$  can generate only symbols  $+$ ,  $-$ ,  $<$ , or  $>$ . Thus, stroke subsets that do not contain any hypothesis with one of such labels as terminals are not evaluated during the parsing process. Analogously, stroke groups that correspond to symbol and relation hypotheses with high mean junk score can be disregarded. Specifically, stroke subsets with a certain number (five, for example) symbol hypotheses, having mean junk score, including both symbol and relation labels, above a given junk threshold  $t_{\text{junk}}$  will not be considered. This pruning is mainly useful when the symbol and relation hypotheses have a large number of labels. High mean junk score indicates that it is unlikely that the underlying group of strokes is parsable.

### 3.5 Optimal parse tree extraction

Once a parse forest is built, the final step consists in traversing it to extract the best tree (interpretation). To characterize what is an optimal tree (best interpretation), we first define a cost function for trees. Roughly stating, an interpretation will be considered of low cost if its corresponding parse tree includes substructures with high confidence scores.

We introduce a few notations that will be helpful. Let  $x$  denote a node in the parse forest. Let  $G_x = (V_x, E_x)$  be the graph instantiated at node  $x$ . Each vertex  $v \in V_x$  has an underlying set of strokes,  $stk(v)$ . For each terminal vertex  $v \in V_x$  there will be a pair  $(label(v), score(v)) \in SL \times [0, 1]$  and for each edge  $e \in E_x$  will be a pair  $(label(e), score(e)) \in RL \times [0, 1]$ .

The cost of a tree can be computed bottom-up. We first define individual costs relative to symbols and relations, and then define how to combine the two to determine the cost of a tree. Let  $t$  be a parse tree and let  $x$  be a node in  $t$ . Let  $child(x)$  denote the child nodes of  $x$ . The subtree in  $t$  with root at  $x$  is denoted  $t_x$ . We first assign to a node  $x$  a symbol cost  $J_s(x)$ :

$$J_s(x) = \begin{cases} -\log score(v), & \text{if } x \text{ is terminal,} \\ & \text{with } V_x = \{v\}, \\ \sum_{y \in child(x)} J_s(y) & \text{if } x \text{ is non-terminal,} \end{cases} \quad (2)$$

and a relation cost  $J_r(x)$ :

$$J_r(x) = \sum_{e \in E_x} -\log score(e) + \sum_{y \in child(x)} J_r(y) \quad (3)$$

Then, the cost of  $t_x$  is defined as

$$J(t_x) = \frac{\alpha}{n_s} J_s(x) + \frac{1-\alpha}{n_r} J_r(x) \quad (4)$$

where  $n_s$  and  $n_r$  are, respectively, the number of symbols and relations under  $t_x$ . Parameter  $\alpha$  weights both types of costs, and could be adjusted to give more relevance to one or to the other.

An example of a tree is shown in Figure 9. Its root node is  $x_1$  and thus the tree is denoted  $t_{x_1}$ . The cost of tree  $t_{x_1}$  is given in Eq. 5.

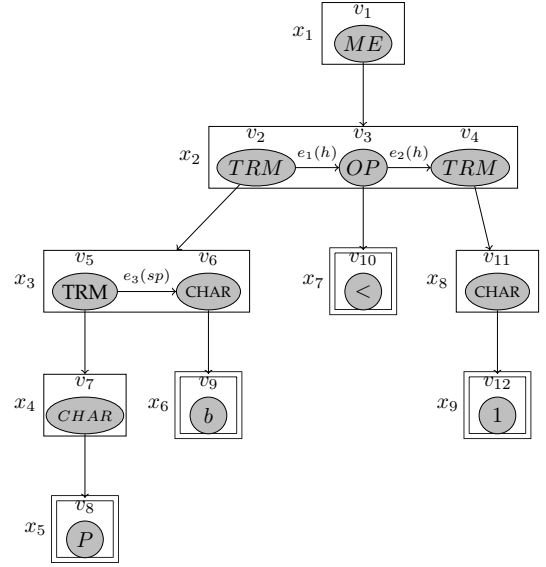


Fig. 9. Parse tree of expression  $P^b < 1$ , extracted from the parse forest of Figure 8. Nodes are indexed as  $x_i$ ,  $i = 1, \dots, 9$ . Similarly, vertices and edges of the instantiated graphs are respectively indexed as  $v_j$ , for  $j = 1, \dots, 12$ , and  $e_k$ , for  $k = 1, \dots, 3$ . Nodes with terminal symbols are depicted with double line borders.

$$J(t_{x_1}) = \frac{\alpha}{4} \left( J_s(v_8) + J_s(v_9) + J_s(v_{10}) + J_s(v_{12}) \right) + \left( \frac{1-\alpha}{3} \right) \left( J_r(e_1) + J_r(e_2) + J_r(e_3) \right) \quad (5)$$

In order to extract the best tree, the cost of each tree in the parse forest must be computed. Since the trees in the parse forest share subtrees, this fact can be explored to avoid computing the cost of a shared subtree repeatedly. In addition, from an application point of view, being able to efficiently retrieve a number of best parse trees rather than just the best one is often desirable. We borrow ideas from the tree extraction technique, in the context of string grammars, proposed by *Boullier et al.* [30]. Given a parse forest, they proposed a method that builds a new parse forest with a fixed number of  $n$ -best trees, using a bottom-up approach. The resulting parse forest can be further processed to improve the recognition result, for example, by doing a re-ranking of the trees, a processing that could be too expensive to be done in the original parse forest.

Note that there might be multiple subtrees with root at a node  $x$  in the parse forest. For instance, in the parse forest of Fig 8, the vertex in the bottom left non-terminal node graph has two possible derivations (“P” or “p”). Whenever there are multiple derivations from a non-terminal vertex, only one of them will be present in a parse tree. Thus, given a node  $x$  in the parse forest, let us denote by  $t_x^{(i)}$ ,  $i \in I_x$ , the spanned trees from  $x$ . The number of possible trees in the forest is combinatorial with respect to the multiple subtrees spanned from the nodes in a path from the root node to a leaf node.

We use a bottom-up approach to compute, for each node  $x$  in the forest, a list of subtrees spanned from it. This information is kept as a table in the node, and each row of the table stores information to recover one of the spanned trees (specifically, it stores the partition of the stroke set resulting from the corresponding derivation). After the bottom-up process finishes, individual trees can be extracted by performing a top-down traversal, starting from each row of the table at the root node of the forest. The best tree, according to the specified cost, is the one recovered by starting the traversal from the first row of the table.

However, since there might be a large number of parse trees in the forest, a naive application of the method described above may be computationally prohibitive. To overcome this problem, a pruning strategy can be applied during the bottom-up step to keep table sizes manageable: for each table, spanned trees that have a cost much higher than the best tree are discarded. To compute relative differences of cost, let  $\min J(x)$  be the minimum cost tree spanned from  $x$ . Then, given  $t_{\text{pr}} \in [0, 1]$ , a spanned tree  $t_x^{(i)}$  is kept if

$$|J(t_x^{(i)}) - \min J(x)| < t_{\text{pr}} * \min J(x). \quad (6)$$

This strategy resembles the one proposed in [30], but it differs in the sense that while they keep a fixed number of best trees, we keep only the relatively likely ones. The more ambiguous the input, the more parse trees are kept. The pruning threshold  $t_{\text{pr}}$  can be empirically estimated.

## 4 APPLICATIONS

The application of the framework requires the definition of some key elements. First, a graph grammar that models the family of graphics to be recognized must be defined. A set of labels for the relations ( $RL$ ) and for the symbols ( $SL$ ), including *junk*, must be defined. Second, a hypotheses graph generated from the set of input strokes, with symbol labels in  $SL$  and relation labels in  $RL$ , must be provided. Terminal nodes of the grammar are named using the labels in  $SL$ , while edges in the graphs of the grammar are labeled using labels in  $RL$ . For parsing, an embedding method must be defined for each grammar rule. In this section, we detail how these elements as well as important parameter values have been defined for the recognition of mathematical expressions and flowcharts. Results and discussions are presented in the next section. The grammars in `xm1` format are available at [www.vision.ime.usp.br/~frank.aguilar/grammars/](http://www.vision.ime.usp.br/~frank.aguilar/grammars/).

Before applying the recognition method itself, we applied to the set of strokes the smoothing and resampling methods described in [31]. Smoothing removes abrupt trajectory changes in the strokes and resampling makes point distribution uniform – equally spaced – along the strokes. In the evaluating datasets, each stroke belongs to only one symbol; thus no additional preprocessing was needed.

### 4.1 Recognition of mathematical expressions

#### 4.1.1 Dataset and Grammar

We use the CROHME-2014 dataset [32]. It consists of handwritten expressions divided into training and test sets, with 9,507 and 986 expressions, respectively. The expressions include 101 symbol classes, and six relation classes

(horizontal as in “ $ab$ ”, above as in “ $\overset{x}{\Sigma}$ ”, below as in “ $\underset{x}{\Sigma}$ ”, superscript as in “ $a^b$ ”, subscript as in “ $a_b$ ”, and inside as in “ $\sqrt{x}$ ”). CROHME-2014 dataset provides a string grammar for the corresponding  $\text{\LaTeX}$  expressions. Based on that string grammar, we defined a graph grammar with 205 production rules, including the rules to generate the 101 symbol labels (terminals).

To define the embeddings, we use the concept of baseline. A baseline in a graph is defined as a maximal path whose connecting edges have only the *horizontal* ( $h$ ) label (this definition can be seen as a graph version of the baseline definition of [20]). A baseline is considered nested to a vertex  $v$  if it is connected to  $v$  by an edge  $(v, v')$ , where  $v'$  is the first vertex of the baseline. A baseline that is nested to no vertex is called dominant baseline. Note that a baseline may consist of a single vertex.

Then, the embedding is defined as follows. Let  $r : G_l := G_r$  be a rule and let  $v'$  be the leftmost and  $v''$  be the rightmost vertices of the dominant baseline of  $G_r$ . Let also  $G$  be a graph with an occurrence of  $G_l$ , identified as a vertex  $u \in V_G$ . The embedding associated to the application of rule  $r$  on  $G$  replaces  $u$  with  $G_r$ , generating an updated graph  $G'$ , such that  $V_{G'} = V_G \setminus \{u\} \cup V_{G_r}$  and  $E_{G'} = [E_G \setminus (\{(u', u) : u' \in V_G\} \cup \{(u, u') : u' \in V_G\})] \cup \epsilon$  where

$$\epsilon = \{(u', v') : (u', u) \in E_G\} \cup \{(v'', u') : (u, u') \in E_G\}. \quad (7)$$

In other words, all edges that were incident on  $u$  will be made incident to  $v'$  and all edges that were originated from  $u$  will be made originating from  $v''$ .

#### 4.1.2 Hypotheses graph building

To generate the hypotheses graph, we used the symbol segmentation and classification methods described in [33], [34], along with the spatial relation classification methods described in [35]. They are based on multilayer neural networks with shape context descriptor [36], and images created from symbols and relations, including neighboring strokes to be used as contextual information. The networks use a softmax output which is then converted to a cost measure (applying the negative logarithm to the output) in order to be used in the cost function defined in Eq. 4.

An important parameter to build the hypotheses graph is the symbol and relation label pruning thresholds,  $t_{\text{symp}}$  and  $t_{\text{rel}}$  (see Eq. 1). These threshold values determine how many and which labels will be attached to each vertex and edge. Since during the parsing process the partitions of the stroke set and labels are constrained by the hypotheses graph, the achievable maximum recognition rates are bounded by possibilities encoded in the hypotheses graph.

From the training set, we randomly selected 950 expressions (about 10%) to serve as a validation set and used the rest for training. Using the trained symbol and relation classifiers, we evaluated the effect of varying values of  $t_{\text{symp}}$  and  $t_{\text{rel}}$  on the validation set. For each threshold value we computed the symbol, relation and complete expression recalls, that is, how many of each of these components were present in the hypotheses graph.

Figure 10 shows the results relative to this evaluation, over  $t_{\text{symp}}$  in the range  $[0.4 - 1]$  (for values less than 0.4,

the performance was similar to the case of 0.4) and  $t_{rel}$  in the range  $[0.1 - 1]$ . Note that this evaluation is concerned with verifying how many of the elements of interest are, in fact, present in the hypotheses graph; it is not related with parsing.

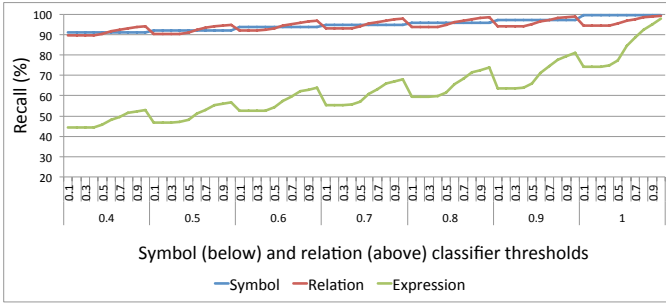


Fig. 10. Symbol, relation and expression level recall of the hypotheses graph generation step. For each symbol classification threshold  $t_{symb}$  in the range  $[0.4 - 1.0]$ , relation classification threshold  $t_{rel}$  is varied in the range  $[0.1 - 1.0]$ .

We can see in Figure 10 that even for the lowest threshold values the recall of symbols and relation is about 90%. For complete expressions (i.e. all symbols and relations of the expressions are in the hypothesis graph), however, the recall for the lowest threshold values is 40%. If symbol classification threshold is set to 1, 99,75% of the symbols are correctly included. Since in this case no stroke group is rejected, 99,75% is also the percentage of symbols identified by the stroke grouping method. If, in addition, we also set the relation classification threshold to 1, almost all relations and expressions are included (99,45% and 98.11%, respectively).

#### 4.1.3 Graph parsing and tree extraction

We also analyzed the effect of different values of  $t_{symb}$  and  $t_{rel}$  on the recall after parsing. We set the maximum value for  $t_{symb}$  to 0.98 and for  $t_{rel}$  to 0.85, as parsing large expressions with thresholds larger than those takes much time to be considered in a real application. In this evaluation, for optimal tree extraction we set  $\alpha = 0.5$  (same weight for the symbol and relation costs, see Eq. 4) and  $t_{pr} = 0.1$  (tree pruning threshold, see Eq. 6). Figure 11 shows the expression recall obtained by the parsing method and the corresponding recall obtained by the hypotheses graph generation step (note that the second indicates the maximum achievable recall). Although for values above  $t_{symb} = 0.9$  and  $t_{rel} = 0.8$  no considerable improvements are observed in the parsing recall, the gap between hypotheses graph recall and parsing recall increases up to about 40%. Thus, we chose  $t_{symb} = 0.98$  and  $t_{rel} = 0.85$ , as these values allow to keep more hypotheses and can be useful during parsing of unseen expressions (better generalization).

Using  $t_{symb} = 0.98$  and  $t_{rel} = 0.85$ , we have also evaluated the effect of different values of  $t_{pr}$  (tree pruning threshold) and  $\alpha$  (weighting in the cost function) on tree extraction on validation set. Through this evaluation, we set  $t_{pr} = 0.1$  and  $\alpha = 0.4$  (this choice was based on the best expression recall).

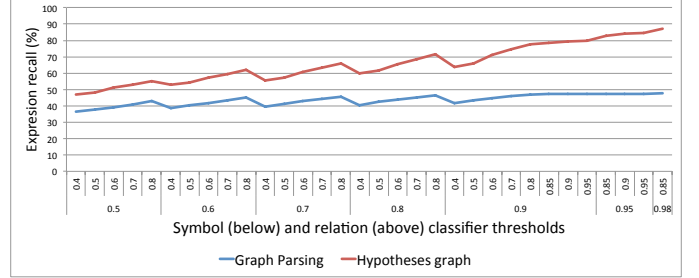


Fig. 11. Expression recall obtained at graph parsing and hypotheses graph generation steps, for different symbol and relation thresholds.

## 4.2 Recognition of flowcharts

### 4.2.1 Dataset and grammar

We use the flowchart dataset described in [37]. The dataset includes 7 symbol classes (*arrow*, *connection*, *data*, *decision*, *process*, *terminator*, and *text*), and three relation classes (*Src*, *Targ*, and *AssTxt*). An example was presented in Section 2 (Fig. 2), with strokes colored according to the symbol type they belong to. In this dataset, relations in each flowchart are established between “adjacent” symbols. For instance, in the flowchart of Figure 2, *Src* and *Targ* relations are defined between the top *arrow* and a *terminal* and *data*, respectively. In the same way, an *AssTxt* relation is defined between the top *terminal* and the *text* inside it. The flowcharts have been written by 36 people, and the dataset is divided into a train set with 248 and a test set with 171 flowcharts. The total number of symbols is about 9,000.

As described in Section 2, *text* symbols have different characteristics than other flowchart symbols, and they are usually recognized through specific methods. Since flowchart recognition is addressed in this work with the aim of illustrating the application of the proposed framework, we are not specially concerned with recognition performance. Thus, we have opted on removing strokes corresponding to text symbols, as well as the respective relations (*AssTxt*) from the flowcharts. Symbol class *text* and relation class *AssTxt* were not considered. We note, however, that it would be equally possible to parse the integral flowchart without any changes in the parsing and tree extraction steps once adequate symbol and relation classifiers are developed for texts.

In contrast to the CROHME-2014 dataset, we found no grammar defined for the flowchart dataset. Thus, we defined a grammar with 16 production rules, where six of them generate the terminal symbols. The embedding is defined in a similar way to the one defined for mathematical expressions, except for the set of edges to be added. Let  $u$  denote the vertex to be replaced in  $G$  and  $v_i \in V_{G_r}$  the vertices in the replacing graph. The edges to be added are defined by:

$$\epsilon = \{(u', v) \mid (u', u) \in E_G \text{ and } v = \arg \min_{v_i} \text{cost}_r(u', v_i)\} \cup \{(v, u') \mid (u, u') \in E_G \text{ and } v = \arg \min_{v_i} \text{cost}_r(v_i, u')\}$$

where  $\text{cost}_r(u, v)$  is the minimum relation cost among relations between a symbol hypothesis under  $u$  and a symbol hypothesis under  $v$ .

#### 4.2.2 Parameter adjustment

For symbol segmentation and classification we used the same method used for mathematical expressions. Symbol and relation classifier thresholds,  $t_{\text{symb}}$  and  $t_{\text{rel}}$ , were set both to 0.95, following the same scheme as done with mathematical expressions.

An important performance difference between the two applications is the relative low accuracy of the flowchart relation classifier compared to the mathematical expression relation classifier. This difference is due to the fact that arrows in flowcharts present a high shape variance and the classifiers we used, which are mainly based on shape histograms of the symbols [35], do not generalize well. We alleviate this deficiency by setting  $t_{\text{rel}} = 0.95$  (in mathematical expressions, we set  $t_{\text{rel}} = 0.85$ ), in order to keep more labels. We also applied the pruning method based on the mean junk score of groups with five or more symbols hypotheses, with  $t_{\text{junk}} = 0.25$  (see Section 3.4.1) to cope with the large number valid partitions. For tree extraction, best validation results were achieved with  $\alpha = 0.8$ , placing more weight to symbol classifier scores than to relation classifier scores, and tree pruning threshold  $t_{\text{pr}} = 0.1$  (same value as in the case of mathematical expressions).

## 5 RESULTS AND DISCUSSIONS

Using the datasets, grammars and parameters as described in the previous section, we applied the recognizers on the test set of the respective applications. Here we present and discuss the results.

### 5.1 Recognition of mathematical expressions

Table 1 shows expression level recognition rates including those reported in the CROHME-2014 competition [32]. The competing systems are identified as I, . . . , VII, as reported in the competition results. The four error columns indicate recognition rates considering recognition with up to 0, 1, 2 and 3 errors, respectively,

TABLE 1  
Expression level recognition rates on the test set of CROHME-2014 competition: competing systems and our method

System	0	# errors		
		≤ 1	≤ 2	≤ 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

Our method recognized 33.98% of the expressions completely. We note, however, that 78.40% of the generated hypotheses graph include the complete expressions. Thus, we conclude that the tree extraction process is failing in retrieving the correct interpretation. This observation is also consistent with the evaluation performed on the validation set and described in the previous section.

The two best systems, I and III, include statistical models [32]. In particular, system III corresponds to the commercial system *MyScript*<sup>2</sup>, which has been optimized over hundreds of thousands of equations that are not publicly available. The statistical information used by both systems could explain, at some extent, their better performance. Nevertheless, our method is very close in performance to system I, the best one among those trained exclusively with CROHME-2014 dataset.

We also note that about 15% of the expressions were not correctly recognized due to up to 3 errors. Figure 12 shows some of the expressions that fall in this case. For instance, in the first example, the last term  $b_0$  was recognized as  $b0$ . In the last example, a 9 is mistaken as  $g$ . Thus, we hypothesize that several of the errors could be eliminated by improving the symbol and relation classifiers. However, some cases are difficult to solve even for humans. For instance, in the second example, the relation between  $p$  and  $-1$  is interpreted as *horizontal* and recognized as  $p - 1$ , when the true relation is *subscript* ( $p_{-1}$ ).

(a)  $2^2 b_2 + 2b_1 + b_0 \rightarrow 2^2 b_2 + 2b_1 + b0$

(b)  $n - n_1 - \dots - n_{p-1} \rightarrow n - n_1 - \dots - n_{p-1}$

(c)  $bag_1 \rightarrow bay_1$

(d)  $a_0 + 3a_1 + 9a_2 + 27a_3 = 0 \rightarrow a_0 + 3a_1 + ga_2 + 27a_3 = 0$

Fig. 12. Expressions recognized with a few errors. For each expression, its ground truth and the system's output is shown as: ground truth  $\rightarrow$  system's output.

Figure 13 shows examples of correctly recognized expressions. Our method is able to correctly recognize some ambiguous symbols as well as relations. For instance, in spite of the relation between the subexpressions " $\frac{1}{2}$ " and " $\sin^2(1)$ " of Figure 13c had received higher score as *superscript*, the optimal parse tree interpreted it correctly as a *horizontal* relation.

We also analyzed the most common symbol-to-symbol relation classification errors on test set. A classification was considered an error if either the relation or one of the symbols were wrongly identified. Table 2 shows the ten most frequent errors. Some of the structures are particularly difficult due to the ambiguity at symbol level. For instance, our system often misrecognizes " $\times$ " as " $x$ " and the trigonometric function " $\sin$ " as tree symbols (like " $s$ ", " $i$ " and " $n$ ") related by *horizontal* relation.

In mathematical expressions, the probability of certain symbols or structures be present in particular subexpressions might help solving ambiguities that can not be solved based only on shapes, relations or time related information.

2. <http://www.myscript.com/>

$$\int \frac{1}{p} dp = \int \frac{2}{a} dt$$

(a)

$$\frac{2}{n\pi} (1 - \cos(n\pi))$$

(b)

$$\left[ \frac{1}{2} \sin^2(1) \right] - \left[ \frac{1}{2} \sin^2(0) \right]$$

(c)

$$\pi \int_{-R}^R R^2 dx - \pi \int_{-R}^R x^2 dx$$

(d)

$$f^{(i+k)}(0) = f^{(i)}(0) f^{(k)}(0)$$

(e)

$$a\sqrt{b} \pm c\sqrt{b} = (a \pm c)\sqrt{b}$$

(f)

Fig. 13. Examples of expressions containing potentially ambiguous interpretations that have been correctly recognized by our system.

TABLE 2

Most frequently misclassified spatial relation between symbols on test set. Relation identity is implicitly indicated by the relative positions of the symbols.

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

For instance, the above common errors of missrecognizing symbol “ $\times$ ” as “ $x$ ” or the trigonometric function “ $\sin$ ” as tree separated symbols are examples that could be adequately handled with a statistical model. In the first case, symbol “ $\times$ ” probably appears more frequently between a pair of numbers (or letters) and probably almost never without two *arguments* (one at its left side and another at its right side); in the second case, the three symbols would probably appear more often as the trigonometric function “ $\sin$ ”, rather than for instance, representing the product of three variables  $s$ ,  $i$  and  $n$ . Hence, statistical information calculated from training data could be associated to the production rules of the grammar and then rule probabilities could be considered when ranking the parse trees, by including a new term in the cost function.

### 5.2 Recognition of flowcharts

Table 3 shows the parsing results regarding stroke and symbol labeling accuracy w.r.t. the flowchart test set. It should be noted, however, that we as well as Bresler *et al.* [38] did not consider *text* symbols.

Concerning flowcharts as a whole, our method fully recognized 34% of the flowcharts in the test set. Three examples are shown in Figure 14. They include linear as well as (nested) loop structures. It is interesting to note that varying shapes of arrows such as the one that extends over a

TABLE 3

Comparison of our method and four state-of-the-art methods, w.r.t. stroke and symbol labeling accuracy (%)

System	Stroke labeling	Symbol labeling
Include text recognition:		
Lemaitre <i>et al.</i> [39]	91.1	72.4
Carton <i>et al.</i> [16]	92.4	75.0
Bresler <i>et al.</i> [17]	95.2	82.8
Wang <i>et al.</i> [40]	95.8	84.3
Without text recognition:		
Bresler <i>et al.</i> [38]	-	74.3
Ours	91.1	85.5

large part of the flowchart in the right side of Figure 14a, or very short ones, or yet curvy ones, are correctly identified.

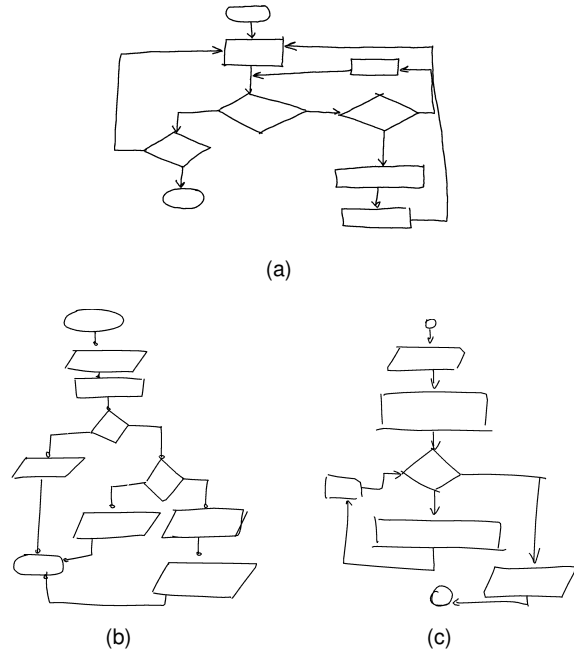


Fig. 14. Examples of flowcharts that have been correctly recognized by our method.

When a true symbol or a true relation is not in the hypotheses graph, the parsing process will fail to recognize the graphic. Figure 15 shows an example of spatial relation and another of a symbol that were not recognized during the hypotheses graph generation.

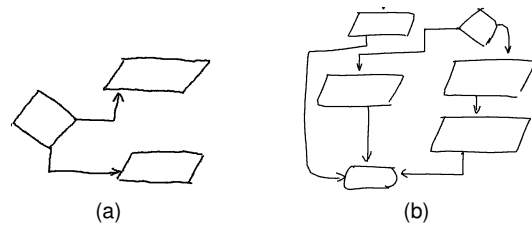


Fig. 15. Parts of flowcharts with missing components in the hypotheses graph. (a) Relation between the top arrow and data symbols has not been identified; (b) the top-center arrow has not been identified.

However, since 67% of the flowcharts were fully represented in the hypotheses graphs, there is a gap of 33% between the achieved rate and the potentially achievable

one. Our explanation for this gap is the fact that although a relatively large number of symbols are correctly recognized (Table 3), many of the true labels in the hypotheses graph presented lower likelihood scores than the false ones, leading to a wrong choice of a tree. Regarding this issue, it is worth to mention that most of the compared methods used specific techniques to identify flowchart symbols, while we used a generic method.

The results indicate, nonetheless, that our method can be applied to flowchart recognition as well. To improve recognition performance, the current bottleneck seems to be in the hypotheses graph generation step. By improving symbol and relation classifiers, a considerable improvement would be possible.

## 6 CONCLUSIONS

We have proposed a general framework for the recognition of online handwritten graphics that is flexible with respect to the family of graphics, offers possibilities to control processing time, and integrates symbol and structural level information in the parsing process. We model graphics as labeled graphs, and the recognition problem as a graph parsing problem guided by a graph grammar. The first step of the framework builds a hypotheses graph that encodes symbol and relation hypotheses computed from the input strokes. The second step parses the set of strokes according to a graph grammar. Rule application is modeled as graph matching between graphs in the rule and graphs induced by partitions of the stroke set. The parsing step typically generates multiple interpretations and thus the third step is for selecting an optimal interpretation. The recognition process is modeled as a bottom-up/top-down approach, where the hypotheses graph relates to the bottom-up part that deals with symbol level information and the graph grammar relates to the top-down part that deals with structural information.

Flexibility with respect to application domains is achieved by encoding all domain specific information in the hypotheses graph and in the grammar, making the parsing method be independent of a particular application. We presented applications of the framework to the recognition of mathematical expressions and flowcharts. Recognition performance are on par with many state-of-the-art methods. Moreover, our evaluations show that there is room for significative improvement. Specifically, in mathematical expression recognition we verified that although 78% of the test expressions were fully represented in the hypotheses graph, only 33.98% of the expressions were fully recognized, corresponding to a gap of almost 45%. Since the parsing algorithm generates all interpretations that are consistent with the grammar, we conclude that the tree extraction step is failing in choosing the correct interpretation. With respect to flowcharts, in many cases the true symbol and relation labels presented very low likelihood or were not even included in the hypotheses graph (it should be noted that no specialized symbol or relation classifier was developed for this application). These evaluations suggest that an immediate improvement would be possible by just improving symbol and relation classifiers. With respect to optimal tree selection, improvement of

symbol and hypotheses likelihood scores will naturally lead to better cost estimation. However, a second improvement could be possible by incorporating in the cost computation a term that captures statistical information with respect to structure occurrence.

Another important feature of our framework is the possibility of managing computational cost. Hypotheses graph is the main tool to reduce the space of partitions to be examined when applying a rule. Only partitions that are present in the hypotheses graph are considered. In addition, there is a set of parameters to control the amount of possibilities to be encoded in the hypotheses graph (symbol and relation label pruning), as well as the number of tree (interpretation) costs to be evaluated (tree pruning). These parameters can be adjusted according to each application particularities.

As future works, we would like to experiment deep neural networks as tools to improve symbol and relation classification in both applications and verify how far recognition rate can be pushed. We would like also to extend the applications to other families of graphics or 2D structures.

## ACKNOWLEDGMENTS

This work has received support from CNPq, Brazil (grant 484572/2013-0). F. Julca-Aguilar thanks FAPESP, Brazil, for the financial support (2012/08389-1 and 2013/13535-0). N.S.T. Hirata is partially supported by CNPq (grant 305055/2015-1).

## REFERENCES

- [1] R. Plamondon and S. N. Srihari, "On-line and off-line handwriting recognition: A comprehensive survey," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 22, pp. 63–84, January 2000.
- [2] K. Marriott, B. Meyer, and K. B. Wittenburg, *A Survey of Visual Language Specification and Recognition*. New York, NY: Springer New York, 1998, pp. 5–85.
- [3] Z. Lin, J. He, Z. Zhong, R. Wang, and H.-Y. Shum, "Table detection in online ink notes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1341–1346, Aug 2006.
- [4] Q. Chen, D. Shi, G. Feng, X. Zhao, and B. Luo, "On-line handwritten flowchart recognition based on logical structure and graph grammar," in *5th International Conference on Information Science and Technology (ICIST)*, April 2015, pp. 424–429.
- [5] F. Álvaro, J.-A. Sánchez, and J.-M. Benedi, "An integrated grammar-based approach for mathematical expression recognition," *Pattern Recognition*, vol. 51, pp. 135 – 147, 2016.
- [6] A.-M. Awal, H. Mouchère, and C. Viard-Gaudin, "Improving on-line handwritten mathematical expressions recognition with contextual modeling," in *Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition*, 2010, pp. 427–432.
- [7] F. Álvaro, J.-A. Sanchez, and J.-M. Benedi, "Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars," in *International Conference on Document Analysis and Recognition (ICDAR)*, Sept 2011, pp. 1225–1229.
- [8] S. MacLean and G. Labahn, "A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets," *International Journal on Document Analysis and Recognition*, vol. 16, no. 2, pp. 139–163, 2013.
- [9] M. Celik and B. Yanikoglu, "Probabilistic mathematical formula recognition using a 2D context-free graph grammar," in *International Conference on Document Analysis and Recognition (ICDAR)*, Sept 2011, pp. 161–166.
- [10] J. Rekers and A. Schürr, "Defining and parsing visual languages with layered graph grammars," *Journal of Visual Languages and Computing*, vol. 8, no. 1, pp. 27 – 55, 1997.
- [11] F. Han and S.-C. Zhu, "Bottom-up/top-down image parsing by attribute graph grammar," in *IEEE International Conference on Computer Vision (ICCV)*, vol. 2, Oct 2005, pp. 1778–1785.



- [12] D. Blostein and A. Grbavec, "Recognition of mathematical notation," in *Handbook of Character Recognition and Document Image Analysis*, H. Bunke and P. Wang, Eds. World Scientific, 1997, pp. 557–582.
- [13] K.-F. Chan and D.-Y. Yeung, "Mathematical expression recognition: a survey," *International Journal on Document Analysis and Recognition*, vol. 3, pp. 3–15, 2000.
- [14] R. Zanibbi and D. Blostein, "Recognition and retrieval of mathematical expressions," *International Journal on Document Analysis and Recognition*, vol. 15, no. 4, pp. 331–357, 2012.
- [15] H. Miyao and R. Maruyama, "On-line handwritten flowchart recognition, beautification and editing system," in *International Conference on Frontiers in Handwriting Recognition*, Sept 2012, pp. 83–88.
- [16] C. Carton, A. Lemaitre, and B. Coüasnon, "Fusion of statistical and structural information for flowchart recognition," in *12th International Conference on Document Analysis and Recognition*, Aug 2013, pp. 1210–1214.
- [17] M. Bresler, T. V. Phan, D. Prusa, M. Nakagawa, and V. Hlavác, "Recognition system for on-line sketched diagrams," Sept 2014, pp. 563–568.
- [18] N. E. Matsakis, "Recognition of handwritten mathematical expressions," Master's thesis, Massachusetts Institute of Technology, Cambridge, 1999.
- [19] E. Tapia and R. Rojas, "Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance," in *Graphics Recognition. Recent Advances and Perspectives*, 2004, vol. 3088, pp. 329–340.
- [20] R. Zanibbi, D. Blostein, and J. R. Cordy, "Recognizing mathematical expressions using tree transformation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 24, pp. 1455–1467, 2002.
- [21] F. Álvaro and R. Zanibbi, "A shape-based layout descriptor for classifying spatial relationships in handwritten math," in *Proceedings of the ACM Symposium on Document Engineering*, 2013, pp. 123–126.
- [22] A.-M. Awal, H. Mouchère, and C. Viard-Gaudin, "Towards handwritten mathematical expression recognition," in *Proceedings of the 10th International Conference on Document Analysis and Recognition*, 2009, pp. 1046–1050.
- [23] —, "A global learning approach for an online handwritten mathematical expression recognition system," *Pattern Recognition Letters*, vol. 35, no. 0, pp. 68 – 77, 2012.
- [24] R. Yamamoto, S. Sako, T. Nishimoto, and S. 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.
- [25] F. Simistira, V. Katsouros, and G. Carayannis, "Recognition of on-line handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars," *Pattern Recognition Letters*, vol. 53, pp. 85 – 92, 2015.
- [26] D. H. Younger, "Recognition and parsing of context-free languages in time  $n^3$ ," *Information and Control*, vol. 10, no. 2, pp. 189 – 208, 1967.
- [27] Z. Yuan, H. Pan, and L. Zhang, "A novel pen-based flowchart recognition system for programming teaching," in *Advances in Blended Learning*, E. W. Leung, F. L. Wang, L. Miao, J. Zhao, and J. He, Eds. Springer-Verlag, 2009, pp. 55–64.
- [28] J. Pflatz and R. A., "Web grammars," in *Proc. First International Joint Conference on Artificial Intelligence*, 1969, pp. 193–220.
- [29] D. Grune and J. C. J.H., *Parsing Techniques: A Practical Guide*, 2nd ed. Springer, 2008.
- [30] P. Boullier, A. Nasr, and B. Sagot, "Constructing parse forests that include exactly the N-best PCFG trees," in *Proceedings of the 11th International Conference on Parsing Technologies*, 2009, pp. 117–128.
- [31] A. Delaye and E. Anquetil, "Hbf49 feature set: A first unified baseline for online symbol recognition," *Pattern Recognition*, vol. 46, no. 1, pp. 117–130, Jan. 2013.
- [32] H. Mouchère, C. Viard-Gaudin, R. Zanibbi, and U. Garain, "Competition on recognition of on-line handwritten mathematical expressions (CROHME 2014)," in *14th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, Sept 2014, pp. 791–796.
- [33] F. Julca-Aguilar, H. Mouchère, C. Viard-Gaudin, and N. S. T. Hirata, *Progress in Pattern Recognition, Image Analysis, Computer Vision, and Applications: 20th Iberoamerican Congress*. Cham: Springer International Publishing, 2015, ch. Top-Down Online Handwritten Mathematical Expression Parsing with Graph Grammar, pp. 444–451.
- [34] F. Julca-Aguilar, C. Viard-Gaudin, H. Mouchère, S. Medjkoune, and N. Hirata, "Mathematical symbol hypothesis recognition with rejection option," in *14th International Conference on Frontiers in Handwriting Recognition*, 2014.
- [35] F. Julca-Aguilar, N. S. T. Hirata, H. Mouchère, and C. Viard-Gaudin, "Subexpression and dominant symbol histograms for spatial relation classification in mathematical expressions," in *23rd International Conference on Pattern Recognition (ICPR)*, Dec 2016, pp. 3446–3451.
- [36] S. Belongie, J. Malik, and J. Puzicha, "Shape matching and object recognition using shape contexts," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, pp. 509–522, April 2002.
- [37] A.-M. Awal, G. Feng, H. Mouchère, and C. Viard-Gaudin, "First experiments on a new online handwritten flowchart database," in *Document Recognition and Retrieval XVIII*, San Francisco, United States, Jan 2011, pp. 7874 – 78740A.
- [38] M. Bresler, D. Prusa, and V. Hlavác, "Modeling flowchart structure recognition as a max-sum problem," in *12th International Conference on Document Analysis and Recognition*, Aug 2013, pp. 1215–1219.
- [39] A. Lemaitre, H. Mouchère, J. Camillerapp, and B. Coüasnon, *Interest of Syntactic Knowledge for On-Line Flowchart Recognition*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 89–98.
- [40] C. Wang, H. Mouchère, C. Viard-Gaudin, and L. Jin, "Combined segmentation and recognition of on-line handwritten diagrams with high order markov random field," in *15th International Conference on Frontiers in Handwriting Recognition (ICFHR)*, 2016, pp. 252–257.

**Frank Julca-Aguilar** is a postdoctoral researcher at University of São Paulo (Brazil). He received his B.Sc degree in Computer Science from National University of Trujillo (Peru), and his PhD degree in Computer Science from University of Nantes (France) and University of São Paulo. His research interests include machine learning and graph-based methods applied to computer vision, handwritten recognition, and image processing.

**Harold Mouchère** received his Ph.D. degree in Computer Science from INSA in Rennes, France, in 2007 and is now Associate Professor at University of Nantes, France. After four years at the IRISA laboratory, he integrated in 2008 the IRCCyN laboratory which became in 2017 the LS2N (*Laboratoire des Sciences du Numérique de Nantes*). His research concerns Pattern Recognition and Machine Learning with application to structured document analysis (handwritten mathematical expression, on-line flowchart and ancient document analysis). Since 2011, he is in the organization committee of CROHME competitions.

**Christian Viard-Gaudin** is a Full Professor at the Electrical and Electronic Engineering Department of University of Nantes. He is a leading researcher in the field of document image processing and handwriting recognition. He has been involved in many projects concerning automatic mail sorting systems, offline and online handwriting recognition. Currently, he is working on mathematical expression recognition, writer identification and document categorization.

**Nina S. T. Hirata** holds a PhD degree in Computer Science. She is currently an associate professor at the Department of Computer Science, Institute of Mathematics and Statistics of University of São Paulo. Her research interests include pattern recognition and image/signal analysis, using approaches based on machine learning, graphs, mathematical morphology, and other tools, with applications in a variety of problems such as document image analysis, graphics recognition, astronomical and plankton image classification, image segmentation, object detection in images, among others.