International Journal on Document Analysis and Recognition (IJDAR) (2020) 23:143–160
https://doi.org/10.1007/s10032-019-00349-6

ORIGINAL PAPER

A general framework for the recognition of online handwritten graphics

Frank Julca-Aguilar[1] · Harold Mouchère[2] · Christian Viard-Gaudin[2] · Nina S. T. Hirata[1]

Received: 4 July 2018 / Revised: 7 September 2019 / Accepted: 6 December 2019 / Published online: 3 January 2020

Abstract
We revisit graph grammar and graph parsing as tools for recognizing graphics. A top-down approach for parsing families of handwritten graphics containing different kinds of symbols and of structural relations is proposed. It has been tested on two distinct domains, namely the recognition of handwritten mathematical expressions and of handwritten flowcharts. In the proposed approach, a graphic is considered as a labeled graph generated by a graph grammar. The recognition problem is translated into a graph parsing problem: Given a set of strokes (input data), a parse tree which represents the best interpretation is extracted. The graph parsing algorithm generates multiple interpretations (consistent with the grammar) that can be ranked according to a global cost function that takes into account the likelihood of symbols and structures. The parsing algorithm consists in recursively partitioning the stroke set according to rules defined in the graph grammar. To constrain the number of partitions to be evaluated, we propose the use of a hypothesis graph, built from data-driven machine learning techniques, to encode the most likely symbol and relation hypotheses. Within this approach, it is easy to relax the stroke ordering constraint allowing interspersed symbols, as opposed to some previous works. Experiments show that our method obtains accuracy comparable to methods specifically developed to recognize domain-dependent data.

Keywords Graphics recognition · Online handwriting recognition · Graph parsing · Graph grammar · 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 consecutive with respect to neither time nor spatial order. Furthermore, a diversified set of relations is possible between arbitrary pairs of symbols. See Fig. 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 in texts is relatively simpler than in graphics as already mentioned. Identification of segmented symbols includes challenges

✉ Frank Julca-Aguilar
faguilar@ime.usp.br

Harold Mouchère
harold.mouchere@univ-nantes.fr

Christian Viard-Gaudin
christian.viard-gaudin@univ-nantes.fr

Nina S. T. Hirata
nina@ime.usp.br

[1] Department of Computer Science, Institute of Mathematics and Statistics, University of São Paulo, São Paulo, Brazil

[2] Laboratoire des Sciences du Numérique de Nantes (LS2N), University of Nantes, Nantes, France

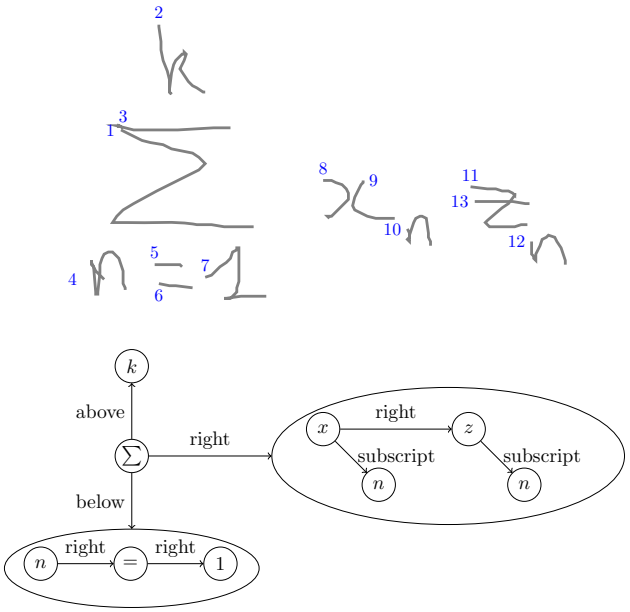


Fig. 1 Handwritten mathematical expression example. Top: a sequence of strokes where the order (indicated by numbers in blue) is given by the input time. Symbols $\sum$ and z are composed of non-consecutive strokes. Bottom: The expression is composed of symbols and several types of spatial relations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

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 toward any orientation). Structural analysis involves the identification of relations between symbols and a coherent integrated interpretation. The large variety of relations might define complex hierarchical structures that increment 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 [3–5].

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 [6–8], 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 $\sum^{n}_{i=1} x_i$), 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. We 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) hypothesis identification, (2) graph parsing, and (3) optimal interpretation retrieval. The first stage computes a graph, called hypothesis 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 predefined graph grammar, using the hypothesis 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 hypothesis 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 then the most likely structures in the input stroke set are captured in the hypothesis 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 hypothesis 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 have already proven adequate to model graphics in general and by previous examples that show specification of families of graphics by means of graph grammars [5,10,11]. In addition, creation of hypothesis graphs is greatly facilitated by modern 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 hypothesis 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 Sect. 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 Sect. 3, we detail the proposed framework. Then, in Sect. 4, we describe how the elements and parameters required by the framework have been defined for the recognition of mathematical expressions and flowcharts. In Sect. 5, we present and discuss the experimental results for both applications, and in Sect. 6 the conclusions and future works are discussed.

2 Related works

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

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 [18–20]. 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 closing and the opening parentheses, respectively. To solve this type of ambiguity, it may be necessary to examine the 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 [21–25] use modifications of context-free string grammars in Chomsky normal form[1] (CNF). Such grammars define a set of 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 $TERM \xrightarrow{superscript} NUMBER\ NUMBER$. 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: $A \xrightarrow{r} A_1 A_2 \ldots A_k$, 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.

Graph grammars offer a more powerful representativeness than string grammars. Although several graph grammar-based methods for graphics recognition have been proposed [9,26–30], their applications have been limited to specific languages, or assume constraints that make their generalization difficult. One of the first of such methods, proposed by Bunke et al. [26], aims at the recognition of circuit diagrams and flowcharts. In addition to a graph grammar that defines the structure and semantics of a graphic, the authors introduce a *control diagram* that defines how the production rules of a graph grammar should be applied to generate a graphic. There is then the need of defining a new *control diagram* for each type of graphic. Lavirotte et al. [30] proposed graph grammars to parse printed mathematical expressions. The approach introduces several spatial assumptions that are

[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

valid for printed expressions, but that might be easily violated in handwritten data. To recognize handwritten mathematical expressions, Celik and Yanikoglu [9] propose a method based on 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. 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. In [11], the authors propose an attributed graph grammar that allows 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.

Most grammar-based algorithms proposed in the literature for the recognition of mathematical expressions are based on the CYK algorithm [31]. 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 [21–23,25]. 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 [21–25], the rules would have at most two components in the RHS and therefore the number of meaningful partitions would be $O(n)$—the first i strokes being assigned to the first component and the rest to the second, with $i \in \{1, \ldots, n-1\}$. On the other hand, if we do not impose CNF, but keep the stroke ordering assumption as in [8], then a rule could have k symbols in its RHS, and the number of meaningful partitions would be $O\binom{n}{k}$, corresponding to $k-1$ split points on the sequence of n strokes. In graph grammars, without any restriction and assuming a rule with k vertices in the RHS, the number of partitions would be $O(n^k)$—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], helps to manage the parsing complexity. Note, however, that in this case the set of recognizable expressions would be 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 may vary greatly. For instance, the flowchart shown in Fig. 2 includes two loops, and adjacent symbols can be located at any (vertical, horizontal, or diagonal) position relative to each other, regardless of 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 [32], 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) 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 the last step. After symbol candidates are identified, the best symbol combination is selected through a max-sum optimization process.

Some more recent methods [33–38] apply deep learning models based on encoder–decoder networks along with attention techniques. These methods are able to process offline representations of the expressions (images) [33,34,37] as well as raw strokes, and combination of both data [36,38]. We note that these end-to-end approaches rely on a linear description of the recognition result (e.g., typesetting in LaTeX, which is not a simple task for graphics in general) or may require a large amount of training data. Thus, extension of the methods to other types of graphics can be challenging.

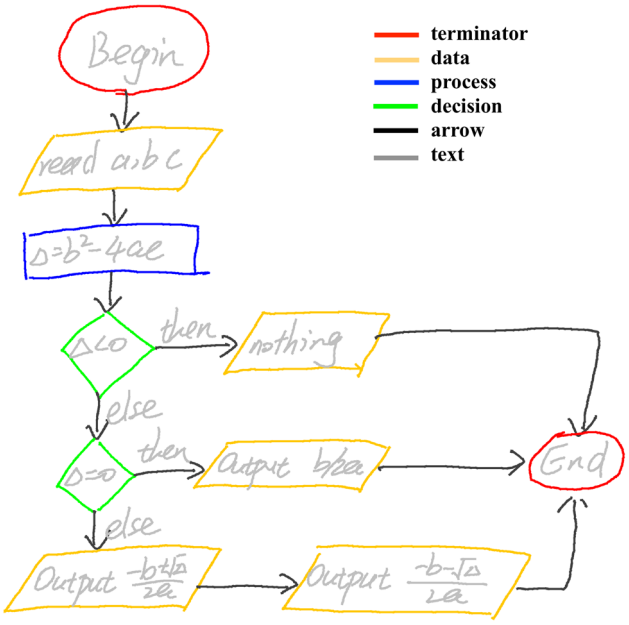


Fig. 2 Flowchart example. Strokes are colored according to the symbol type they belong to

3 The proposed recognition framework

In the method proposed in this work, instead of 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 in [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 hypothesis graph (see Sect. 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 in the hypothesis graph building. It is expected that this characteristic will make the method general enough to be applied to the recognition of a variety of graphic notations.

As illustrated in Fig. 3, the proposed recognition framework is composed of three main parts: (1) **hypothesis 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 hypothesis graph. In the second part, valid interpretations (potentially multiple of them) are built from the hypothesis 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 Sect. 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 [39]. These concepts are illustrated in Fig. 4a, b.

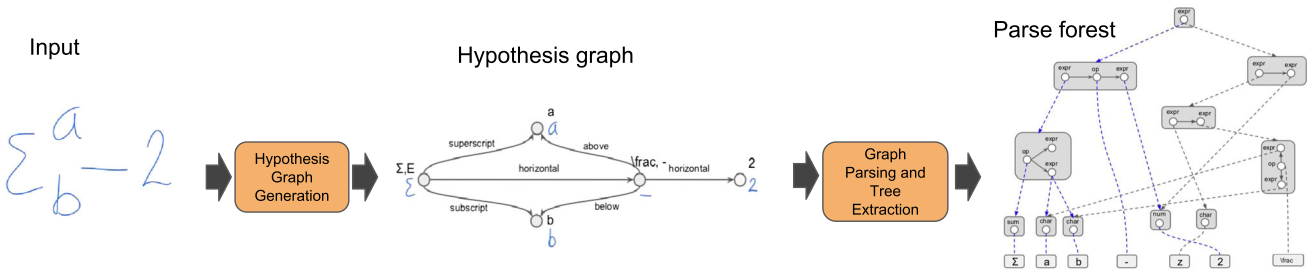


Fig. 3 Our proposed method consists of three steps: hypothesis graph generation, graph parsing, and optimal tree extraction. In the first step, the input strokes are grouped into subsets that have high probability of forming symbols and the relations between them are also identified. Both symbols and relations are stored in a hypothesis graph. The second step evaluates the most probable combinations of the symbols and relations and builds a parse forest that stores multiple interpretations of the input. This step uses a graph grammar to validate the interpretations. In the third step, the best interpretation is extracted considering a scoring function that considers the whole structure of the expressions

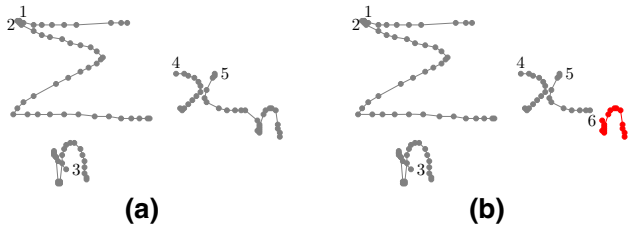


Fig. 4 Handwritten expressions representing $\sum_n x_n$. Each expression is composed of a set of strokes, where each stroke is a sequence of bidimensional coordinates (dots in gray). In **a**, stroke 5 belongs to two symbols. In **b**, each stroke belongs to only one symbol

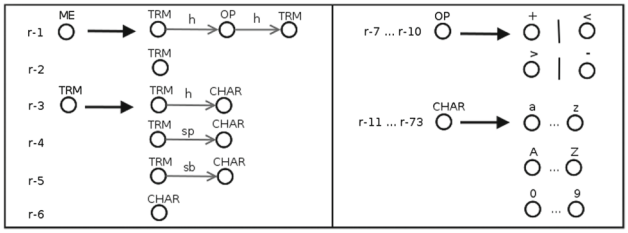


Fig. 5 Graph grammar that models basic mathematical expressions. The grammar is defined by non-terminals $N = \{ME, TRM, OP, CHAR\}$, relation labels $RL = \{sp, sb, h\}$, terminals $T = \{+, -, <, >, a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$, rules $R = \{r-1, \ldots, r-73\}$, and ME at the left-hand 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

3.2 Graph grammar model

A graph grammar [40] 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 [40]. The language defined by a graphic grammar M 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 5 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).

Figure 6 shows a graph generation process using the grammar of Fig. 5. 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* [40], 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 $G \backslash G_l$. The *attachment* may be defined by a set of edges that link the replacing graph G_r to $G \backslash G_l$. For instance, Fig. 7 shows two different embeddings for a same production rule and the graphs generated for 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 Sect. 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 Sect. 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 Hypothesis graph generation

Given a set of strokes S, we define a hypothesis graph as an attributed graph $H = (V_H, E_H)$, where V_H is a set of symbol hypothesis and E_H is a set of relation hypothesis 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 $L(v) = \{(l_i, s_i), i = 1, \ldots, k_v\}$ 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., hypoth-

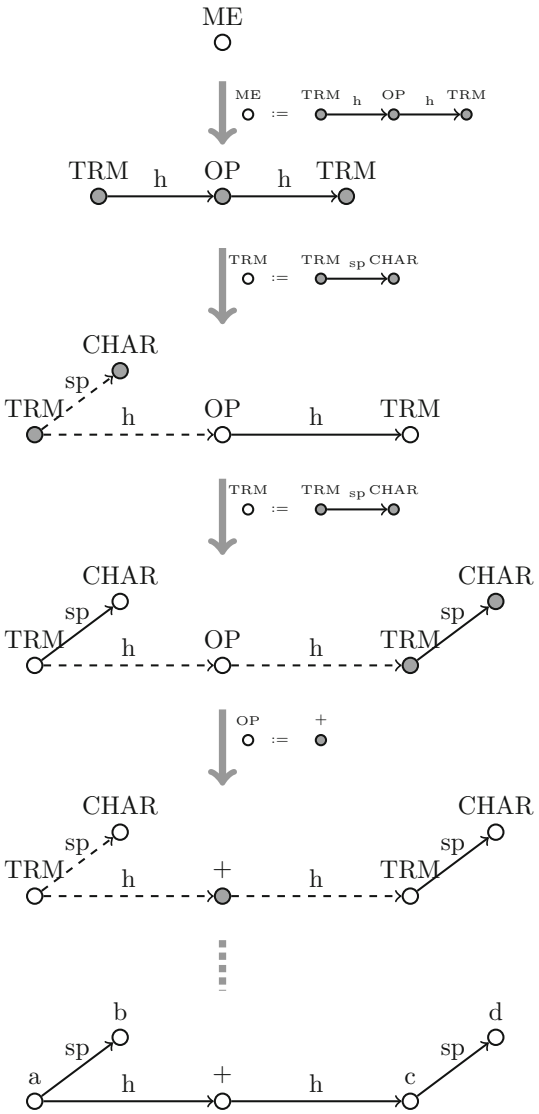


Fig. 6 Generation of a graph that represents the expression $a^b + c^d$. At each rule application, the replacing graph nodes are depicted in dark gray. Edges that link the replacing graph with the host graph are depicted with dashed arrows. Rule applications after the fourth one are not shown

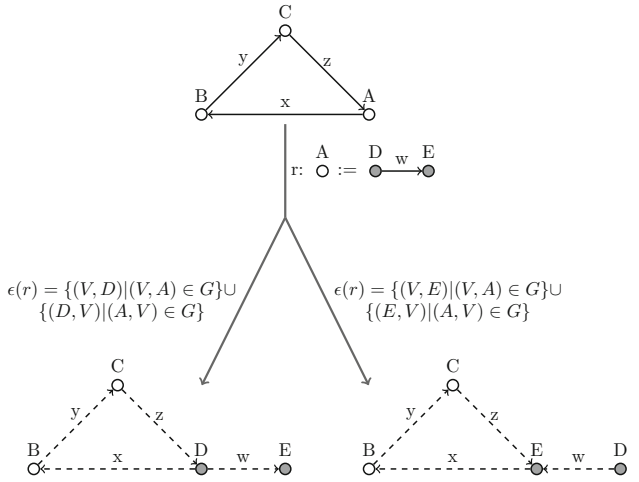


Fig. 7 Graph transformation with two different embeddings. The top graph is transformed through rule r. Each embedding defines edges between vertices that are linked to vertex A of the top graph with vertex D (left-hand-side embedding) or E (right-hand-side embedding) of the replacing graph. Dashed arrows represent the edges defined by each embedding

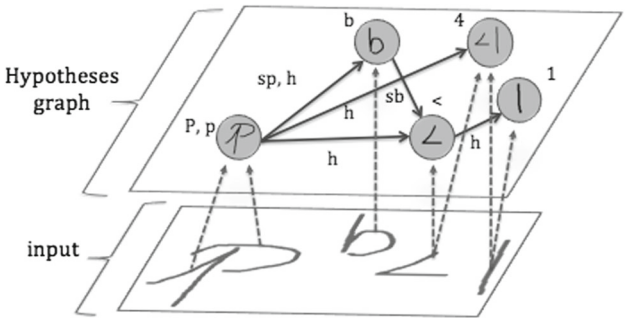


Fig. 8 Hypothesis graph example. Vertices represent the symbol hypothesis and edges represent the relations between symbols. The labels associated with symbols and relations indicate their most likely interpretations

esis 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 8 shows a handwritten mathematical expression and a hypothesis graph calculated from it.

To build a hypothesis graph, machine learning methods are effective in identifying groups of strokes that may form symbols and, similarly, relations among them. (See application example in Sect. 4). Since many stroke groups do not correspond to an actual symbol and many pairs of symbols are not directly related to each other within a graphic, rather than training classifiers to identify only true hypothesis, 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 hypothesis. As will become clear later, hypothesis graphs play an important role to constrain the search space during the parsing process. A high precision and recall in the identification of true symbols and relations are thus desirable to efficiently constrain the search space, without losing components that are part of the correct interpretation.

3.3.1 Label list pruning

To define the labels and respective likelihood scores of symbol and relation hypothesis, we could use the confidence 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 $\{(l_i, s_i), i = 1, \ldots, n_h\}$ 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 an 1), we define the minimum number of k top ranked labels whose confidences sum up to at least tr:

$$k = \arg\min_{x} \sum_{i=1}^{x} s_i > tr \tag{1}$$

Hypothesis h is rejected if it presents the highest score for the $junk$ class label with score above the threshold tr. Otherwise, we set $L(h) = \{(l_i, s_i) : i = 1, \ldots, k\}$. We define label pruning thresholds t_{symb} for symbols and t_{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_{input}, according to a grammar. Since there might be more than one interpretation, multiple trees might be generated, possibly sharing subtrees with each other. Thus, they will be stored in a parse forest.

Figure 9 shows a parse forest calculated from the hypothesis graph of Fig. 8, using the graph grammar of Fig. 5. 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 with 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 A, for each rule $A := B$ associated with A, 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 is 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.

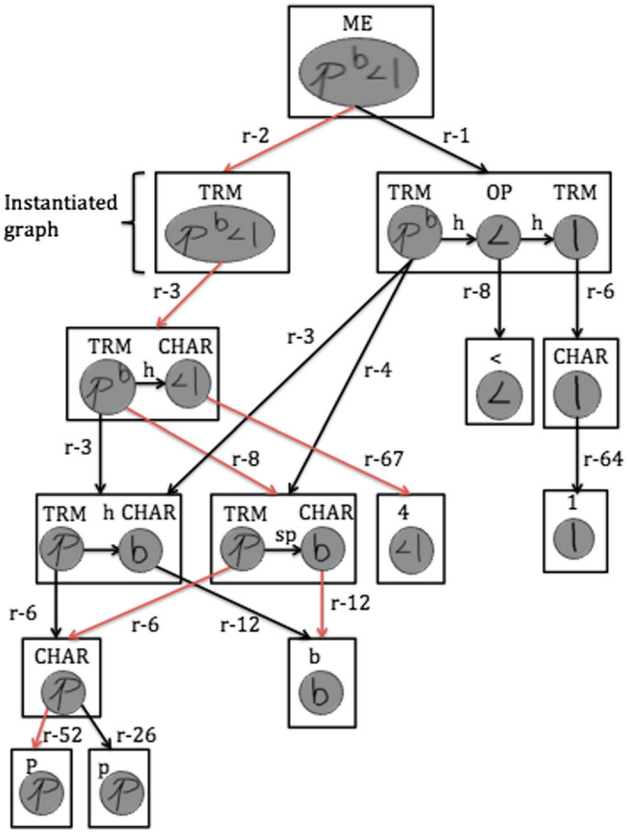


Fig. 9 A parse forest representing multiple interpretations of a mathematical expression. Labels on arrows refer to grammar rules of Fig. 5. Red arrows represent a parse tree that corresponds to the interpretation "P^b4"

A main strategy of our method is to constrain the number of partitions to be examined with the aid of the hypothesis graph, denoted below as H. We assume that all meaningful interpretations are present in the hypothesis graph as a subgraph. Thus, before starting the parsing process, we build the set of all stroke groups, denoted hereafter as STK, that underlie any valid connected subgraph of H. Note that these stroke groups must not contain repeated strokes. Furthermore, not all stroke groups will be necessarily parsable. We also record in STK the relation between two stroke groups 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 Fig. 5. 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 Fig. 5. 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 : parseGraphic(S,NT)
Parses a set of strokes S from a non-terminal NT
Input: (S, NT)
Output: $parsedG = \{(G_1, r_1), \ldots, (G_q, r_q)\}$
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** $parsedG$

Algorithm 1 receives as inputs a stroke set $S = \{stk_1, \ldots, stk_n\}$ 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.

To avoid recomputation, a global table TBL indexed by pairs $(S = \{stk_1, \ldots, stk_n\}, NT)$ is used. An entry in TBL is of the form:

$$TBL[(S, NT)] = \{(G_1, r_1), \ldots, (G_q, r_q)\},$$

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 hypothesis 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 size and minimum size of non-terminal nodes are a strategy that has been previously used in text parsing [41]. 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 strategy is to explore the information regarding terminals that can be generated by a non-terminal. For instance, in the grammar of Fig. 5, non-terminal OP can generate only symbols $+$, $-$, $<$, or $>$. Thus, stroke subsets that do not contain any hypothesis with one of these labels as terminals do not need to be evaluated during the parsing process. Analogously, stroke groups that correspond to symbol and relation hypothesis with mean junk score (computed over all labels assigned to any of its subgroups) above a certain threshold t_{junk} can be disregarded. This pruning is mainly useful when the symbol and relation hypothesis 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 $e \in E_x$ there 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 $J_s(x)$:

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

and a relation cost $J_r(x)$:

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

Then, the cost of t_x is defined as

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

where n_s and n_r are, respectively, the number of symbols and relations under t_x. Parameter α 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 Fig. 10. 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:

$$\begin{aligned} J(t_{x_1}) = & \frac{\alpha}{4}\Big(J_s(v_8) + J_s(v_9) + J_s(v_{10}) + J_s(v_{12})\Big) \\ & + \left(\frac{1-\alpha}{3}\right)\Big(J_r(e_1) + J_r(e_2) + J_r(e_3)\Big). \end{aligned} \tag{5}$$

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. [42]. 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.

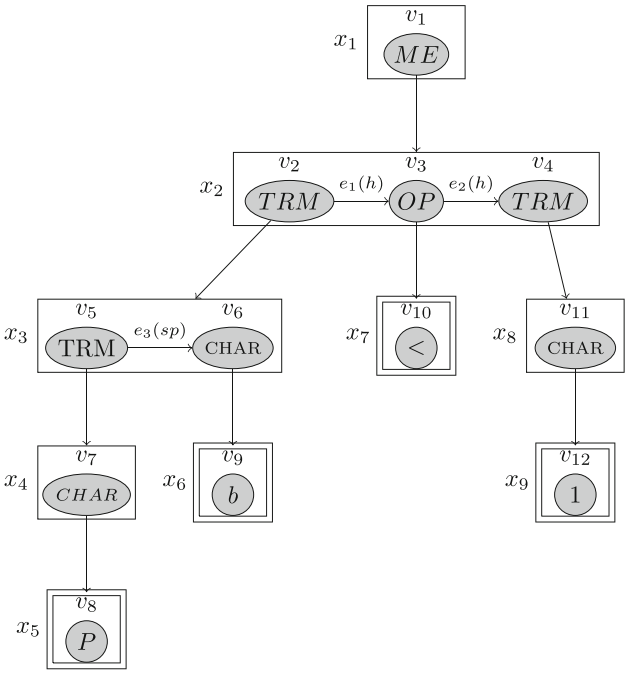


Fig. 10 Parse tree of expression $P^b < 1$, extracted from the parse forest of Fig. 9. Nodes are indexed as x_i, $i = 1, \ldots, 9$. Similarly, vertices and edges of the instantiated graphs are, respectively, indexed as v_j, for $j = 1, \ldots, 12$, and e_k, for $k = 1, \ldots, 3$. Nodes with terminal symbols are depicted with double line borders

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. 9, 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 $minJ(x)$ 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

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

This strategy resembles the one proposed in [42], 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 the parse trees are kept. The pruning threshold t_{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 hypothesis 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 `xml` format are available at 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 [43]. 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-2016 dataset [44]. It consists of handwritten expressions divided into training, validation, and test sets, with 8836, 986, and 1147 expressions, respectively. From here onwards, we refer to the training plus the validation sets as the training set. The expressions include 101 symbol classes, and six relation classes (*horizontal* as in "ab", *above* as in "$\overset{x}{\sum}$", *below* as in "$\underset{x}{\sum}$", *superscript* as in "a^b", *subscript* as in "a_b", and *inside* as in "$\sqrt{x}$"). To define the (infinite) set of acceptable expressions, the CROHME-2016 competition provides a grammar using the latex string representations. 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 with the application of rule r on G replaces u with G_r, generating an updated graph G', such that $V_{G'} = V_G \backslash \{u\} \cup V_{G_r}$ and $E_{G'} = [E_G \backslash (\{(u', u) : u' \in V_G\} \cup \{(u, u') : u' \in V_G\})] \cup E^+$ where

$$E^+ = \{(u', v') : (u', u) \in E_G\} \cup \{(v'', u') : (u, u') \in E_G\}. \tag{7}$$

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 Hypothesis graph building

To generate the hypothesis graph, we used ideas similar to the ones used in the symbol segmentation and classification methods described in [45,46] and the spatial relation classification methods described in [47]. These methods use multilayer neural networks with softmax output, having images of the symbols and surrounding region rather than stroke-related features as inputs. Here, we just replace standard multilayer neural networks with simple LeNet-5 [48] like convolutional neural networks (CNN). The network outputs are 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 hypothesis graph is the symbol and relation label pruning thresholds, t_{symb} and t_{rel}. (See Eq. 1). These threshold values determine how many

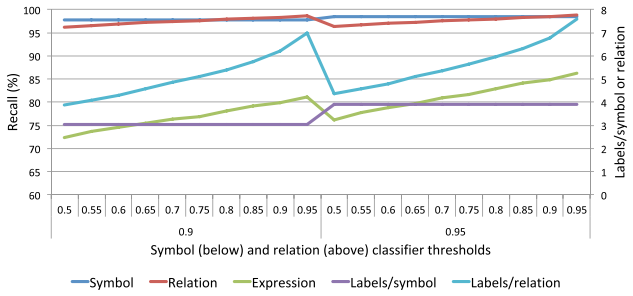


Fig. 11 Symbol, relation, and expression level recall (left axis) and the number of labels of the graph relative to the number of actual symbols and relations (right axis) generated during the hypothesis graph generation step. The relation classification threshold t_{rel} is varied in the range [0.5–1.0], for each symbol classification threshold $t_{\text{symb}} \in \{0.9, 0.95\}$

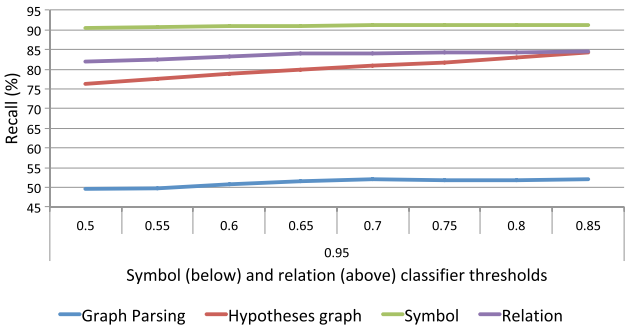


Fig. 12 Symbol, relation, and expression recalls obtained from graph parsing and hypothesis recall potential expressed in the hypothesis graph, for symbol threshold $t_{\text{symb}} = 0.95$ and different relation thresholds t_{rel}

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 hypothesis graph, the achievable maximum recognition rates are bounded by possibilities encoded in the hypothesis graph.

From the training set, we randomly selected 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_{symb} and t_{rel} on the validation set. Two types of measures have been computed for each threshold value. First, we computed the symbol, relation, and complete expression recalls, which indicate how many of the components of the expressions are in fact represented in the hypothesis graph. Second, we computed the ratio between the number of symbol hypothesis labels and actual number of symbols as well as between the relation hypothesis labels and the number of actual relations per expression. If this ratio is one (the ideal case), it means that there are only one label attached for each symbol.

The evaluation results are shown in Fig. 11, for $t_{\text{symb}} \in \{0.9, 0.95\}$ and t_{rel} in the range $[0.5 - 1]$. The left-hand axis shows the recall, and the right-hand axis shows the ratios. Note that this evaluation is concerned with verifying how many of the elements of interest are, in fact, present in the hypothesis graph; it is not related to parsing.

As can be seen, with respect to symbols and relations, even for the lowest threshold values ($t_{\text{symb}} = 0.9$ and $t_{\text{rel}} = 0.5$) the recall is above 95%. However, for complete expressions (meaning all symbols and relations of the expressions are in the hypothesis graph), the recall for the lowest threshold values is 72%. Additionally, while there is no large improvement in symbol and relation recalls as we increase these thresholds, expression recall presents large improvements. The ratios, on the other hand, indicate high precision of symbol and relation classification (given that the potential number of symbol and relation hypothesis is exponential to the number of input strokes).

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 0.95 and for t_{rel} to 0.85, as they provide a good trade-off between recognition accuracy and time. [2] In this evaluation, for optimal tree extraction we set $\alpha = 0.5$ (same weight for the symbol and relation costs; see Eq. 4) and $t_{\text{pr}} = 0.1$ (tree pruning threshold; see Eq. 6). Figure 12 shows the symbol, relation, and expression recall obtained by the parsing method. It also shows (in red) the expression recall potential (maximum achievable recall) expressed in the hypothesis graph. The parsing recall is almost constant, regardless of the value of t_{rel}, while the gap between the potential recall and the parsing recall increases up to about 32%. Based on this evaluation, we chose $t_{\text{symb}} = 0.95$ and $t_{\text{rel}} = 0.85$, as these values allow to keep more hypothesis and can be useful during parsing of unseen expressions (better generalization).

Using $t_{\text{symb}} = 0.95$ and $t_{\text{rel}} = 0.85$, we have also evaluated on the validation set the effect of different values of t_{pr} (tree pruning threshold) and α (weighting in the cost function) in the tree extraction step. Through this evaluation, we set $t_{\text{pr}} = 0.1$ and $\alpha = 0.4$. (This choice was based on the best expression recall).

4.2 Recognition of flowcharts

4.2.1 Dataset and grammar

We use the flowchart dataset described in [49]. The dataset includes seven symbol classes (*arrow*, *connection*, *data*, *decision*, *process*, *terminator*, and *text*) and three relation classes (*Src*, *Targ*, and *AssTxt*). An example is presented in Sect. 2 (Fig. 2), with strokes colored according to the symbol

[2] Processing the whole validation set in up to 3 h (mean recognition time of 10 s).

type they belong to. In this dataset, relations in each flowchart are established between "adjacent" symbols. For instance, in the flowchart of Fig. 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 9000.

As described in Sect. 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 text 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-2016 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:

$$E^+ = \{(u', v) \mid (u', u) \in E_G \text{and} v = \arg\min_{v_i} cost_r(u', v_i)\} \cup \{(v, u') \mid (u, u') \in E_G \text{and} v = \arg\min_{v_i} cost_r(v_i, u')\},$$

where $cost_r(u, v)$ 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_{symb} and t_{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 [47], 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 hypothesis, with $t_{\text{junk}} = 0.25$ (see Sect. 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).

Table 1 Structure recognition rates over the test set of CROHME-2016 competition

System	Structure	Structure + labels		
		≤ 0	≤ 1	≤ 2
MyScript	88.14	67.65	75.59	79.86
Wiris	74.28	49.61	60.42	64.69
Tokyo	61.55	43.94	50.91	53.70
São Paulo	57.02	33.39	43.50	49.17
Nantes	21.45	13.34	21.02	28.33
Ours	64.78	40.80	52.39	58.15

5 Results and discussion

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.

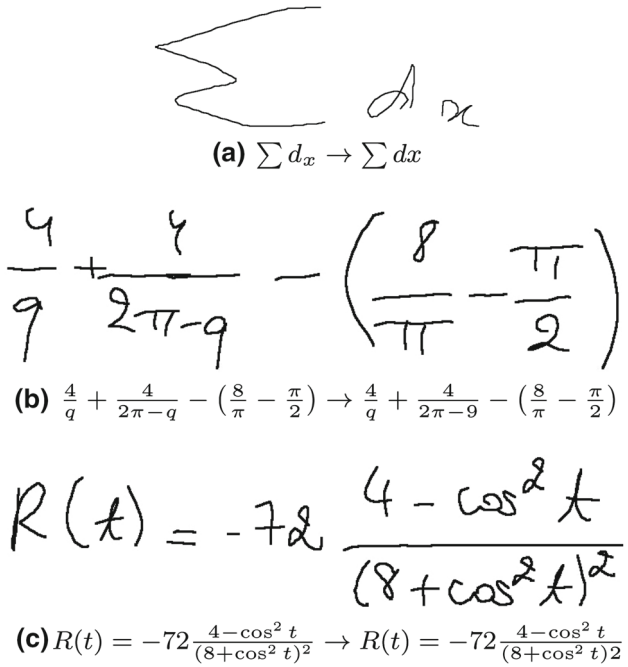
5.1 Recognition of mathematical expressions

Table 1 shows structure recognition rates including those reported in the CROHME-2016 competition [44]. The competing systems are identified as reported in the competition results. Note that the system identified as "São Paulo" is also ours, however, using standard multilayer networks as classifiers. The error columns indicate recognition rates considering recognition with up to 0, 1, and 2 incorrect labels for stroke and directed stroke edges.

The two best systems, *MyScript* and *Wiris*, include statistical models built using 592, 000 expressions from Wikipedia [44]. In particular, *MyScript* is a commercial system[3] that has been optimized over hundreds of thousands of handwritten equations that are not publicly available. The statistical information used by both systems could explain, at some extent, their better performance.

Our method recognized 40.80% of the expressions completely (correct identification of labels and structures). We

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

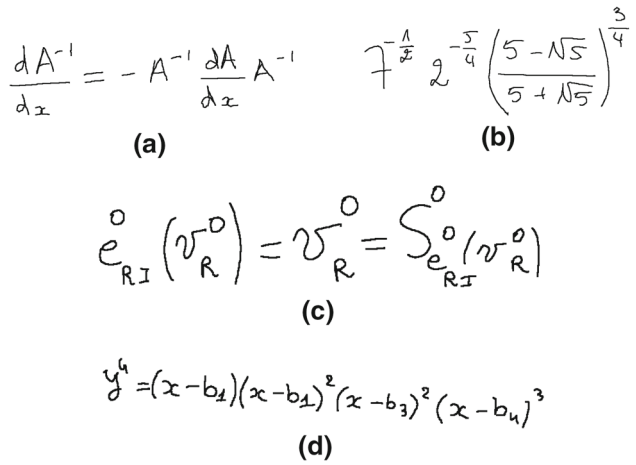


(a) $\sum d_x \rightarrow \sum dx$

(b) $\frac{4}{q} + \frac{4}{2\pi - q} - \left(\frac{8}{\pi} - \frac{\pi}{2}\right) \rightarrow \frac{4}{q} + \frac{4}{2\pi - 9} - \left(\frac{8}{\pi} - \frac{\pi}{2}\right)$

(c) $R(t) = -72\frac{4-\cos^2 t}{(8+\cos^2 t)^2} \rightarrow R(t) = -72\frac{4-\cos^2 t}{(8+\cos^2 t)2}$

Fig. 13 Expressions recognized with one or two errors. For each expression, its ground truth and the system's output are shown as: ground truth → system's output



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

Fig. 14 Examples of expressions correctly recognized by our system

have noticed, however, that the complete expressions were present in 79.69% of the hypothesis graphs. 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.

We also note that the difference between the percentage of correctly recognized expressions and those that were not correctly recognized due to up to 2 errors is 17.35%. This difference is the highest among all the systems, which indicates that our system has the largest margin for improvement by correcting those errors. Figure 13 shows some of the expressions that fall in this case.

Our method is able to correctly recognize several ambiguous symbols as well as relations. Figure 14 shows examples of those challenging cases correctly recognized by our system.

Table 2 shows the comparison of the recall and precision rates of the symbol segmentation, classification, and spatial relations of our system and systems of the CROHME-2016 competition. We can see that while our system obtains high recall and precision at symbol level, it obtains low rates at relations level. Furthermore, classification recall is close to the one obtained on the validation set (91%). Thus, this is an indication that significative improvement is possible by improving the relation classifier.

Figure 15 shows the mean recognition time of our implementation over expressions grouped by their number of strokes. We can see that our system can recognize expressions with up to thirty strokes within a few seconds. Above that number of strokes, our system shows a high increase in terms of recognition time. However, we can also adjust the system to have a faster recognition response by lowering the symbol and relation classifier thresholds. For instance, by setting the symbol and relation classification thresholds to $t_{symb} = 0.9$ and $t_{rel} = 0.7$, we only obtain a expression recognition rate 3% lower than our best system, but with a much faster response, as illustrated in Fig. 15.

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. [50] 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 Fig. 16. 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 large part of the flowchart in the right side of Fig. 16a, or very short ones, or yet curvy ones, are correctly identified.

When a true symbol or a true relation is not in the hypothesis graph, the parsing process will fail to recognize the graphic. Figure 17 shows an example of spatial relation and another of a symbol that were not recognized during the hypothesis graph generation.

However, since 67% of the flowcharts were fully represented in the hypothesis 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 hypothesis graph presented lower likelihood scores than the false ones, lead-

Table 2 Symbol and spatial relation detection comparison of our method with systems of the CROHME-2016 competition

System	Segmentation		Classification		Relations	
	Recall	Precision	Recall	Precision	Recall	Precision
MyScript	98.89	98.95	95.47	95.53	95.11	95.11
Wiris	96.49	97.09	90.75	91.31	90.17	90.79
Tokyo	91.62	93.25	86.05	87.58	82.11	83.64
São Paulo	92.91	95.01	86.31	86.26	81.48	84.16
Nantes	94.45	89.29	87.19	82.42	73.20	68.72
Ours	95.32	96.84	89.39	90.81	78.68	79.93

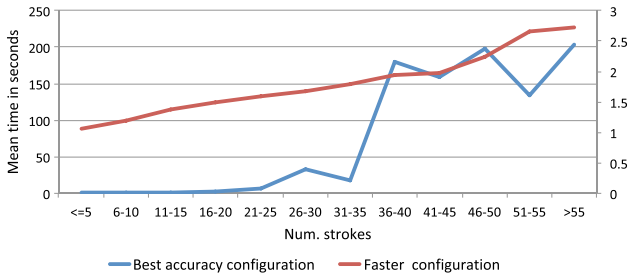


Fig. 15 Mean recognition time of our system with two configurations: the configuration used for obtaining our best results in terms of expression recognition rate (left axis) and an alternative faster configuration with $t_{symb} = 0.9$ and $t_{rel} = 0.7$ (right axis)

Table 3 Flowchart recognition: comparison of our method and four state-of-the-art methods, w.r.t. stroke (str. lab.) and symbol labeling (sym. lab.) accuracy (%)

System	Str. lab.	Sym. lab.
Include text recognition		
Lemaitre et al. [51]	91.1	72.4
Carton et al. [16]	92.4	75.0
Bresler et al. [17]	95.2	82.8
Wang et al. [52]	95.8	84.3
Without text recognition		
Bresler et al. [50]	–	74.3
Ours	91.1	85.5

ing 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 hypothesis graph generation step. By improving symbol and relation classifiers, a considerable improvement should be possible.

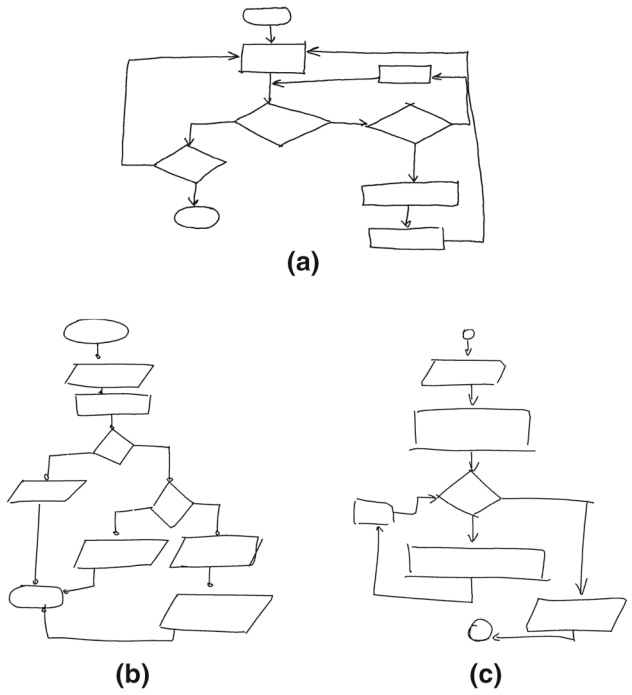


Fig. 16 Examples of flowcharts that have been correctly recognized by our method

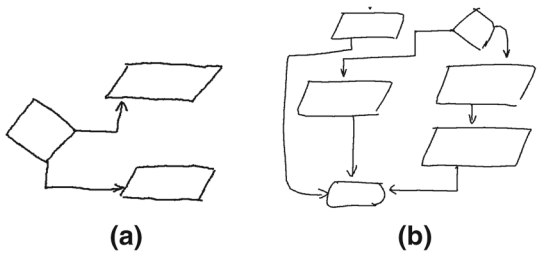


Fig. 17 Parts of flowcharts with missing components in the hypothesis graph. **a** Relation between the top arrow and data symbols has not been identified; **b** the top-center arrow has not been identified

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 hypothesis graph that encodes symbol and relation hypothesis 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 hypothesis 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 hypothesis 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 is on a par with several 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 79% of the test expressions were fully represented in their corresponding hypothesis graph, only 40% of the expressions were fully recognized, corresponding to a gap of almost 39%. 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 hypothesis 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 in symbol and hypothesis likelihood scores will naturally lead to better cost estimation. A second improvement could be possible by incorporating in the cost computation a term that captures statistical information with respect to structure occurrences.

Another important feature of our framework is the possibility of managing computational cost. Hypothesis 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 hypothesis graph are considered. In addition, there is a set of parameters to control the amount of possibilities to be encoded in the hypothesis 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.

Although deep learning models have proven very powerful to perform an end-to-end recognition process, understanding how recognition happens is still a big challenge. An intermediary approach would be to use deep neural networks in specific parts of our method, such as in symbol and relation classification or in data-driven generation of grammar rules. We believe this has potential to improve the recognition rates, without weakening understandability. We would like also to extend the applications to other families of graphics or 2D structures.

Acknowledgements 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. Plamondon, R., Srihari, S.N.: On-line and off-line handwriting recognition: a comprehensive survey. IEEE Trans. Pattern Anal. Mach. Intell. **22**, 63–84 (2000)
2. Marriott, K., Meyer, B., Wittenburg, K.B.: A Survey of Visual Language Specification and Recognition, pp. 5–85. Springer, New York (1998)
3. Lin, Z., He, J., Zhong, Z., Wang, R., Shum, H.Y.: Table detection in online ink notes. IEEE Trans. Pattern Anal. Mach. Intell. **28**(8), 1341–1346 (2006)
4. Chen, Q., Shi, D., Feng, G., Zhao, X., Luo, B.: On-line handwritten flowchart recognition based on logical structure and graph grammar. In: 5th International Conference on Information Science and Technology (ICIST), pp. 424–429 (2015)
5. Álvaro, F., Sánchez, J.A., Benedí, J.M.: An integrated grammar-based approach for mathematical expression recognition. Pattern Recognit. **51**, 135–147 (2016)
6. Awal, A.M., Mouchère, H., Viard-Gaudin, C.: Improving online handwritten mathematical expressions recognition with contextual modeling. In: Proceedings of the 12th International Conference on Frontiers in Handwriting Recognition, pp. 427–432 (2010)
7. Álvaro, F., Sanchez, J.A., Benedi, J.M.: Recognition of printed mathematical expressions using two-dimensional stochastic context-free grammars. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 1225–1229 (2011)
8. MacLean, S., Labahn, G.: A new approach for recognizing handwritten mathematics using relational grammars and fuzzy sets. Int. J. Doc. Anal. Recognit. **16**(2), 139–163 (2013)
9. Celik, M., Yanikoglu, B.: Probabilistic mathematical formula recognition using a 2D context-free graph grammar. In: International Conference on Document Analysis and Recognition (ICDAR), pp. 161–166 (2011)
10. Rekers, J., Schürr, A.: Defining and parsing visual languages with layered graph grammars. J. Vis. Lang. Comput. **8**(1), 27–55 (1997)
11. Han, F., Zhu, S.C.: Bottom-up/top-down image parsing by attribute graph grammar. IEEE Int. Conf. Comput. Vis. (ICCV) **2**, 1778–1785 (2005)
12. Blostein, D., Grbavec, A.: Recognition of mathematical notation. In: Wang, P., Bunke, H. (eds.) Handbook of Character Recognition

and Document Image Analysis, pp. 557–582. World Scientific, Singapore (1997)
13. Chan, K.F., Yeung, D.Y.: Mathematical expression recognition: a survey. Int. J. Doc. Anal. Recognit. **3**, 3–15 (2000)
14. Zanibbi, R., Blostein, D.: Recognition and retrieval of mathematical expressions. Int. J. Doc. Anal. Recognit. **15**(4), 331–357 (2012)
15. Miyao, H., Maruyama, R.: On-line handwritten flowchart recognition, beautification and editing system. In: International Conference on Frontiers in Handwriting Recognition, pp. 83–88 (2012)
16. Carton, C., Lemaitre, A., Coüasnon, B.: Fusion of statistical and structural information for flowchart recognition. In: 12th International Conference on Document Analysis and Recognition, pp. 1210–1214 (2013)
17. Bresler, M., Phan, T.V., Prusa, D., Nakagawa, M., Hlavác, V.: Recognition system for on-line sketched diagrams. In: 14th International Conference on Frontiers in Handwriting Recognition, pp. 563–568 (2014)
18. Matsakis, N.E.: Recognition of handwritten mathematical expressions. Master's thesis, Massachusetts Institute of Technology, Cambridge (1999)
19. Tapia, E., Rojas, R.: Recognition of on-line handwritten mathematical expressions using a minimum spanning tree construction and symbol dominance. In: Lladós, J., Kwon, Y.B. (eds.) Graphics Recognition. Recent Advances and Perspectives, vol 3088, Springer, Berlin, pp. 329–340 (2004)
20. Zanibbi, R., Blostein, D., Cordy, J.R.: Recognizing mathematical expressions using tree transformation. IEEE Trans. Pattern Anal. Mach. Intell. **24**, 1455–1467 (2002)
21. Álvaro. F., Zanibbi, R.: A shape-based layout descriptor for classifying spatial relationships in handwritten math. In: Proceedings of the ACM Symposium on Document Engineering, pp. 123–126 (2013)
22. Awal, A.M., Mouchère, H., Viard-Gaudin, C.: Towards handwritten mathematical expression recognition. In: Proceedings of the 10th International Conference on Document Analysis and Recognition, pp. 1046–1050 (2009)
23. Awal, A.M., Mouchère, H., Viard-Gaudin, C.: A global learning approach for an online handwritten mathematical expression recognition system. Pattern Recognit. Lett. **35**, 68–77 (2012)
24. Yamamoto, R., Sako, S., Nishimoto, T., Sagayama, S.: 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. Simistira, F., Katsouros, V., Carayannis, G.: Recognition of online handwritten mathematical formulas using probabilistic SVMs and stochastic context free grammars. Pattern Recognit. Lett. **53**, 85–92 (2015)
26. Bunke, H.: Attributed programmed graph grammars and their application to schematic diagram interpretation. IEEE Trans. Pattern Anal. Mach. Intell. PAMI **4**(6), 574–582 (1982)
27. Fahmy, H., Blostein, D.: A survey of graph grammars: theory and applications. In: 11th IAPR International Conference on Pattern Recognition, pp. 294–298 (1992)
28. Baumann, S.: A simplified attributed graph grammar for high-level music recognition. In: ICDAR (1995)
29. Fahmy, H., Blostein, D.: A graph grammar programming style for recognition of music notation. Mach. Vis. Appl. **6**(2), 83–99 (1993)
30. Lavirotte, S., Pottier, L.: Optical formula recognition. In: Proceedings of the Fourth International Conference on Document Analysis and Recognition, vol. 1, pp. 357–361 (1997)
31. Younger, D.H.: Recognition and parsing of context-free languages in time n^3. Inf. Control **10**(2), 189–208 (1967)
32. Yuan, Z., Pan, H., Zhang, L.: A novel pen-based flowchart recognition system for programming teaching. In: Wang, F.L., Miao, L., Zhao, J., He, J., Leung, E.W. (eds.) Advances in Blended Learning, pp. 55–64. Springer, Berlin (2009)
33. Deng, Y., Kanervisto, A., Ling, J., Rush, A.M.: Image-to-markup generation with coarse-to-fine attention. In: Proceedings of the 34th International Conference on Machine Learning, ICML, pp. 980–989 (2017)
34. Zhang, J., Du, J., Zhang, S., Liu, D., Hu, Y., Hu, J., Wei, S., Dai, L.: Watch, attend and parse: an end-to-end neural network based approach to handwritten mathematical expression recognition. Pattern Recognit. **71**, 196–206 (2017)
35. Le, A.D., Nakagawa, M.: Training an end-to-end system for handwritten mathematical expression recognition by generated patterns. In: 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), vol. 01, pp. 1056–1061 (2017)
36. Zhang, T., Mouchère, H., Viard-Gaudin, C.: A tree-BLSTM-based recognition system for online handwritten mathematical expressions. Neural Comput. Appl. (2018). https://doi.org/10.1007/s00521-018-3817-2
37. Zhang, J., Du, J., Dai, L.: Multi-scale attention with dense encoder for handwritten mathematical expression recognition. In: 24th International Conference on Pattern Recognition, pp. 2245–2250 (2018)
38. Zhang, J., Du, J., Dai, L.: Track, attend, and parse (TAP): an end-to-end framework for online handwritten mathematical expression recognition. IEEE Trans. Multimed. **21**(1), 221–233 (2019)
39. Keysers, D., Deselaers, T., Rowley, H.A., Wang, L., Carbune, V.: Multi-language online handwriting recognition. IEEE Trans. Pattern Anal. Mach. Intell. **39**(6), 1180–1194 (2017)
40. Pflatz, J., Rosenfeld, A.: Web grammars. In: Proceedings of First International Joint Conference on Artificial Intelligence, pp. 193–220 (1969)
41. Grune, D., Jacobs, C.J.H.: Parsing Techniques: A Practical Guide, 2nd edn. Springer, Berlin (2008)
42. Boullier, P., Nasr, A., Sagot, B.: Constructing parse forests that include exactly the N-best PCFG trees. In: Proceedings of the 11th International Conference on Parsing Technologies, pp. 117–128 (2009)
43. Delaye, A., Anquetil, E.: Hbf49 feature set: a first unified baseline for online symbol recognition. Pattern Recogninit. **46**(1), 117–130 (2013)
44. Mouchère, H., Viard-Gaudin, C., Zanibbi, R., Garain, U.: ICFHR2016 CROHME: Competition on recognition of online handwritten mathematical expressions. In: 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 607–612 (2016)
45. Julca-Aguilar, F., Mouchère, H., Viard-Gaudin, C., Hirata, N.S.T.: Progress in pattern recognition, image analysis, computer vision, and applications. In: 20th Iberoamerican Congress, Springer International Publishing, Cham, chap Top-Down Online Handwritten Mathematical Expression Parsing with Graph Grammar, pp. 444–451 (2015)
46. Julca-Aguilar, F., Viard-Gaudin, C., Mouchère, H., Medjkoune, S., Hirata, N.: Mathematical symbol hypothesis recognition with rejection option. In: 14th International Conference on Frontiers in Handwriting Recognition (2014)
47. Julca-Aguilar, F., Hirata, N.S.T., Mouchère, H., Viard-Gaudin, C.: Subexpression and dominant symbol histograms for spatial relation classification in mathematical expressions. In: 23rd International Conference on Pattern Recognition (ICPR), pp. 3446–3451 (2016)
48. Le Cun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient based learning applied to document recognition. Proc. IEEE **86**(11), 2278–2324 (1998)
49. Awal, A.M., Feng, G., Mouchère, H., Viard-Gaudin, C.: First experiments on a new online handwritten flowchart database. Document Recognition and Retrieval XVIII. San Fransisco, United States, pp. 7874–78740A (2011)

50. Bresler, M., Prùa, D., Hlavác, V.: Modeling flowchart structure recognition as a max-sum problem. In: 12th International Conference on Document Analysis and Recognition, pp. 1215–1219 (2013)
51. Lemaitre, A., Mouchère, H., Camillerapp, J., Coüasnon, B.: Interest of Syntactic Knowledge for On-Line Flowchart Recognition, pp. 89–98. Springer, Berlin (2013)
52. Wang, C., Mouchère, H., Viard-Gaudin, C., Jin, L.: Combined segmentation and recognition of online handwritten diagrams with high order Markov random field. In: 15th International Conference on Frontiers in Handwriting Recognition (ICFHR), pp. 252–257 (2016)