An Error-Correction Graph Grammar to Recognize Textured Symbols *

Gemma Sánchez[1,2], Josep Lladós[1], Karl Tombre[2]

[1]Computer Vision Center, Dept. Informàtica,

Universitat Autònoma de Barcelona, 08193 Bellaterra (Barcelona), Spain

[2]Loria, Campus scientifique, B.P. 239, 54506 Vandœuvre-lès-Nancy CEDEX, France

Abstract

This paper presents an algorithm for recognizing symbols with textured elements in a graphical document. A region adjacency graph represents the graphical document, with the nodes being polygons and the edges the neighborhood relations between them. The textured symbols are modeled by a graph, where nodes are polygons (represented by strings) or textured areas (represented by a graph grammar with error-correction rules). The recognition process is done by a graph matching process that uses a string edit distance to recognize the static parts of the symbol and a parsing process that segments the subgraph in the original graph, following the rules of the graph grammar.

1 Introduction

The detection and recognition of symbols is an important area within graphics recognition. One of the key issues is to be able to describe what a symbol is supposed to be and of which graphical components it is composed. There are typically two kinds of structures that can

*Work partially supported by project CICYT TIC2000-0382.

be considered to form a symbol: those consisting of fixed primitives, i.e. parts that can be represented by a prototype pattern, and structures consisting of a repetitive pattern, such as crosshatched areas in engineering drawings, tiled patterns in architectural plans, etc. Each type of symbol induces its own recognition strategy. Prototype-based symbols can be recognized by many kinds of pattern matching approaches, whereas repetitive structures need a strategy able to iteratively capture the repetitions.

In a previous work, we have studied prototype-based symbol recognition and textured symbol recognition using separate strategies [8]. In this work, we propose a unified representational structure and recognition process for symbols with textured or not textured parts, and we focus on the syntactic modeling and recognition of texture-based components.

The symbol recognition process is based on the graph matching process explained in [8], but now the nodes of the graph can represent static parts as well as textured parts. The present paper focuses on explaining the textured parts recognition, as the static parts recognition remains the same as in [8]. In this work, the symbol recognition process is applied to an architectural drawing understanding application. An input graphical document is transformed, assuming a previous vectorization process, into a RAG (*Region Adjacency Graph*) $H(V, E, LV, LE, FV, FE)$, constructed as in [8]. Thus, in the attributed graph $H(V, E, LV, LE, FV, FE)$, V is the set of nodes corresponding to regions and E is the set of edges representing the region adjacencies. Every graph vertex, $v \in V$, is given as attribute a cyclic string that represents the sequence of lines forming the shape contour of the region. Every graph edge (v, v') is given as attribute the straight segment connecting the centers of the shapes v and v'. In this kind of graphical documents, different kinds of textured symbols can appear (see Fig. 1); some of them are two-dimensional, as those in Fig. 1(a) and Fig. 1(b), others are one-dimensional, as in Fig. 1(c) and Fig. 1(d); some are formed by only one kind of shape, or primitive, as in Fig. 1(a), Fig. 1(c), and Fig. 1(d), and others with two, as in Fig. 1(b). All of them can be characterized by a primitive or some primitives which are placed following a rule. This structure can be represented by a RAG. In the RAG, some distortions may appear, due to the scanning process or to the vectorization, while in the textured symbol appearing in the RAG, other kinds of distortions may appear due to the finite nature of the symbol; in particular, symbol primitives may be fragmented on the borders of a textured zone. Therefore, due to the process used to obtain the RAG representing the document, and to the finite nature of the texture, four kinds of distortions may appear in the symbol, presented in Fig. 2: (i) Distortions in the shapes forming the texture, see Fig. 2(a), (ii) Distortions in the placement rules of the shapes forming the texture, see Fig. 2(b), (iii) Fusion of two or more elements forming the texture, see Fig. 2(c), (iv) Partial occlusion of the elements forming the texture, due to the finite nature of the textured area, see Fig. 2(d). The present work presents a graph grammar, based on the one presented in [10], improved with some error-correction rules, to model textured symbols. The automatic grammar inference process and the parser process to recognize textured symbols in a given RAG are also explained.

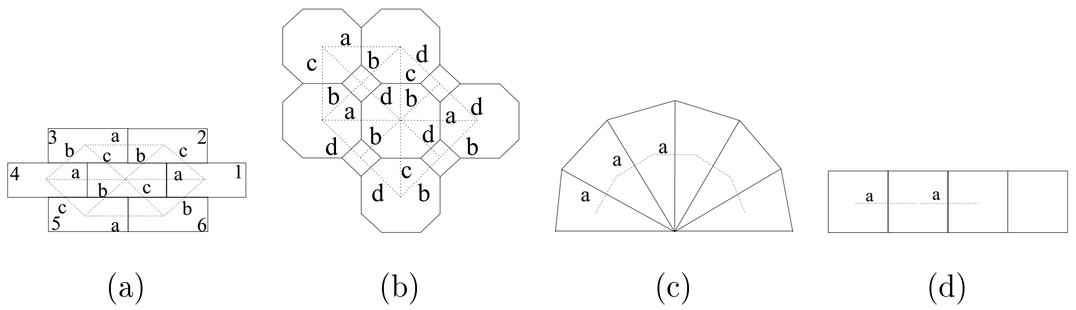

Figure 1: (a) 1-primitive 2D textured symbol. (b) 2-primitives 2D textured symbol. (c) 1-primitive 1D textured symbol. (d) 1-primitive 1D textured symbol.

The paper is structured as follows: in section 2, the grammar to model textured components and its inference process is presented; section 3 explains the recognition process by means of parsing a given graph using a graph grammar with error correction. Section 4 is devoted to conclusions and future work.

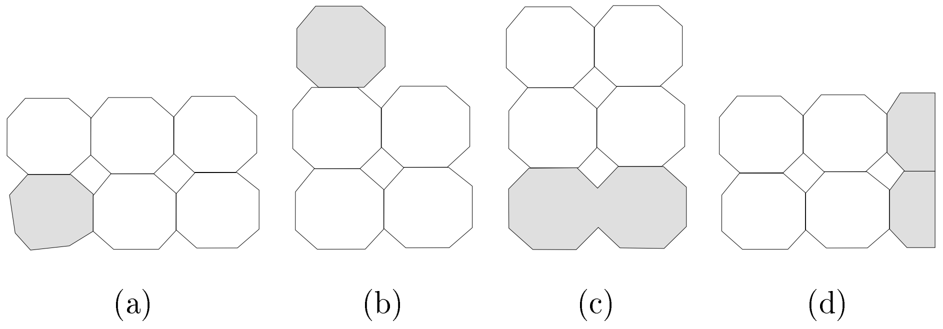
Figure 2: (a) Shape distortions. (b) Placement rules distortions. (c) Merging of the shapes. (d) Occlusion of the shapes.

2 Graph Grammar Inference

Graph grammars have been used in syntactic pattern recognition to extract the description of an input pattern or to classify it. When patterns are multidimensional, their description using string grammars is difficult, and then graph grammars can represent the space relations among their parts. Graph grammars have been used in diagram recognition [1], music notation [3], visual languages [6], drawing dimensions recognition [2, 9] and mathematical formula recognition [5]. In this section we present the definition of our grammar and the way to infer it automatically from a given example.

Given a textured symbol X, we infer a context-sensitive error-correction graph grammar to recognize it. A graph grammar is a six tuple $G = (\Sigma, \Delta, \Omega, P, S)$ where Σ is the alphabet of non-terminal node labels, Δ is the alphabet of terminal node labels, Ω is the alphabet of terminal edge labels, P is the finite set of graph productions or rewriting rules, and S the set of initial graphs, usually consisting of one node with a non-terminal label. A graph production P is a four tuple $P = (h_l, h_r, T, F)$, where h_l is the left hand graph, h_r is the right hand graph, T is the embedding transformation $T = \{(n, n')|n \in V_{h_l}, n' \in V_{h_r}\}$ and F is the set of attribute transferring functions, for edges and nodes in h_r. The direct derivation of a graph H' from a host graph H by applying a production $P = (h_l, h_r, T, F)$, $H \xrightarrow{P} H'$ is defined by locating h_l^{host}, a subgraph of H isomorphic to h_l, and replacing h_l^{host} by h_r^{host}, a subgraph isomorphic to h_r. Let $H^{-h_l^{host}}$ be the graph remaining after deleting h_l^{host} from H, the edges between the subgraph h_r^{host} and $H^{-h_l^{host}}$ are *the embedding of*

h_r^{host} *in* $H^{-h_l^{host}}$, and defined by T. In the present work we use induced isomorphism. An induced subgraph of a graph must include all local edges of the host graph, i.e. all edges that connect two nodes in the subgraph. For a more comprehensive explanation of graph grammars, see [7] and [4].

Given a textured symbol X consisting of regular repetitions of n different primitives, $N_X = \{N_1, \ldots, N_n\}$, and m different kinds of neighborhoods $R_X = \{R_1, \ldots, R_m\}$, the inference process is as follows: First, for each primitive N_i forming X, we define one hierarchy of non-terminal node labels L_i, and three terminal node labels l_i, cl_i, dl_i, representing the primitive, the primitive cut on a border and more than one primitive merged into one shape respectively, and the start node label S', having $\Delta = \{L_1, \ldots, L_n, S'\}$ and $\Sigma = \{l_1, cl_1, dl_1 \ldots, l_n, cl_n, dl_n\}$. For each hierarchy L_i we define two derived labels, IL_i and RL_i, representing an inserted node and a real one, respectively. Then, we define one terminal edge label e_i for each kind of neighborhood R_i, having $\Omega = \{e_1, \ldots, e_m\}$. For each primitive $N_i \in N_X$ we denote as $NR_i = \{R^i_1, \ldots, R^i_{m_i}\}$, $\forall j = 1 \ldots m_i, R^i_j \in R_X$, the set of different kinds of neighborhoods that N_i has in counter-clockwise order, for example for the symbol in Fig. 1(a), N_i being the rectangle in the middle, $NR_i = \{a, b, c\}$. For each primitive $N_i \in N_X$ we denote as $NN_i = \{N^i_1, \ldots, N^i_{n_i}\}$, $\forall j = 1 \ldots n_i, N^i_j \in N_X$, the set of neighbors that N_i has in counter-clockwise order, starting with the N^i_1 such that the label of the edge (N_i, N^i_1) is R^i_1, for example in the same symbol shown in Fig. 1(a), N_i being the rectangle in the middle, and denoting each node neighbor with the natural number the rectangle has in its corner, i.e. $N_1, \ldots, N_6$, $NN_i = \{N_1, \ldots, N_6\}$. Then the productions are defined in the following way:

1. For each $N_i \in N_X$ we define a production with the left hand side being the start node S', and the right hand side a subgraph with a node n labeled as l_i, and with all the closed loops starting on N_i, labeling the neighboring nodes of n with the following non-terminal node labels $\{L^i_1, \ldots, L^i_{m_i}\}$ in the counter-clockwise order, and the edges among all of them with their corresponding terminal label.

2. Each non terminal node can have one or more terminal nodes as neighbors. Each terminal node has all its neighbors at least labeled as a non terminal node. Then,

for each shape N_i, we generate one set of productions for each number of terminal nodes that N_i can have, i.e. we generate one set of productions when N_i has one terminal node as a neighbor, then when it has two, and so on until n_i, and each production in each of this set of productions is defined taking the first terminal neighbor following one possible kind of neighboring.

3. The inserted nodes allow us to end the texture and to correct errors of shapes appearing only partially because they are on the border of the texture or shapes which are merged because of distortions in the acquisition process. For these inserted nodes, one rule is added that substitutes it by lambda, the cut terminal label, cl_i, or the joined terminal label, dl_i.

Figure 3 shows the example of the graph grammar representing the texture in Fig. 1(b). There are two kinds of primitives forming the texture: the square and the octagon. For the octagon, three terminal labels are defined: o, co, do, representing the normal octagon, the cut one, and the one representing some octagons merged into one shape, and one hierarchy of non-terminal labels, O, which has as derived labels, Ro representing an existing node, and Io, representing an inserted one. For the square, three other terminal labels are defined, r, cr, dr, representing the normal square, the cut one, and the one representing some squares merged into one shape, and one hierarchy of non-terminal labels, R, which has as derived labels, Rr representing an existing node, and Ir, representing an inserted one. For each production P, h_l and h_r are graphs whose nodes have a number or a number with $'$, this number represents the embedding rule, i.e. the node on the left side with number i, or the corresponding one on the right side with number i'. Rules 1 and 10 are the starting rules for the square and the octagon respectively, while rules 9 and 40 are the end rules and error-correction rules for the square and the octagon, respectively. Notice that each time a label O appears on the production, it can represent both derived labels, respectively Io and Ro, and the same for label R, with its derived labels Ir and Rr. However, we can only rewrite one non-terminal node by a terminal one when it is a real one, that is Ro or Rr, the only exceptions being the rules to correct errors. We should also notice that each rule is representing itself and its symmetric rule, and the rules with labels with parameters $e1$ and $e2$ are representing four kind of rules: those without the nodes and edges parameterized with them, those with the nodes parameterized with $e1$, those with the nodes parameterized with $e2$, and those with all of them.

3 Recognition Process: Parsing

Given a host graph H representing an architectural drawing, and a graph grammar G representing the textured symbol, the parsing process on H, to recognize the textured symbol, is done in the following way: First, nodes in H are taken one by one until a start rule can be applied to one of them. Let us denote by n this selected node. This means that the shape represented by n is similar to the shape expected on the left side of the starting rule, and that there is a subgraph isomorphism between a subgraph around n and the graph h_r of this starting rule. The similarity between the shapes represented by the nodes is computed by means of the string edit distance explained in [8]. In this process, n is marked as an element forming the textured symbol, and the set of nodes and edges of H around n which have an equivalence at the right hand side of the grammar rule are labeled as the rule points, being the nodes labeled as non-terminal inserted into a list, Lnt, to be analyzed by the parser, while the nodes and edges not found are labeled as inserted nodes and inserted into a separate list, Li, to be analyzed during a post-processing phase, to find possible errors and to finalize the textured symbols. These inserted nodes are pointing to existing nodes in H, which are in a relative position with respect to n similar to the expected corresponding nodes in the h_r of the grammar rule. Then, for each non-terminal node n in Lnt, the rule to be applied each time is directly selected by counting the number of terminal neighbor nodes it has and in which position they start. Then the rule is applied as in the previous step, using a subgraph isomorphism and inserting into Lnt and Li the non-terminal and inserted nodes to be analyzed, respectively. Node n is marked as an element forming the textured symbol. Once all the non-terminal nodes in Lnt are analyzed, we have a set of nodes in H marked as part of the texture, and, surrounding them, the nodes marked as inserted. Then, for each inserted node n in Li, we apply the error-correcting or ending rules. Ending rules delete the inserted nodes

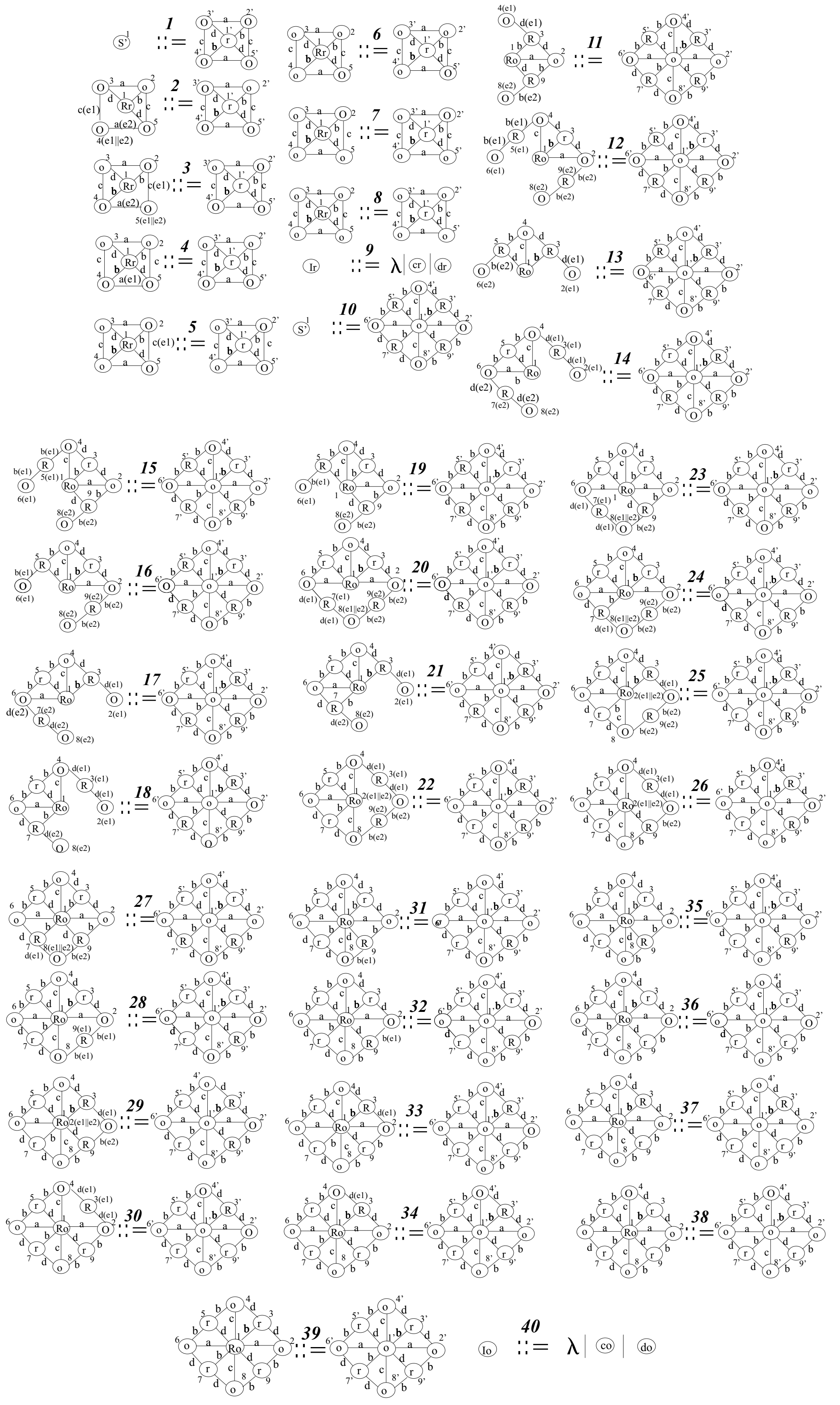

Figure 3: Graph Grammar representing the textured symbol of Fig. 1(b).

while error-correcting rules test if there is a cut shape or a split shape in the position of the inserted node. Each time an error-correcting rule is applied, a cost is associated with it, that quantifies the distortion of the textured symbol from the model. Once this cost reaches a given threshold, no more error-corrections are possible, and the remaining inserted nodes are deleted.

In Fig. 4, an example of a parsing process is presented. Given a graph H in Fig. 4(b) representing the image in Fig. 4(a) we parse it following the grammar presented in Fig. 3. Then in Fig. 4(c) we take the first node and compare it with the two possible shapes obtaining that it is the octagonal shape. Then we apply the first starting rule 10 in Fig. 3, having the graph in Fig. 4(c), where the nodes with Ir or Io are the inserted ones, the node with white o the one we are rewriting as a terminal node, and the nodes with R and O the ones we are rewriting as non-terminal nodes. Then all the non-terminal nodes are in a list and the inserted nodes in a separate list. Taking the first non-terminal node from its list, we apply rule 11 in its symmetric form with parameters $e1$ and $e2$ as false, modifying the graph as it appears in Fig. 4(e), where the nodes already parsed as being part of the symbol appear in grey color. In the same way, we apply the rules marked in Fig. 4(e)–(m), where the parameters T or F denote whether the nodes labeled with $e1$ ad $e2$ do exist or not, respectively, and $Sym.$ denotes whether the rule is applied in its symmetric version. In Fig. 4(f) rule $13(false, false)$ from Fig. 3 is applied, with the resulting graph of Fig. 4(f); we should notice that this shape has some distortions, but can be matched with the original octagon, because the distortions are under a certain threshold. Once the whole list of non-terminal nodes has been analyzed, we are at the point of Fig. 4(n), where all the nodes recognized as forming the textured symbol represented by the grammar in Fig. 3 are in grey and all the inserted nodes appear around them. Then, we apply the grammar rules 9 and 40, to delete the non existing inserted nodes having the graph in Fig. 4(o). From that point, we apply rule 40 to correct errors, selecting the cut shapes and the double ones, represented by the terminal nodes oc and os, respectively. At the end, the whole graph is recognized as the textured symbol.

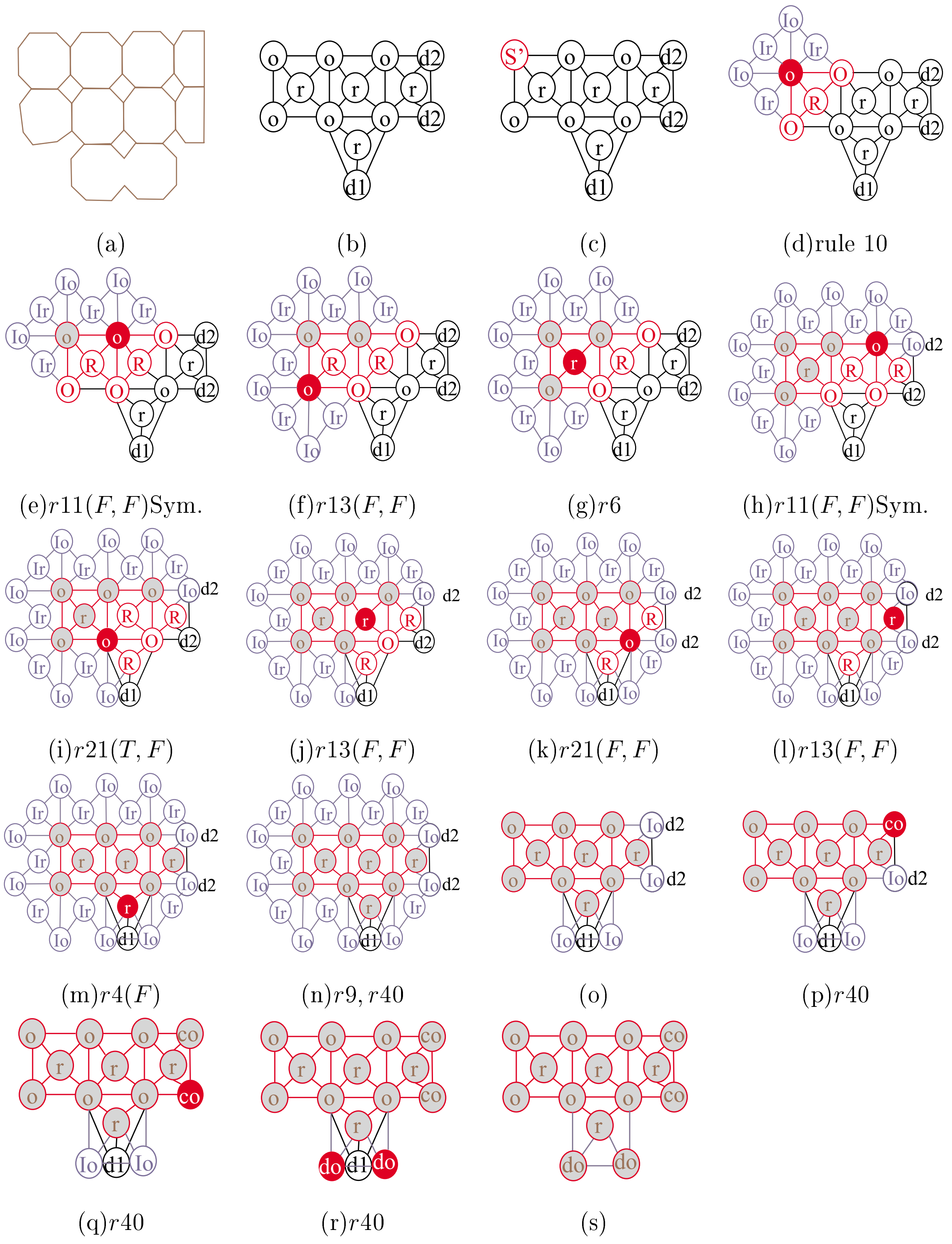

Figure 4: Parsing process of a graph.

4 Conclusions and Future Work

In this paper, we have presented a model to represent textured symbols. The model is based on a graph, with some nodes representing fixed patterns—polygons—and other nodes representing structural textures. Polygons are represented by a string while textures are represented by a graph grammar. The recognition process is done through graph isomorphism, starting a parser process on the graph when a textured part of the symbol has to be found. The grammar allows four kind of distortions in the textured parts of the symbol: Distortions in the shapes forming the texture, distortions in their neighborhood, shapes which are split due to an insufficient resolution in the lines around the shape or because of noise, and shapes which appear only partially because they are at the border of the texture. It is also points out the need to add a cost function to model the distortion of the textured symbol found from its model, in order to control which degree of deformation we allow.

The present work is the first step of a process to recognize textured symbols in graphical documents. Given the nature of such documents, several distortions may appear on these textured symbols. Here, we have presented some solutions for a group of these distortions, but others may appear, due to the acquisition process or because other symbols may appear, superimposed on the texture, so that they create holes in it and several kinds of partitions in its shape. Because of all of that, some distortions have to be allowed, but they must be controlled, to guarantee that the parsing subgraph is really the textured symbol to be recognized. For that purpose, an error distance should be computed by adding a cost function each time we apply a correcting rule. This cost function should be in terms of the distortion of the shapes we are taking as distorted ones and their neighborhoods, but it should be weighted, using the number of correct elements forming the texture, and allowing more distortions when the set of correct elements is larger.

References

[1] H. Bunke. Attributed Programmed Graph Grammars and Their Application to Schematic Diagram Interpretation. *IEEE Transactions on Pattern Recognition and Machine Intelligence, Vol:4, N.6*, pp. 574–582, November 1992.

[2] D. Dori. A syntactic/geometric approach to recognition of dimensions in engineering machine drawings. *Computer Vision, Graphics and Image Processing*, 1989.

[3] H. Fahmy and D. Blostein. A graph grammar programing style for recognition of music notation. *Machine Vision and Aplications (1993) Vol. 6.* pp. 83–99, Springer-Verlag

[4] H. Fahmy and D. Blostein. A Survey of Graph Grammars: Theory and Applications. *12 ICPR*, pp. 294–298, 9-13 October 1994, Jerusalem (Israel).

[5] A. Kosmala, S. Lavirotte, L. Pottier, and G. Rigoll. On-Line Handwritten Formula Recognition using Hidden Markov Models and Context Dependent Graph Grammars. In *Proc. of 5th International Conference on Document Analysis and Recognition*, 1999.

[6] J. Rekers and A. Schürr. Defining and Parsing Visual Languages with Layered Graph Grammars. *Journal of Visual Languages and Computing, Vol. 8, No. 1, London: Academic Press (1997), pp. 27–55.*

[7] G. Rozenberg. Handbook of Graph Grammars and Computing by Graph Transformation. Vol. I, Foundations, World Scientific, 1997.

[8] J. Lladós, G. Sánchez and E. Martí. A string-based method to recognize symbols and structural textures in architectural plans. *Graphics Recognition, Algorithms and Systems. K. Tombre and A. K. Chhabra (eds). Lecture Notes in Computer Science, Springer-Verlag*, 1389:91–103, 1998.

[9] W. Min, Z. Tang, and L. Tang. Using web grammar to recognize dimensions in engineering drawings. *Pattern Recognition*, 26(9):1407–1916, 1993.

[10] G. Sánchez, J. Lladós. A Graph Grammar to Recognize Textured Symbols. To appear in proceedings of ICDAR 2001.