An Incremental On-line Parsing Algorithm for Recognizing Sketching Diagrams

Joan Mas Romeu, Gemma Sanchez, Josep Llados, Bart Lamiroy

An Incremental On-line Parsing Algorithm for Recognizing Sketching Diagrams

Joan Mas, Gemma Sánchez, Josep Lladós
Computer Vision Center, Dept. Ciències de la Computació
Edifici O, UAB, 08193 Bellaterra, Spain
{jmas,gemma,Josep}@cvc.uab.es

Bart Lamiroy
INPL-LORIA,Ecole des Mines
Nancy CEDEX, France
Bart.Lamiroy@loria.fr

Abstract

This paper presents a syntactic recognition approach for on-line drawn graphical symbols. The proposed method consists in an incremental on-line predictive parser based on symbol descriptions by an adjacency grammar. The parser analyzes input strokes as they are drawn by the user and is able to get ahead which symbols are likely to be recognized when a partial subshape is drawn in an intermediate state. In addition, the parser takes into account two issues. First, symbol strokes are drawn in any order by the user and second, since it is an on-line framework, the system requires real-time response. The method has been applied to an on-line sketching interface for architectural symbols.

1. Introduction

Sketch Understanding is a discipline of growing interest due to the advent of new pen-based devices. It makes close the domains of document analysis and human computer interaction. A sketch can be defined as an on-line document consisting of a set of hand drawn strokes that represent reality concepts in a rough way. Sketch understanding is thus concerned on the association of valid semantic interpretation of such strokes in the domain where they appear. It involves the use of a pen-based interface that allows the user to draw a sketch in a natural way. Examples of sketch understanding applications are Landay [5] and Alvarado [1].

Symbol recognition is at the heart of a sketch understanding scenario, with the remark that it is performed in on-line mode. It requires representation and recognition approaches able to cope with two difficulties. First the distortion involved in hand drawn strokes and second the orderless processing of the input strokes.

The problem of symbol recognition has been deeply studied in Document Image Analysis and in particular, in Graphics Recognition. A lot of effort has been made in the last decade to develop good symbol and shape recogni-

tion methods inspired in either structural or statistic pattern recognition approaches [6]. The basis of a symbol recognition process is to choose the appropriate shape descriptor depending on the working domain. For a good survey on shape descriptors see [10]. Generally speaking, shape descriptors can be classified in *Global* (or pixel-based) descriptors or *Structural* descriptors. The former are usually based on representing the symbol by a feature vector and the recognition process can be formulated by a statistical approach. Examples are Zernike Moments, Geometric Moments or Fourier Descriptors. In structural description methods, a symbol is described in terms of basic primitives such as arcs, segments or regions, and spatial relationships among them. These methods require a primitive extraction method as polygonal approximation, curve decomposition, etc. Strings, graphs or grammars are usual structural and syntactic descriptors. Therefore, the recognition in such cases is performed by a matching or a parsing process. Symbol recognition in on-line sketch processing has the added value of using dynamic information. It allows to use curvature, speed or pressure information of strokes. In addition, in sketching frameworks, symbols can be classified in two categories: freehand symbols and gestures.

Syntactic approaches are usually used for sketch understanding. At symbol level, a grammar allows to describe a shape in terms of its structuring primitives, usually straight lines, arcs, regions, and the spatial relationships among them. At semantic level, a sketch usually is drawn according to a diagrammatic notation or a visual language. In that case, a grammar defined in terms of symbols, allows to validate an input graphic diagram. Two major families of grammars can be found, namely one dimensional grammars or string grammars as PDL and Plex grammars introduced on [2], and bidimensional grammars or graph grammars.

The approach proposed in this paper is inspired in string grammars. In particular we propose a parser for symbols described by an adjacency grammar. An adjacency grammar [4] represents the symbols in terms of a set of primitives and the relations among them. Representing the productions as a set of primitives instead of a list, provides to

the grammar the issue of no order on the input of the primitives.

A parser is the process to recognize a symbol using a syntactic approach. Given an input symbol the parser determines if it belongs to a class modelled by a grammar. Formally it is the process that returns if a given input belongs to the language generated by the grammar, $L(G)$. Depending on the kind of grammar used to describe the symbol we will have different classes of parsers to recognize it. There exists parsers applied to graph grammars [9], string grammars, relational grammars and so on.

There are other desirable characteristics for a parser coping with online frameworks or sketchy. One is to be able of taking into account possible modifications on the input and to analyze only one part. Another is to give a feedback to the user with a partial input, that kind of parsers are called *predictive*.

In this paper we present an incremental parser approach to recognize hand drawn symbols in a sketchy framework, see Fig. 1 for some illustrative examples. Our approach parses input strokes in real time according to an adjacency grammar that models symbol classes. The grammar described in [] represents symbols in terms of primitives (segments) and relational rules (adjacency, parallelism,...) among them. Although symbols are processed as the strokes are drawn, the main contribution of the parser is that it can process the input strokes to recognize symbols in an order free way. Since the parser analyzes the input in an incremental way, it has also a predictive nature, i.e., once a partial input has been processed, the parser is able to propose to the user a set of final acceptance states (valid symbols) that have as subshapes the current intermediate state. To measure the distortion, each rule has an associated value that represents how well the symbol matches the model.
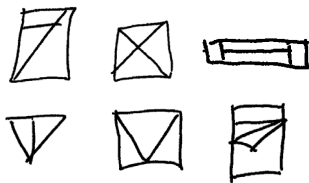
Figure 1. Online sketchy instances.

This paper is organized as follows: Section 2 describes the grammatical formalism that has been used. Section 3 presents the parser methodology used on this work. In section 4 the experimental part of the work is presented and finally section 5 exposes the conclusions of the work.

2. Symbol representation by an Adjacency Grammar

The different classes of graphical symbols are represented by rules using an Adjacency Grammar. This kind of grammars describe a symbol in terms of a set of primitives and the constraints among them. Adjacency grammars were first introduced by Jorge in [4]. This formalism has been adapted by the authors in [7], adding a distance measure that specifies how well the symbol performs the model and the capability of inferring the rules describing each class of symbols from a set of sketched on-line instances.

Formally an Adjacency Grammar is defined as a 5-tuple $G = \{V_t, V_n, S, P, C\}$ where:

- $\mathbf{V_t}$: is the alphabet of terminal symbols. In this work, $V_t = \{segment, arc\}$, *arc* refers to open or closed arcs.

- $\mathbf{V_n}$: is the alphabet of non-terminal symbols.

- $\mathbf{S} \,\epsilon \mathbf{V_n}$: is the start symbol of the grammar.

- $\mathbf{C}$: is the set of adjacency constraints. In this work, C = {incident, adjacent, intersects, parallel, perpendicular}, see Fig. 2.

- $\mathbf{P}$: are the productions of the grammar defined as:

$$\alpha \rightarrow \{\beta_1, \ldots, \beta_j\} \; if \; \Gamma_1(\Phi_1, c_1), \ldots, \Gamma_k(\Phi_k, c_k) \tag{1}$$

Where $\alpha \,\epsilon\, V_n$ and $\forall i \,\epsilon\, [1 \ldots j] \; \beta_i \,\epsilon\, \{V_t \cup V_n\}$, constitute the possibly empty multiset of terminal and non-terminal symbols. Γ_l are the adjacency constraints defined on the attributes of the subsets $\Phi \subset \{\beta_1, \ldots, \beta_j\}$, and c_l are the cost functions associated to each constraint. In our case the attributes are:

. **For segments:** $(\mathbf{X_0}, \mathbf{Y_0})$ and $(\mathbf{X_f}, \mathbf{Y_f})$, the starting and end points respectively.

. **For arcs:** $\mathbf{X_c}, \mathbf{Y_c}$ and R, the center of the arc and its radius respectively.

. **For subshapes:** Bounds, denotes the bounding-box of the subsymbol.

The cost function value is in the range $[0..1]$, being 0 when the constraint is well accomplished and 1 in the worst case. The productions of the grammar describes the topology of the symbols.

3. Symbol Recognition: Parser Methodology

Syntactic symbol recognition requires a grammar to represent a class of symbols and a parser to recognize it. Then a
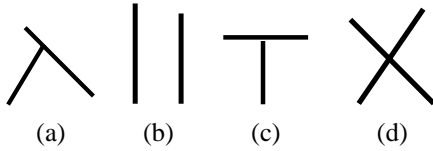
Figure 2. Constraints: (a) Incidence, (b) Parallel, (c) Perpendicular and (d) Intersects.

parser is the process that decides if a given input w belongs to the language generated by the grammar G, that means if $w \in L(G)$. A parser computes the sequence of rules that allow to rewrite w from the start symbol S of the grammar G. Depending on the direction the sequence is constructed we can have *bottom-up* (from w to S) or *top-down* parsers (from S to w). There are other possible classifications of parsers depending on the way the input is analyzed, as: *Sequential parsers*, *Syntax Directed*, *Incremental*, etc.

The parser presented in this work has to cope with an on-line sketchy input framework. This framework gives a list of strokes representing a symbol in an incremental way without a predefined order, and it expects an answer from the parser even if the symbol is not completed. We notice that strokes representing a symbol may contain over-tracing. In the following paragraphs we define the characteristics of our parser that make possible to work in this framework.

Working with an incremental list of strokes the parser has to be able to modify its state with each input stroke. That leads us to an incremental parser, see Costagliola et al in [3].

On-line input frameworks need to obtain which symbol is drawn even if the input is not completed. That leads us to a predictive sub-sentence detector parser. This means that the parser analyzes the input stroke by stroke and constructs a list of possible rewriting rules. In the process the sub-symbols appearing in the input are detected.

The parser presented in this paper works in an on-line input framework. The input strokes are treated one by one and in a predictive way they are grouped to construct rules. That means the parser is bottom-up, predictive and incremental. Informally speaking the parser incrementally process the input strokes according to the underlying grammar until an acceptance or rejection state is achieved. Each input generates a transition between two intermediate states. A state represents the processed input and, accordingly, the predicted symbols that can be accepted in future states. Formally let us define a state Q_i as a triplet $Q_i(O_i, P_i, M_i)$, where:

- **O_i**(Open): is the list of possible accepted rules, indexed by the restrictions they contain and its number of occurrences.

- **P_i**(Prediction): is the list of rules that have been partially applied after processing the input up to the current state.

- **M_i**(Memory): is the processed input strokes together with their restrictions.

The state Q_0 has P_0 as the empty list of rules, M_0 as the empty list of strokes and O_0 as all the rules of the grammar.

A transition from a state Q_i to a state Q_j is generated after getting a new stroke w_i from the input. M_j is generated adding to M_i the stroke w_i and the processed restrictions between it and the strokes in M_i. O_j is an updated version of Oi after removing those rules incompatible with the list of restrictions of M_j. Finally P_j is created as the list of partially processed rules from O_j.

An example of parsing input strokes is shown in Fig. 4. Given the set of symbols of Fig. 4(a) a grammar is constructed in an automatic way as described in [8]. Two rules of the grammar are presented in Fig. 3, and the rest are computed analogously. From the list of rules forming the grammar the state Q_0 is constructed, see Q_0 in Fig. 4(b), we can observe that all the rules are presented in P_0 organized by the constraints presented on them and its number of occurrences. P_0 is empty and also M_0. Then the parser process several strokes and recompute each state till it arrives to state Q_4, there 4 strokes have been analyzed, printed in Fig. 4(c) and entered in the order marked by the number beside. We can observe that some rules in O_0 have been disappeared in O_4 because they are already non compatible with the analyzed sequence. We can also see, in P_4 the predicted rules that can be drawn by the user, all of them except rule 5 because the drawn subshape is a rectangle and all the rules have this subsymbol. Finally in Fig. 4(d) we can see the final state Q_6 where the symbol represented by rule4 have been recognized.

Figure 5 shows two situations to illustrate how the parser can predict a rule depending on the order of the input strokes. In both the grammar consists of the same set of productions. In the first, Fig. 5(a) the input is introduced to the system as marks the numbers beside the strokes. When the two first primitives are introduced to the system the computed constraints Incidence and perpendicular appear in all the symbols of the grammar. This makes that the set of possible rules in O_2 is the same than in O_0. In fact till the last step when the stroke number 6 is introduced the parser do not know which symbol is drawn. On the contrary, in Fig. 5(b), the set of possible rules in O_2 is already rule2, as it is the only symbol containing two crossed lines and they have been introduced first.

The spatial and temporal complexity of the parser have been analyzed. Spatial complexity is of order $\mathcal{O}\left(n^2\right)$ where n is the maximum of the orders on the sets C and P. The temporal complexity is of order $\mathcal{O}\left(n^3\right)$ being n the max-
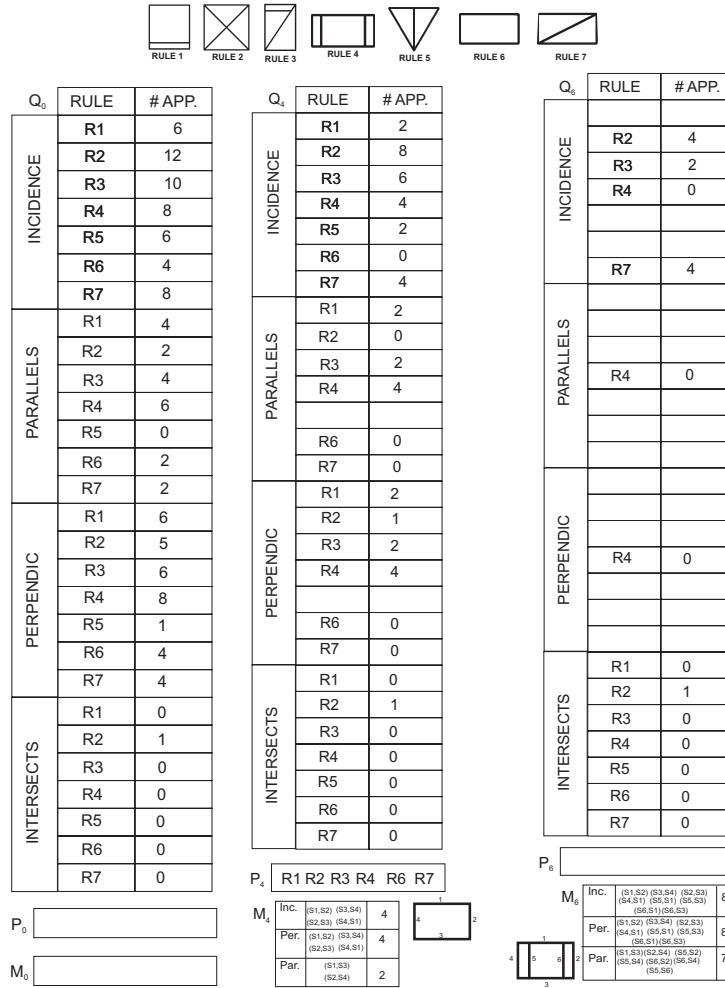
Figure 4. Scheme of the parser methodology.

imum between the order of the set C and the number of primitives on the input.

4. Experimental Results.

Two qualitative issues have been evaluated in our experiments. First the ability of the parser to analyze a set of hand-drawn valid inputs in a predictive way. Afterwards the ability to detect a valid symbol as a subshape of an invalid input.

To perform the experiments we have used a benchmarking database of 169 on-line instances belonging to 7 different classes, see Fig. 4(a). 6 of the classes share a common subsymbol, a square. The instances are from 20 different persons what allows us to test the independence on the order of the primitives. All the instances have been detected correctly.

For non-valid inputs the parser gives as a result the major sub-sentence that it is able to detect. See the examples of Fig. 6. Two symbols not belonged to the set of models represented by the grammar are analyzed. For each one the numbers in Fig. fig:notvalidinput1 represent the input order of the strokes. In Fig. 6(a) after processing the strokes 1 to 4 the parser recognize it as a square (rule 6), but if the user continues drawing then the parser rejects the symbol. In Fig. fig:notvalidinput1(b) the strokes 1 to 4 are also recognized as a square. After drawing the stroke 5 rule 1 is accepted, and when the stroke 6 is added rule 4 is accepted. Finally after drawing the stroke 7 the final symbol is rejected since it does not match any predefined model.

RULE3 --> {S1,S2,S3,S4,S5,S6}	RULE4 --> {S1,S2,S3,S4,S5,S6}
Inc(S1,S2,&c)	Inc(S1,S2,&c)
Inc(S2,S3,&c)	Inc(S2,S3,&c)
Inc(S3,S4,&c)	Inc(S3,S4,&c)
Inc(S4,S1,&c)	Inc(S4,S1,&c)
Inc(S5,S1,&c)	Inc(S5,S1,&c)
Inc(S6,S1,&c)	Inc(S5,S3,&c)
Inc(S6,S2,&c)	Inc(S6,S1,&c)
Inc(S6,S3,&c)	Inc(S6,S3,&c)
Inc(S6,S4,&c)	Per(S1,S2,&c)
Inc(S5,S6,&c)	Per(S2,S3,&c)
Per(S1,S2,&c)	Per(S3,S4,&c)
Per(S2,S3,&c)	Per(S4,S1,&c)
Per(S3,S4,&c)	Per(S5,S1,&c)
Per(S4,S1,&c)	Per(S5,S3,&c)
Per(S5,S1,&c)	Per(S6,S1,&c)
Per(S5,S3,&c)	Per(S6,S3,&c)
Par(S1,S3,&c)	Par(S1,S3,&c)
Par(S2,S4,&c)	Par(S2,S4,&c)
Par(S2,S5,&c)	Par(S2,S5,&c)
Par(S5,S4,&c)	Par(S5,S4,&c)
	Par(S2,S6,&c)
	Par(S6,S4,&c)
	Par(S5,S6,&c)

Figure 3. Example of rules describing a symbol. Being Par parallelism, per perpendicularity and Inc incident.)
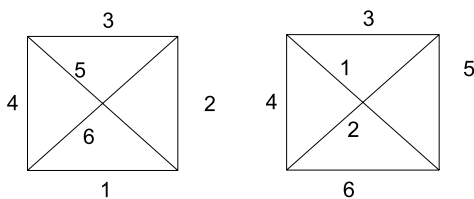

Figure 5. Different Situations on the input order of the strokes.

5. Conclusions

We present a parser methodology able to analyze an on-line sketched input, meanwhile it is introduced to the system. The parser is also able to predict which symbol the user is intended to draw. The complexity of the parser has been studied and we may conclude that in terms of spatial complexity is linear and in terms of temporal complexity follows the formula described in the section 3.

Future work on the parser will be in terms of apply the methodology with other grammar formalisms that take into account the primitives forming the rules and some constraints among them. Another kind of constraints should be also defined in our grammar formalism.
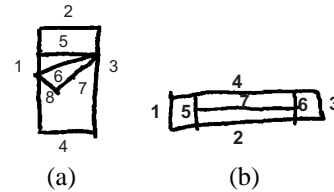
Figure 6. Non-valid input situations.

Acknowledgments

References

[1] C. Alvarado and R. Davis. Resolving ambiguities to create a natural computer-based sketching environment. In *IJCAI*, pages 1365–1374, 2001.

[2] H. Bunke. Hybrid pattern recognition methods. In H. Bunke and A. Sanfeliu, editors, *Syntactic and Structural Pattern Recognition. Theory and Applications*, pages 307–347. World Scientific Publishing Company, 1990.

[3] G. Costagliola and V. Deufemia. Visual language editors based on lr parsing techniques. In *Proceedings of 8th International Workshop on Parsing Technologies*, 2003. Nancy, France.

[4] J. Jorge and E. Glinert. Online parsing of visual languages using adjacency grammars. In *Proceedings of the 11th International IEEE Symposium on Visual Languages*, pages 250–257, 1995.

[5] J. Landay and B. Myers. Sketching interfaces: Toward more human interface design. *IEEE Computer*, 34(3):56–64, March 2001.

[6] J. Lladós, E. Valveny, G. Sánchez, and E. Martí. Symbol recognition: Current advances and perspectives. In D. Blostein and Y. Kwon, editors, *Graphics Recognition: Algorithms and Applications*, pages 104–127. Springer, Berlin, 2002. Vol. 2390 of LNCS.

[7] J. Mas, G. Sanchez, and J. Llados. An incremental parser to recognize diagram symbols and gestures represented by adjacency grammars. In J. L. W. Liu, editor, *Graphics Recognition: Ten Year Review and Perspectives*, volume 3926 of *Lecture Notes in Computer Science*, pages 252–263. Springer-Verlag, 2006.

[8] J. Mas Romeu, B. Lamiroy, G. Sanchez, and J. Llados. Automatic adjacency grammar generator from user drawn sketches. In *Proceedings of 18th International Conference on Pattern Recognition*, august 2006. Hong-Kong.

[9] J. Rekers and A. Schurr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages and Computing*, 8(1):27–55, 1997.

[10] D. Zhang and G. Lu. Review of shape representation and description techniques. *Pattern Recognition*, 37(1):1–19, January 2004.