A String Based Method to Recognize Symbols and Structural Textures in Architectural Plans

Josep Lladós, Gemma Sánchez, and Enric Martí

Computer Vision Center - Dep. Informàtica
Universitat Autònoma de Barcelona
08193 Bellaterra (Barcelona), Spain

Abstract. This paper deals with the recognition of symbols and structural textures in architectural plans using string matching techniques. A plan is represented by an attributed graph whose nodes represent characteristic points and whose edges represent segments. Symbols and textures can be seen as a set of regions, i.e. closed loops in the graph, with a particular arrangement. The search for a symbol involves a graph matching between the regions of a model graph and the regions of the graph representing the document. Discriminating a texture means a clustering of neighbouring regions of this graph. Both procedures involve a similarity measure between graph regions. A string codification is used to represent the sequence of outlining edges of a region. Thus, the similarity between two regions is defined in terms of the string edit distance between their boundary strings. The use of string matching allows the recognition method to work also under presence of distortion.

1 Introduction

One of the most promising areas within the field of graphics recognition is the interpretation of maps and plans. An accurate vectorization constitutes a first approach to solve this goal. However, a vectorization only gives the segments constituting the document and their geometrical attributes. Interpreting a document requires an additional stage: understanding the document in terms of its structural elements. Usually maps and plans contain three types of elements: symbols, structural textures and dimensioning elements. This work focuses on two of these three entities: symbols and textures, proposing a structural method that is able to recognize both of them. This method has been applied to architectural drawings, either printed or hand drawn, consisting in floor plan sketches. Symbols represent building elements in the plan, like doors, windows, furniture, etc. They have an a priori defined pattern and their recognition is performed by matching with a set of models stored in a library. In contrast, textures do not represent particular entities but show regions with a particular meaning: tiled floors, level, solid regions, etc. Textures do not have a fixed pattern but are characterized by a repetitive element, called texel, and a structural rule. Interestingly, the problem of texture discrimination in map documents usually assumes a previously defined texture (hatched patterns in most cases [1][3][8]).

In this work, the only a priori assumed constraint is that the texture texel is a polygon with a regular structure but its particular shape and structure, will be inferred in run time.

Since maps and plans are documents mainly composed by lines, *attributed graphs* are an efficient structure to represent them. In this work, the input document is scanned and vectorized obtaining an attributed graph structure. The graph nodes represent the characteristic points of the document (junctions, end points or corner points), and the graph edges approximate the segments between characteristic points. Starting from this structure, a Region Adjacency Graph (RAG) is computed. The RAG nodes represent the regions, i.e. minimal closed loops of the former graph, and the edges are the neighbouring relations between loops. A given graph region (RAG node) is represented by an attributed cyclic string containing the sequence of graph edges defining the region. Thus, the similarity between two regions can be computed in terms of the string edit distance between them. This is the basis of our algorithm. Notice that, since symbols as well as textures can be seen as a set of regions with a particular arrangement, the basis for the recognition is a matching between these regions and those stored in the library of patterns, for symbols, or a matching between neighbouring regions, for textures. The use of string matching techniques offers the advantage that, since a measure of similarity is given, the method can also be applied to disturbed documents such as hand drawn, inaccurately vectorized or documents scanned with an insufficient resolution.

Graph matching methods are often proposed to solve the problem of recognizing symbols in diagrams (e.g. [6][9][11]). The problem consists in finding a model graph that represents the symbol as a subgraph of an input graph representing the diagram. To deal with the presence of noise and distortion, a usual problem in computer vision, inexact or error-tolerant graph matching techniques are proposed [5]. In this work, symbols are recognized in terms of an error-tolerant graph matching that computes the minimum distance from a model RAG, representing the symbol to recognize, to an input RAG, representing the document. This distance measure is considered to be the weighted sum of the costs of edit operations (insertion, deletion and substitution of RAG nodes and edges) to transform one graph into the other. Since RAG nodes are represented as strings, the RAG edit operations must be defined in terms of string edit distance. The texture discrimination problem consists in grouping the neighbouring regions of the input RAG according to their similarity. This is computed in terms of a weighted sum of the string edit distance between their boundary strings and the difference of their areas.

The remainder of this paper is organized as follows: Section 2 describes the graph-based structure used to represent our documents, and summarizes the string matching techniques as the basis for the further described algorithms. In Section 3 an error-tolerant subgraph isomorphism for symbol recognition is described. Section 4 describes a hierarchical clustering algorithm designed to extract structural textures in plans. Experimental results are presented in Section 5. Finally, Section 6 is devoted to the conclusion.

2 Definitions and notation

2.1 From raster images to RAGs

In this work, after a vectorization step, a line drawing is represented by an *attributed graph*. An *attributed graph* G is defined as a 4-pla (V, E, L_V, L_E) where V is the set of nodes; $E \subseteq V \times V$ is the set of edges; L_V and L_E are two labeling functions defined as $L_V : V \to \Sigma \times A^k$ and $L_E : E \to \Sigma \times A^l$, where Σ is a set of symbolic labels and A is a set of attributes. A graph $G'(V', E', L_{V'}, L_{E'})$ is a *subgraph* of G (denoted by $G' \subseteq G$) if $V' \subseteq V$ and $E' \subseteq E$. Matching between two graphs G and G' by a *graph isomorphism* means a bijective mapping $f : V \to V'$ such that the structure of the edges is preserved by the matching function f. An injective mapping $f : V \to V'$ is a *subgraph isomorphism* if there is a subgraph $G'' \subseteq G'$ such that f is a graph isomorphism from G to G''. In this paper we use, for the sake of simplicity, $G(V, E)$ instead of $G(V, E, L_V, L_E)$.

As it has been discussed above, since symbols and textures can be seen as a set of regions with a particular arrangement, in both cases their recognition can be made on the basis of a procedure that computes region similarities. Thus, given a graph $G(V, E)$, their minimal closed loops are extracted using the Jiang and Bunke's algorithm [7]. As can be seen below, every graph region is represented by an attributed cyclic string. Afterwards, an RAG (*Region Adjacency Graph*) $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$ which describes the regions of G and their adjacency relations is constructed. Thus, $\mathcal{V}$ is the set of nodes which correspond to regions in G and $\mathcal{E}$ is the set of edges which represent the region adjacencies. $\mathcal{L}_\mathcal{V}$ and $\mathcal{L}_\mathcal{E}$ are two labeling functions defined as $\mathcal{L}_\mathcal{V} : \mathcal{V} \to E^*$ and $\mathcal{L}_\mathcal{E} : \mathcal{E} \to E^*$, where $\mathcal{L}_\mathcal{V}$ returns the string representing a region and $\mathcal{L}_\mathcal{E}$ returns the string shared by two neighbouring regions. In this paper, analogously to plain graphs, RAG notation is simplified as $\mathcal{G}(\mathcal{V}, \mathcal{E})$ instead of $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{L}_\mathcal{V}, \mathcal{L}_\mathcal{E})$.

2.2 Cyclic string matching: the basis of our algorithms

String matching is a widespread technique used in the recognition of 2D shapes represented by their boundaries [4][12][14]. The basic idea is to represent the boundaries of the input and the prototype shapes as sequences of symbols and use an approximate string matching algorithm to compute their similarity. The well-known algorithm of Wagner and Fischer [15] computes it in polynomial time and space. Let $G(V, E)$ and $\mathcal{G}(\mathcal{V}, \mathcal{E})$ be a graph representing a document and its corresponding RAG, respectively. A graph region $\omega \in \mathcal{V}$ is represented by an attributed string, i.e. $\mathcal{L}_\mathcal{V}(\omega) = e_1 \ldots e_n$, where $e_i \in E$ are the graph edges which outline the region ω. Each symbol of the string is attributed by its length and orientation. The distance between two regions ω and ω', represented by the strings X and Y respectively, is computed in terms of the elementary edit operations required to transform X into Y with minimum cost. Conventionally, three edit operations are defined:

1. *substitution* of a symbol x in X by a symbol y in Y, denoted as $x \to y$;

2. *insertion* of a symbol x in Y, denoted as $\lambda \rightarrow x$;
3. *deletion* of a symbol x in X, denoted as $x \rightarrow \lambda$;

where λ denotes the empty string. An *edit sequence* S is defined as an ordered sequence of edit operations $s_1, \ldots, s_p$. Let γ be a cost function that assigns a non-negative real number $\gamma(s)$ to each edit operation s. The cost of an edit sequence S is defined as $\gamma(S) = \sum_{i=1}^{p} \gamma(s_i)$. The *edit distance* between the strings X and Y is defined by $d(X,Y) = min\{\gamma(S) : S$ is a sequence of edit operations transforming X into $Y\}$.

Two considerations must be made. First, given the unsuitability to define the starting point of a closed boundary, our strings are considered to be cyclic, i.e. strings representing all possible cyclic shifts of their symbols. The edit distance between two cyclic strings can also be computed in polynomial time and space by the algorithm proposed by Maes in [12]. Second, shape boundaries in images may contain some distortions and the three basic edit operations may not be enough to efficiently compute the similarity between two shapes. To overcome this problem Tsai and Yu [13] extended the three basic edit operations by a *merging operation* which allows a whole sequence of symbols to be substituted by another.

In this work, given two graph regions ω and ω' their similarity is computed by the cyclic string edit distance between their boundary strings $\mathcal{L}_{\mathcal{V}}(\omega)$ and $\mathcal{L}_{\mathcal{V}}(\omega')$. The costs of the edit operations are defined as a weighted sum of an angle cost and a length cost, analogously to [14].

3 Symbol recognition

Two major drawbacks arise when symbols are to be recognized: first, the presence of noise and distortion in the document, and second, the segmentation of symbols, i.e. selection of regions of interest. The former requires an error model underlying the matching process. The latter depends on the availability of an *a priori* knowledge about the nature of the document: in circuit diagrams, symbols are large entities connected by lines, in music notation symbols have a uniform background. However, in plans, symbols appear embedded in the diagram and their recognition is the unique way to segment them. To overcome the problem of distorted documents, an error-tolerant subgraph isomorphism algorithm between RAGs has been designed. Moreover, subgraph isomorphism is a high time-consuming procedure, that belongs to the class of NP-complete problems. In architectural drawings, the combinatorial component is especially noticeable due to the difficulty in extracting regions of interest. With the aim to speed up the matching we propose an heuristic criterion in which the neighbourhood between regions guides the order of the matching.

3.1 An error-tolerant subgraph isomorphism for symbol recognition

Given a model graph G_M and an input graph G_I our goal is to find those subgraphs of G_I close to G_M under a similarity function. The use of RAGs instead

of plain graphs allows the similarity between two graphs to be computed in terms of the global distortion of graph regions rather than the local distortion of edges and nodes. Our algorithm computes an error-tolerant subgraph isomorphism between a model RAG and an input RAG. It requires the definition of a distance function between two RAGs. This distance function is considered to be the weighted sum of the costs of edit operations (insertion, deletion and substitution of nodes and edges) to transform one RAG to the other. Actually, in this work, this set of operations is reduced to node substitution and node shift which are described below. Since a subgraph isomorphism means a mapping from the model nodes to a subset of input nodes, the remainder input nodes are implicitly inserted to transform the model graph into the input graph. For this reason, node insertion has no cost associated. Concerning to the node deletion operation, removing a whole region is supposed to be an excessive distortion and hence, it is not considered. Thus, given a model RAG $\mathcal{G}_M(\mathcal{V}_M, \mathcal{E}_M)$ and an input RAG $\mathcal{G}_I(\mathcal{V}_I, \mathcal{E}_I)$, we define the following edit operations to transform $\mathcal{G}_M$ into $\mathcal{G}_I$:

1. *Substitution* of a node $\omega_M^i \in \mathcal{V}_M$ by a node $\omega_I^j \in \mathcal{V}_I$, denoted by $\omega_M^i \rightarrow \omega_I^j$. The cost for this operation is computed in terms of the string edit distance between the corresponding region strings, i.e.,

$$\gamma(\omega_M^i \rightarrow \omega_I^j) = d(\mathcal{L}_\mathcal{V}(\omega_M^i), \mathcal{L}_\mathcal{V}(\omega_I^j)).$$

2. *Shift* of a node $\omega_I^j \in \mathcal{V}_I$, which is adjacent to a node $\omega_I^l \in \mathcal{V}_I$, along the boundary of ω_I^l; denoted as $\omega_I^l \circ \omega_I^j[X, Y]$, where X is the substring shared by ω_I^j and ω_I^l, and Y is the substring shared by ω_I^j and ω_I^l after being ω_I^j shifted. The cost for this operation is computed in terms of the overlapping percentage between X and Y, i.e.,

$$\gamma(\omega_I^j \circ \omega_I^l[X, Y]) = 1 - l_{XY}/max(l_X, l_Y),$$

where l_X and l_Y are the lengths of X and Y, respectively, and l_{XY} is the length of the common substring of X and Y. Shift is a joint operation for edge substitution, insertion and deletion which allows the adjacency relationships between graph regions to be preserved.

Using these edit operations, an error-tolerant subgraph isomorphism between $\mathcal{G}_M$ and $\mathcal{G}_I$ is computed in terms of the minimum cost edit sequence transforming $\mathcal{G}_M$ into $\mathcal{G}_I$. Our algorithm is a state-space search based on an A^* algorithm. It expands a search tree such that each state in the tree represents a partial match from a subset of $\mathcal{G}_M$ nodes to a subset of $\mathcal{G}_I$ nodes. The generation of successor states is guided by the cost of edit operations, i.e. the state with the minimum cost is expanded at each step by mapping a new model region to every input region not yet used in this partial match. The cost of each mapping is a sum of the cost of substituting a model region by an input region and shifting this input region. In this state expansion only the neighbouring regions of the matched regions are considered to be candidates. A partial match is represented by a

5-pla $(R_M, R_I, ct, S, \mathcal{S})$ where R_M and R_I are the regions obtained by merging the matched model regions and the matched input regions, respectively, S is the string edit sequence which transforms the boundary string of R_M into the boundary string of R_I, ct is the cost of this partial match, and $\mathcal{S}$ is the edit sequence transforming the model RAG into the input RAG.

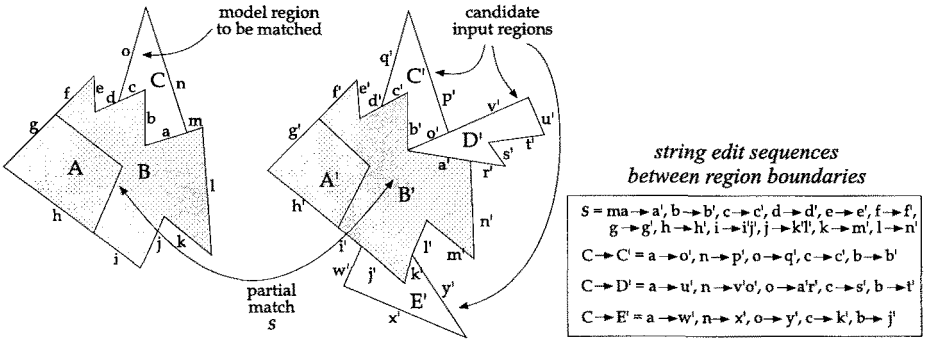

Fig. 1. Shift operation in a partial matching.

Figure 1 shows an example of an intermediate state of the search tree corresponding to the partial match $(AB, A'B', ct, S, (A \rightarrow A', B \rightarrow B'))$. The shadowed regions represent the already matched regions. At this point, there are three possible successors depending on whether the selected operation is $C \rightarrow C'$, $C \rightarrow D'$ or $C \rightarrow E'$. Each of these three successors would have a similar cost if the substitution cost was just considered (clearly, $\gamma(C \rightarrow C') \simeq \gamma(C \rightarrow D') \simeq \gamma(C \rightarrow E')$). This might result in a wrong match. The shift operation adds structure-preserving information which prevents us from these conflictive cases. Let us further analyse the case $C \rightarrow C'$. C is adjacent to AB by the substring abc, and C' is adjacent to $A'B'$ by the substring $b'c'$. Moreover, according to the edit sequence S, abc is mapped to $a'b'c'$, i.e. the region C' should be adjacent to $B'C'$ by the substring $a'b'c'$ to best preserve the adjacency relationship between C and AB. Therefore, to transform the model graph into the input graph, a shift must be applied to the region C when it is substituted by C'. According to our notation, it is denoted by $A'B' \circ C'[b'c', a'b'c']$. As stated above, $\gamma(A'B' \circ C'[b'c', a'b'c'])$ is computed in terms of the overlapping percentage between $b'c'$ and $a'b'c'$. Analogously, $\gamma(A'B' \circ D'[a', a'b'c'])$ and $\gamma(A'B' \circ E'[j'k', a'b'c'])$ are computed, and the three successor states generated.

Figure 1 also illustrates a second problem. Although the successor guided by the operation $C \rightarrow D'$ is clearly a mismatch due to the wrong orientation of D', the shift cost $\gamma(A'B' \circ [a', a'b'c'])$ is likely to be low. This suggests that, to guarantee an adjacency-preserving constraint, the shift cost should also be computed in the reverse order, i.e. $\gamma(D' \circ A'B'[a', s't'u'])$. In summary, the cost in a node expansion is computed in terms of a sum of the cost of substitution and the maximum of the two shift costs.

3.2 Growing strings as an heuristic criterion to prune the search space

In an A^*-based algorithm the number of states in the search space grows exponentially in the worst case. Thus, some works propose to complete the cost of the partial matching in a certain state with the result of a lookahead procedure which represents an heuristic estimation of the future cost. This carries out a reduction in the number of the expanded states. Let $(R_M, R_I, ct, S, \mathcal{S})$ be a partial match. We define a simple lookahead criterion which can be stated as follows: given a model region $\omega_M^i \in \mathcal{V}_M$ adjacent to R_M by a substring X, the unique neighbouring regions of R_I that will be considered as candidates are those that have boundary substrings that appear in S as mappings of X. For example, in Fig. 1, $X = abc$ is mapped in S to $a'b'c'$, thus only C' and D' are considered as candidates and E' is rejected. This criterion brings forward the adjacency constraint consideration modeled as the shift operation. The overall idea is that after an initial model region is matched to an input region, the boundary strings of these regions grow in terms of the similarity of their neighbourhood.

An illustrative example of matching two graphs by growing one of their regions is shown in Fig. 2. In this figure we can follow the progress of the mapping from model regions to input regions until the subgraph isomorphism is completed. The matching starts by mapping the model region A to the input region A'. In the next level of the search tree one of the neighbouring regions of model region A is mapped to one of the neighbouring regions of the input region A'. In the example, the mapping $B \rightarrow B'$ is selected. Then, the new mapped regions are integrated in the partial matching, that is, regions A and B are merged (denoted by AB), i.e. their boundary strings are merged into a single boundary string which outlines the region AB, and analogously the regions A' and B' are merged into a region $A'B'$. In Fig. 2, thicker lines represent the boundary strings of the partial matchings at each level. New model-to-input region mappings are gradually incorporated into partial matchings until the subgraph isomorphism is attained.

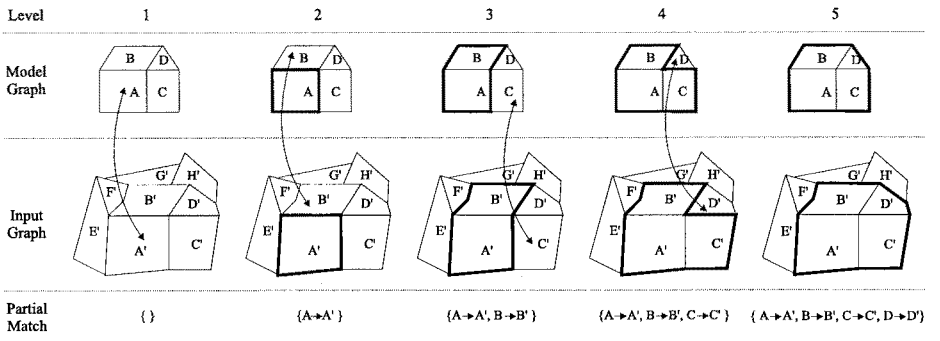

Fig. 2. Example of string growing in terms of the neighbouring region similarity.

4 Textured-region segmentation

Graphic documents, and plans in particular, usually contain structured textures with a regular repetition, i.e. textures where an element, called texel, is regularly placed according to certain rules. Finding these textured zones and analyzing their structure would be an important step into the global understanding of the document. Since symbol recognition is a high time-consuming process, an efficient extraction of the segments defining the texture would drastically reduce the information and focus further interpretation on the remainder lines. Furthermore an efficient recognition of the structure allows the texture to be stored in a more compact way.

Two types of texels are addressed in this work: straight segments and polygonal shapes. According to the placement rules, we can distinguish two types of textures: those formed by adjacent polygonal shapes, and those formed by isolated texels, placed following a tessellated structure. The first type of textures are discriminated by grouping similar adjacent polygonal shapes. Their similarity is determined according to their areas and shapes. The second type of textures requires a previous step where a neighbourhood relation between texels is established. Voronoi Polygons (VP) [2] are a useful tool to partition the plane in terms of a nearest-neighbour criteria between its elements. VP are computed from isolated texels. With the VP computation, a texture of the second type becomes a texture of the first type, so the same clustering method can be applied. Figure 3(a) shows an example of the two kinds of textures.

The global process is applied to the RAGs representing: the input image $\mathcal{G}(\mathcal{V}, \mathcal{E})$, in order to find textures formed by adjacent polygonals, as in Fig. 3(b), and the Voronoi tessellation computed from it, $\mathcal{G}_{VP}(\mathcal{V}_{VP}, \mathcal{E}_{VP})$, in order to find textures formed by isolated segments as in Fig. 3(d). Given an RAG, the process groups its similar neighbouring regions according to their area and shape. When the regions have been grouped, the textured zones are obtained. The clustering algorithm applied for grouping the regions is a modification of the algorithm described in [10]. Both use an irregular pyramid to build a hierarchical clustering. However, our algorithm is applied to an RAG instead of a grey image as done in [10]. Furthermore, a different distance function dt has been defined as:

$$dt(P, Q) = w_1 d(\mathcal{L}_{\mathcal{V}}(P), \mathcal{L}_{\mathcal{V}}(Q)) + w_2 d_a(P, Q),$$

where d_a is the difference between P's area and Q's area, d is the cyclic string edit distance from P to Q, and w_1 and w_2 are two weighting constants experimentally set. First of all, the algorithm computes the area of each region of $\mathcal{G}(\mathcal{V}, \mathcal{E})$ and the string edit distance between each pair of neighbouring regions. A graph whose vertices represent clusters, and whose edges represent neighbourhoods is constructed. Initially, each region is a cluster. A random weight is associated to each cluster, and the clusters with the highest local weight are marked as survivors. Each non-survivor cluster is associated to the closest neighbouring survivor according to the distance dt. If this distance exceeds a certain threshold, the cluster becomes a survivor. Each survivor becomes a new vertex in the new

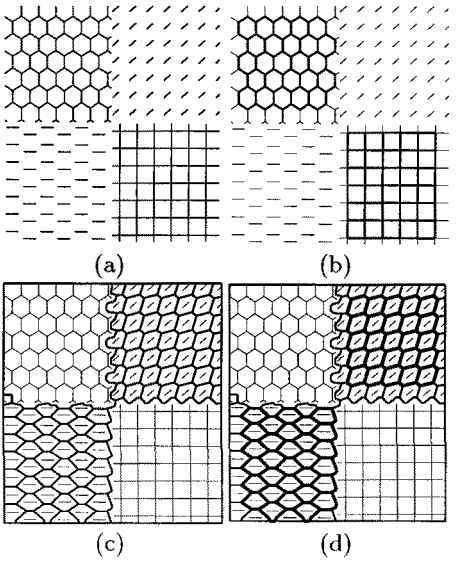

Fig. 3. (a)Vectorized image. (b) Polygonal textured zone. (c) Voronoi polygons from the vectorized original image. (d) Clusters from the Voronoi tesselation.

level, and the area and the distance between the other clusters is computed as the mean of the clusters that are children of the previous level. The process iterates until the number of clusters in one level is the same as in the previous one. The regions of the $\mathcal{G}(\mathcal{V}, \mathcal{E})$, that are children of clusters with a high number of descendants, define the textured areas.

5 Results and discussion

The described algorithms have been applied to a set of fifteen documents of plans with different instances of symbols and textures and different rates of distortion. Every instance was drawn in a letter-sized sheet and was scanned in a range between 150 and 300 *dpi*.

In Fig. 4, after a vectorization of the original image (Fig. 4(a)) the matching algorithm proposed in section 3 has been applied. The recognized symbols are displayed with bolded lines in Fig. 4(b), on the graph approximation of the input image. The value written on each symbol is the dissimilarity degree. Fig. 4(c) shows the textured regions found in the input graph. In the borders of a textured area, the texels could be cut off and then they would be grouped in a separate cluster. This problem can be solved by a post-clustering procedure with a more tolerant dissimilarity measure between clustered texels. Finally, the symbol recognition procedure has been applied to the input graph after removing the edges belonging to textured regions (Fig. 4(d)). This drastically reduces the computation time.

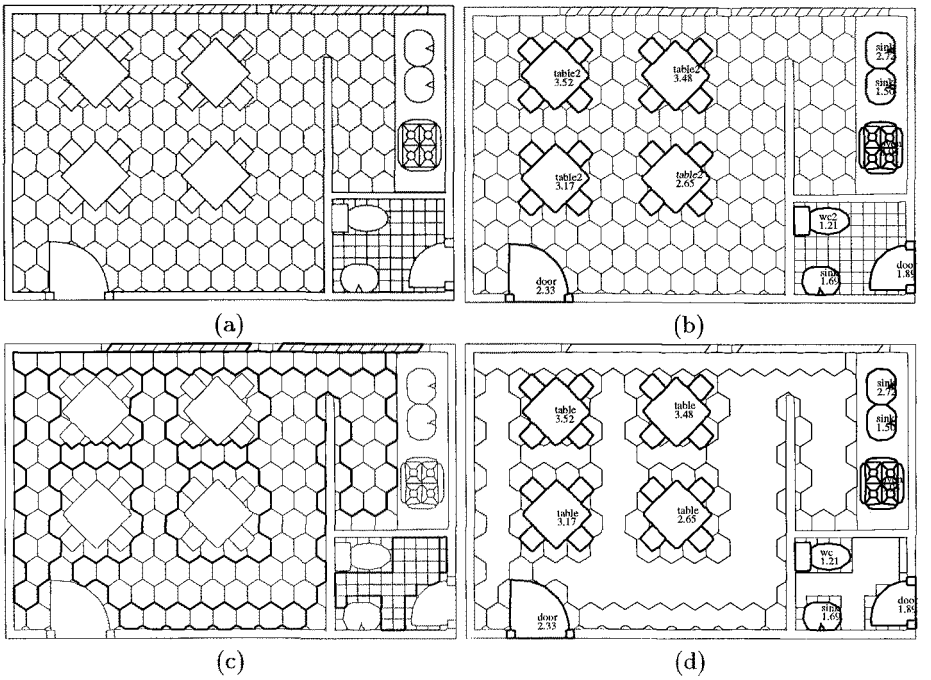

Fig. 4. (a) Original image, (b) Recognized symbols, (c) Textured regions, (d) Symbol recognition after removing the textured regions from the input graph.

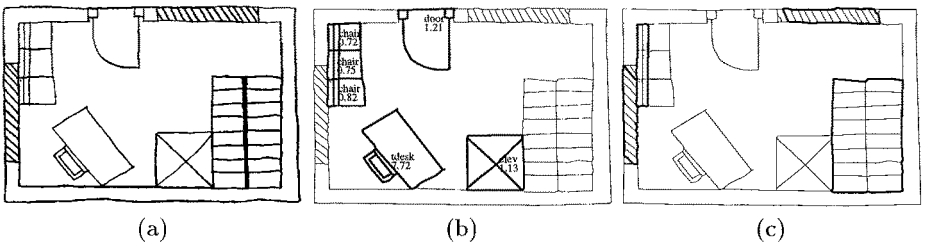

Fig. 5. (a)(b)(c) Results in a hand drawn image: (a) Original image. (b) Recognized symbols. (c) Textured regions.

Figure 5 illustrates the reliability of the method even in presence of distortion when it is applied to a hand drawn input document (Fig. 5(a)). Figure 5(b) shows three instances of the symbol *chair* with increasing degrees of distortion, as could be expected at first glance. This symbol illustrates in Fig. 5(c) a conflict which sometimes may appear: some regularly placed regions of the symbol have been discriminated as a texture. This suggests that, though the texture and symbol procedures can operate in parallel, a dialog should be established between them to solve these ambiguities.

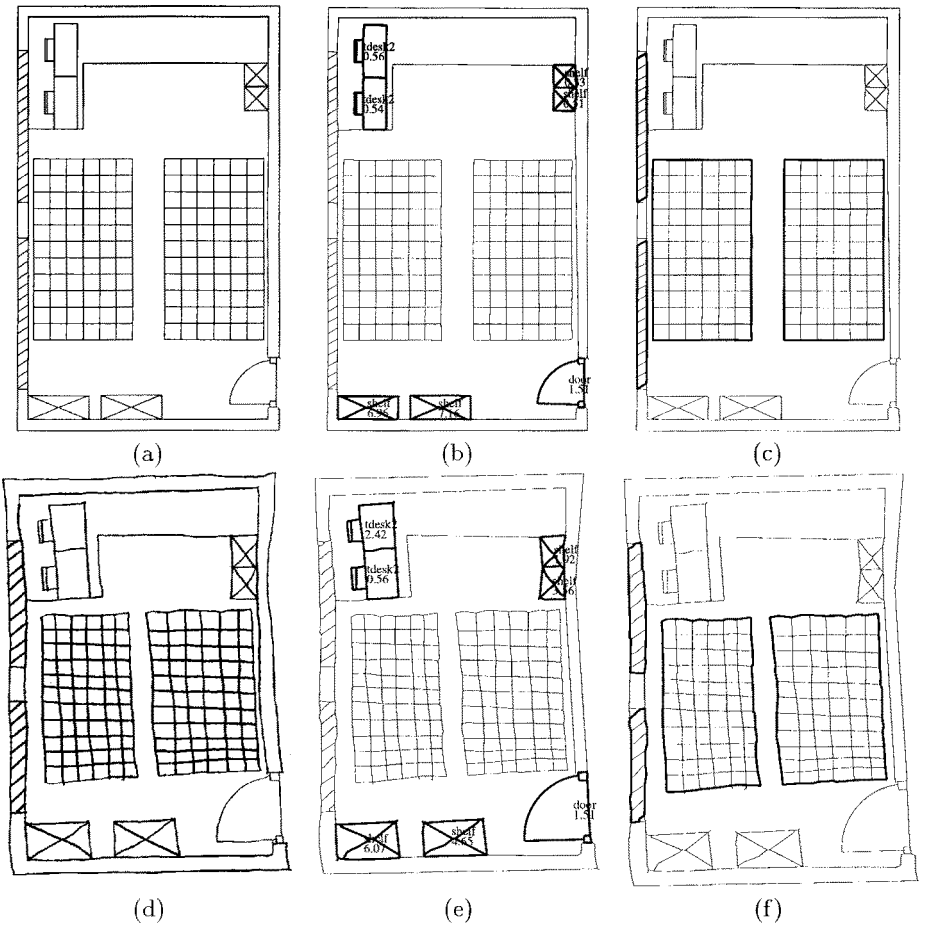

Fig. 6. Results in two different versions of the same document with different distortion rates. (a)(b)(c) Printed document and (d)(e)(f) hand-drawn document.

Figures 6(a) and 6(d) represent a computer-drawn and a hand-drawn instances, respectively, of the same document. In both cases, symbols (Figs. 6(b)

and 6(e)) and textures (Figs. 6(c) and 6(f)) have been recognized. This example illustrates that local distortions due to hand input do not result in a very increased degree of dissimilarity, but this is rather a global measure, to the extent of having a higher value in Fig. 6(b) than in Fig. 6(e) for the bottom instances of the symbol *shelf*.

6 Conclusion

An interpretation method for architectural floor plans has been proposed. The input document is represented in terms of its closed loop structure by an RAG. Attributed strings are used to represent the boundaries of the regions (RAG nodes). Using this representation, the similarity between two regions can be computed in terms of string matching procedures. This is the basis to develop two recognition goals in a document: building elements, by graph matching methods, and structural textures, by clustering methods. The use of attributed strings to represent the regions of a document offers two main advantages: the algorithms developed for symbol recognition and texture discrimination have a common basis, and the method also works for disturbed documents as string edit distance is an error-tolerant measure.

References

1. D. Antoine, S. Collin, and K. Tombre. Analysis of technical documents: The redraw system. In H.S. Baird, H. Bunke, and K. Yamamoto, editors, *Structured document image analysis*, pages 385–402. Springer Verlag, 1992.
2. F. Aurenhammer. Voronoi diagrams- a survey of a fundamental geometric data structure. *ACM Comput. Surveys*, 23(3):345–405, 1991.
3. L. Boatto et al. An interpretation system for land register maps. *Computer*, 25(7):25–33, July 1992.
4. H. Bunke and U. Buhler. Applications of approximate string matching to 2d shape recognition. *Pattern Recognition*, 26(12):1797–1812, 1993.
5. H. Bunke and B.T. Messmer. Efficient attributed graph matching and its application to image analysis. In C. Braccini, L. DeFloriani, and G. Vernazza, editors, *Proc. of 8th ICIAP, San Remo, Italy*, pages 45–55. Volume 974 of LNCS, Aug 1995.
6. A.H. Habacha. Structural recognition of disturbed symbols using discrete relaxation. In *1st. ICDAR*, pages 170–178, Sep-Oct 1991. Saint Malo, France.
7. X.Y. Jiang and H. Bunke. An optimal algorithm for extracting the regions of a plane graph. *Pattern Recognition Letters*, (14):553–558, 1993.
8. R. Kasturi, S.T. Bow, W. El-Masri, J. Shah, Gattiker J.R., and Mokate U.B. A system for interpretation of line drawings. *IEEE Trans on PAMI*, 12(10):978–992, Oct 1990.
9. P. Kuner and B. Ueberreiter. Pattern recognition by graph matching. Combinatorial versus continuous optimization. *IJPRAI*, 2(3):527–542, Sep 1988.
10. Stephen W.C. Lam and Horace H.S. Ip. Structural texture segmentation using irregular pyramid. *Pattern Recognition Letters*, pages 691–698, July 1994.

11. S.W. Lee, J.H. Kim, and F.C.A. Groen. Translation-, rotation-, and scale-invariant recognition of hand-drawn symbols in schematic diagrams. *IJPRAI*, 4(1):1–25, 1990.
12. M. Maes. Polygonal shape recognition using string-matching techniques. *Pattern Recognition*, 24(5):433–440, 1991.
13. W.H. Tsai and S.S. Yu. Attributed string matching with merging for shape recognition. In *7th. ICPR*, pages 1162–1164, 1984. Montreal, Canada.
14. Y.T. Tsay and W.H. Tsai. Model-guided attributed string matching by split-and-merge for shape recognition. *IJPRAI*, 3(2):159–179, 1989.
15. R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21(1):168–173, 1974.