Hierarchical Plausibility-Graphs for Symbol Spotting in Graphical Documents

Klaus Broelemann
Dept. of Maths. and Comp. Sc.
University of Münster
Münster, Germany
broele@uni-muenster.de

Anjan Dutta
Computer Vision Center
Univ. Autònoma de Barcelona
Barcelona, Spain
adutta@cvc.uab.es

Xiaoyi Jiang
Dept. of Maths. and Comp. Sc.
University of Münster
Münster, Germany
xjiang@uni-muenster.de

Josep Lladós
Computer Vision Center
Univ. Autònoma de Barcelona
Barcelona, Spain
josep@cvc.uab.es

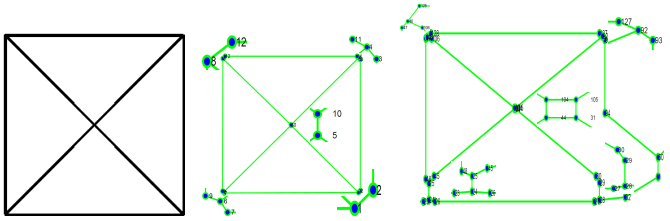
***Abstract*—Graph representation of graphical documents often suffers from noise viz. spurious nodes and spurios edges of graph and their discontinuity etc. In general these errors occur during the low-level image processing viz. binarization, skeletonization, vectorization etc. Hierarchical graph representation is a nice and efficient way to solve this kind of problem by hierarchically merging node-node and node-edge depending on the distance. But the creation of hierarchical graph representing the graphical information often uses hard thresholds on the distance to create the hierarchical nodes (next state) of the lower nodes (or states) of a graph. As a result the representation often loses useful information. This paper introduces plausibilities to the nodes of hierarchical graph as a function of distance and proposes a modified algorithm for matching subgraphs of the hierarchical graphs. The plausibility-annotated nodes help to improve the performance of the matching algorithm on two hierarchical structures. To show the potential of this approach, we conduct an experiment with the SESYD dataset.**

I. Introduction

Graphs are an efficient way of representing graphical documents specially for line drawings. Since documents often suffer from noise, the representation with graphs also results in distorted graphs, for example with spurious nodes, spurious edges and discontinuity in them (see Fig. 1). Hierarchical graph representation is a way of solving this kind of structural errors hierarchically where the node-node and node-edge are merged hierarchically depending on the node-node or node-edge distance [1].

The main motivation of the work comes from [2], where the authors used the hierarchical representation of the segmented image regions and later used an approximated maximal clique finding algorithm on the association graph of the two hierarchical graphs to match the two hierarchical representations [3], [4]. In particular, the aforementioned work was applied to match two different natural images for classification.

The construction of graph representation of documents is followed by some inter-dependent pre-processing steps viz. binarization, skeletonization, polygonal approximation. These low level pre-processing steps result in the vectorized documents which often contain some structural errors. In this work our graph representation considers the critical points as the nodes and the lines joining them as the edges. So often the graph representation contains spurious nodes, edges, disconnection between nodes etc (see Fig. 1). Our present work is an extension of the work in [1] where we dealt



(a) (b) (c)

Fig. 1: Examples of the structural distortions (spurious nodes, edges, discontinuous edges) for a graphical symbol: (a) A graphical symbol called *table1*, (b), (c) Graph representations of two different instances of the symbol *table1* when appeared in floorplans, these instances are cropped from bigger graphs representing floorplans. Graph representation of documents involves low level image processing viz. binarization, skeletonization, vectorization etc. which further add structural noise such as spurious nodes, edges etc. The example shows how even a undistorted symbol can become distorted after represented with graph (note the spurious nodes and edges near the junction and corners.).

with this kind of distortions in the graph level, to do that we proposed hierarchical representation of graphs because the hierarchical representation of graphs allows to incorporate the various segmentation errors hierarchically. But the node-node or node-edge merging was performed depending on a hard threshold which resulted in some loss of information. In this work we are assigning plausibilities to the nodes as a function of the distance and use them for matching. The work is still in progress and the results show some methodological improvement.

The rest of the paper is organized into four sections. In Section II we present the methodology of sub graph matching. Section III contains the detailed experimental results. After that, in Section IV, we conclude the paper and discuss future work.

II. Methodology

An essential part for graph-based symbol spotting methods is the representation of symbols. This representation often contains low-level vectorization errors that will affect later

graph matching methods. In this section we present a hierarchical representation that overcomes these problems by covering different possible vectorizations and estimating their plausibilities.

First we will give a brief overview of the initial vectorization and some errors that can occur due to it. Afterwards we will describe our hierarchical representation and how this representation overcomes the vectorization errors.

A. Vectorization

Graph representation of documents follows some preprocessing steps, vectorization is one of them. Here vectorization can be defined as approximating the binary images to a polygonal representation. In our method we have used the Rosin-West algorithm [5] which is implemented in the Qgar package[1]. This algorithm works without any parameter except one to prune the isolated components. The algorithm produces a set of critical points and the information whether they are connected. Our graph representation considers the critical point as the nodes and the lines joining them as the edges.

Vectorization errors: The resulting graph can contain vectorization errors. Reasons for that can be inaccurate drawings, artifacts in the binarization or errors in the vectorization algorithm. There are different kinds of vectorization errors that can occur. Among these, we concentrated on the following ones:

Gaps In the drawing there can be small gaps between lines that ought to be connected. Reasons for that can be inaccurate drawings as well as mistakes in the binarization. The result can either be two unconnected nodes at the border of the gap or a node on one and an edge on the other side of the gap. Beside caused by errors, gaps can also be drawn intentionally to separate nearby symbols.

Split nodes On the other hand, one original node can be split into two or more nodes. This can happen, if lines in the drawing do not intersect exactly at one point. Another reason are artifacts from the skeletonization step. Nearby nodes that seem to be a split node can be the result of fine details instead of vectorization errors.

Dispensable nodes The vectorization can create nodes of order two that divide a straight edge into two or more parts. One reason for these nodes are small inaccuracies in the drawing that cause a local change in direction. For a later symbol spotting, these nodes are often undesired and should be removed. Nevertheless, in some cases such structures reflect details of the symbol. Examples of these errors can be seen in Fig. 1.

Though all these errors can be corrected in a post-processing step, a simple post-processing causes other problems: often it is not clear for the system whether a situation is an error or intentional. To deal with this uncertainty, we introduce a hierarchical representation that will be described in the next part.

[1]http://www.qgar.org/

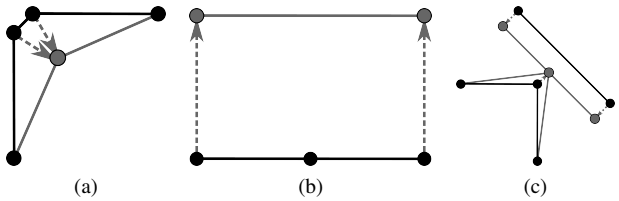


Fig. 2: Three cases for simplification. Displayed are the original nodes and edges (black) and the simplified nodes and their edges (gray): (a) Merge nodes (b) Remove dispensable node (c) Merge node and edge.

B. Hierarchical graph construction

This section describes the construction of hierarchical graph that is able to cover different possible vectorizations. This enables a later graph matching algorithm to deal with the uncertainties whether a part of the graph is intentional drawn or caused by a vectorization error.

The basic idea of our approach is to extend a given graph G so that it contains the different possibilities of interpretation. These possibilities are connected hierarchically and assigned with a plausibility measure. The hierarchy allows us to embed the constraint just to match one interpretation into the graph matching. In section II-C we will give further details for the graph matching, the hierarchical constraint and how to use plausibilities for the matching.

In order to create different possible vectorizations, we take the initial vectorization represented in G and simplify it step by step. For this purpose, we identify three cases that allow a simplification. These three cases will be motivated in the following. Afterwards the plausibilites are introduced and based on this a formal definition of our simplification steps is given.

Nearby nodes. Both gaps in drawing as well as split nodes result in nodes near to each other and can be solved by merging these nodes. Since nearby nodes can also be the result of correct vectorization, e.g. due to two nearby symbols, we store both versions and hierarchically connect the merged node with the basic nodes. The merged node inherits all connection of its basic nodes. Fig. 2 (a) shows an example for such a merging step.

Dispensable nodes. In case of dispensable nodes, the vectorization error can be solved be removing the node. Again, a hierarchical structure can store both versions. As described before we only consider dispensable nodes that have two neighbors. The simplified versions of these neighbors are directly connected. This is shown in Fig. 2 (b). Applying this rule multiple times allows us to remove chains of dispensable nodes.

Nodes near to edges. The third simplification is the merging of nodes with nearby edges. In this way the second kind of gaps can be corrected. To merge a node with an edge, the edge has to be divided into two edges by a copy of the node. This can be seen for an example in Fig. 2 (c).

1) Plausibility: The major novelty of this paper is the use of plausibilities. The aim of these plausibilities is to measure

the likelihood of a certain simplification to be correct. By doing so, we can prioritize matching of likely structures and still keep the ability of matching unlikely ones. To compute the plausibility for a certain simplification we identify basic features and describe the plausibility as function of these features. The features are described in the following:

Merging nodes. The plausibility for merging very near nodes is high and it decreases with increasing distance between the nodes. Thus, the distance between the nodes is taken as feature to measure the plausibility.

Removing nodes. Removing a node means to merge two edges. We consider the removal as plausible if the resulting edge is similar to the two original edges. There different features that can be used to measure this similarity. One possibility is the angle between bot edges. If the angle is near to 180, the resulting edge will be near to the original edges. Another measurement is the distance of the node to the resulting edge, either absolute or relative to the length of the edge. For our experiments we used the angle feature.

Merging nodes with edges Similar to merging nodes, we take the distance of the edge to the node as feature for the plausibility.

To measure the plausibility for the three previously mentioned cases, we define three functions

1) function $\delta_1 : V \times V \to \mathbb{R}$ to measure the plausibility for merging two nodes.
2) function $\delta_2 : V \to \mathbb{R}$ to measure the plausibility for removing a node.
3) function $\delta_3 : V \times E \to \mathbb{R}$ to measure the plausibility for merging a node with an edge.

For the concrete implementation we used exponential functions applied to the features, e.g.

$$\delta_1(u, v) = \alpha_1 \exp(-\beta_1 \cdot d(u, v))$$

The functions δ_2 and δ_3 are defined analogous, replacing $d(u, v)$ by the respective features.

Our approach also allows other plausibility measurements. Note that our previous work [1] without plausibilities can be seen as a special case of this work by choosing binary measurements, i.e.

$$\delta_1(u, v) = \begin{cases} 1 & \text{if } d(u, v) < T_1 \\ 0 & \text{otherwise} \end{cases}$$

The plausibilities are used to identify possible simplifications. For this purpose we define a threshold T_0 and only perform hierarchical simplifications for constellations that have a plausibility greater than a T_0.

2) Recursive definition: Based on the previous motivation we will give a recursive definition of our hierarchical graphs that reflects the construction algorithm based on the vectorization outcome.

The result of the vectorization is an undirected graph $G = (V_G, E_G, \sigma_G)$ where V_G it the set of nodes, $E_G \subseteq V_G \times V_G$ is the set of edges and $\sigma_G : V_G \to \mathbb{R}^2$ is a labeling function that maps the nodes to their coordinates in the plane.

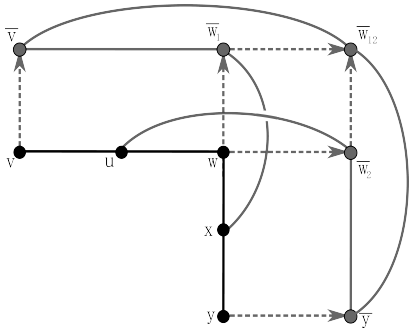


Fig. 3: An example for removing nodes. Note that the possibility of removing two adjacent nodes of w creates four different possible interpretations of w, e.g. $\bar{w}_1$ stands for removing u but keeping x

A hierarchical graph has two kinds of edges: undirected neighborhood edges and directed hierarchical edges. Hierarchical edges represent simplification operations, i.e. they link nodes from the original graph arising from the vectorization to successor nodes representing simplified vectorizations. In addition, each node is assigned with a plausibility value. Formally, we define a hierarchical graph H as a tuple $H = (V, E_N, E_H, \sigma, p)$ with the neighborhood edges $E_N \subseteq V \times V$, the hierarchical edges $E_H \subseteq V \times V$ and plausibility values $p : V \to \mathbb{R}$.

Note that there is a difference between the plausibility of a node (given by the function p) and the plausibility of a simplification (given by δ_i, $i = 1, 2, 3$). The reason for this difference is, that a plausible simplification with unplausible nodes results in unplausible nodes.

Furthermore, given two nodes $u, v \in V$ let $u \rightsquigarrow v$ denote that v is a hierarchical successor of u and $L(u)$ denote the set of all predecessors of u that belong to G: $L(u) = \{v \in V_G | v \rightsquigarrow u\}$. Based on these functions and formulations we can define the hierarchical simplification $H = \mathcal{H}(G) = (V, E_N, E_H, \sigma, p)$ of G by the following rules:

Initial. As initialization for the recursion, G is a subgraph of H. We define a base plausibility $p(v) = 1$ for all initial nodes $v \in V_G$.

Merging. For $u, v \in V$ with $\delta_1(u, v) > T_0$ there is a merged node $w \in V$ with

- w is a hierarchically successor of u and v:
  $\forall s \in V : s \rightsquigarrow w \Leftrightarrow s \rightsquigarrow u \vee s \rightsquigarrow v \vee s \in \{u, v\}$
- w has all neighbors of u and v except u and v:
  $\forall s \in V : (s, w) \in E_N \Leftrightarrow ((s, u) \in E_N \vee (s, v) \in E_N) \wedge s \notin \{u, v\}$
- w lies in the center of its leaf nodes:
  $\sigma(w) = \frac{1}{|L(w)|} \sum_{s \in L(w)} \sigma(s)$
- The plausibility of w is defined by δ_1 and the plausibilities of u and v:
  $p(w) = \delta_1(u, v) p(u) p(v)$
  If there are different ways to create w, we assign the maximal plausibility to w.

Removing. For a dispensable node $u \in V$ with $\delta_2(u) > T_0$ there exist two neighbor nodes $v, w \in V_G$, i.e. $(u, v), (u, w) \in E_N$. Since v and w can have hierarchical successors from other simplifications, these have to be included in the definition: for all $v_i : (v_i, u) \in E_N \wedge v \in L(v_i)$ there exists a $\bar{v_i}$. In the same way a set of $\bar{w_j}$ is defined.

- $\bar{v_i}$ hierarchical successor of v_i: $(v_i, \bar{v_i}), (w_j, \bar{w_j}) \in E_H$
- to cover all possibilities, there is neighborhood connection between all of $\bar{v_i}$ and all $\bar{w_j}$. Furthermore, the $\bar{v_i}$ has the same connections as v_i with exception of the removed node u:
  $(s, \bar{v_i}) \in E_N \Leftrightarrow ((s, v_i) \in E_N \wedge s \neq u) \vee \exists j : s = w_j$.
  (analogous for w_j)
- The coordinates do not change: $\sigma(v_i) = \sigma(\bar{v_i})$, $\sigma(w_j) = \sigma(\bar{w_j})$
- $p(\bar{v_i}) = \delta_2(u)p(v_i)$

In this definition the successors of u and w have to be included. The reason for this can be seen in the example in Fig. 3: if the removing is done iteratively, removing u will lead to $\bar{v}$ and $\bar{w_1}$. A subsequent removal of x has to create $\bar{w_2}$ and $\bar{w_{12}}$ in order to cover both possibilities: just remove x and remove u and x. This will give a plausibility of:

$$p(\bar{w_{12}}) = \delta_2(x)p(\bar{w_1}) = \delta_2(x)\delta_2(u)p(w)$$

Node/Edge merging. For $u \in V, e = (v, w) \in E$ with $\delta_3(u, e) > T_0$ there exist simplifications $\bar{u}, \bar{v}, \bar{w}$ with

- $\bar{u}, \bar{v}, \bar{w}$ are hierarchically above u, v, w:
  $\forall s \in V : s \rightsquigarrow \bar{u} \Leftrightarrow s \rightsquigarrow u \vee s = u$ (analog for v,w)
- $\bar{u}$ intersects the edge between $\bar{v}$ and $\bar{w}$:
  $\forall s \in V : (s, \bar{u}) \in E_N \Leftrightarrow ((s, u) \in E_N \vee s \in \{\bar{v}, \bar{w}\}$
- The coordinates do not change: $\sigma(u) = \sigma(\bar{u})$, $\sigma(v) = \sigma(\bar{v})$ and $\sigma(w) = \sigma(\bar{w})$
- $p(\bar{u}) = \delta_3(u, e)p(u)$ (analog for v,w)

Based on these recursive rules, we construct the smallest hierarchical graph that satisfies these rules, i.e. no additional nodes are added. Here it is to be noted that the hierarchical simplification $\mathcal{H}(G)$ of the graph G always contains the graph G.

C. Graph matching

In this section we will describe how to make use of the hierarchical graph representation described in the previous section for subgraph matching in order to spot symbols for vectorial drawings. Graph matching has a long history in pattern recognition and there exist several algorithms for this problem [6]. Our approach is based on solving maximal weighted clique problem in association graphs [4]. In this section we will first give a brief overview over the graph matching algorithm. This method relies on similarities between nodes. Hence, we will present a geometric node similarity for hierarchical graphs afterwards.

Given two hierarchical graphs $H^i = (V^i, E^i_N, E^i_H, \sigma^i)$, $i = 1, 2$, we construct the association A. Each node of A consists of a pair of nodes of H^1 and H^2, representing the matching between these nodes. Two nodes (u_1, u_2), $(v_1, v_2) \in H_1 \times H_2$ are connected in A, if the matching is consistent with each other. For hierarchical graphs we define the constraints for edges in A: u_i and v_i are different, not hierarchically connected and if u_1 and v_1 are neighbor, this also holds for u_2 and v_2. By forbidding the matching of hierarchically connected nodes, we force the matching algorithm to select one version of the vectorization. The first and the third constraint keep the structure between both subgraphs same.

We use replicator dynamics [4] to find the maximal weighted clique of the association graph and, hence, the best matching subgraphs of H^1 and H^2. Based on the results of this, we perform the following steps to spot symbols. Let us consider H^1 be the query graph or the model graph and H^2 be the input graph where we want to spot the instances of H^1. First of all we perform n iterations and in each iteration we perform the replicator dynamics to find the correspondences of the H^1 to H^2. Since the replicator dynamics only provide a one-to-one matching, in each iteration we obtain the correspondences from the nodes of H^1 to the nodes of H^2. So for m nodes in H^1 we get m nodes in H^2. But it is not constrained that these m nodes in H^2 will belong to the same instance of H^1. So to obtain the different instances of the H^1 we consider each of the m nodes in the H^2 and all the neighborhood nodes of a node which can be reached within a k graph path distance. The graph path distance between two nodes is calculated as the minimum total number of nodes between the two nodes. Let us denote this set of nodes as V^1_s and consider all the hierarchical and normal edges connecting the nodes in V^1_s as in H^1, this forms a subgraph which we can denote as $H^1_s = (V^1_s, E^1_{s_N}, E^1_{s_H}, \sigma^1_s)$. We again apply the replicator dynamics to get the best matching subgraph and compute the bounding box around the nodes of best correspondences. The bounding box gives the best matching region of interest expected to contain instance of a query symbol.

The complexity of replicator dynamics is $\mathcal{O}(|A|^2)$ (see [2]). Since we perform n iterations, we get a complexity of $\mathcal{O}(n \cdot |A|^2)$. In order to reduce the computation time we use the fact that the symbols are much smaller than floorplans. We create overlapping parts of the floorplan and perform the matching between the symbol and the parts. These parts have to be big enough to ensure that the symbols are completely included in at least one part. The the construction of the hierarchical graph takes about 2 or 3 seconds for an average floorplan, the matching takes several minutes.

Node attributes: The graph matching algorithm operates on the association graph with similarity labels for the nodes. To use this algorithm, we have to define the similarity between two nodes of the hierarchical graph. Since the matching reflects geometric structures, we use geometric attributes for the similarity.

In a non-hierarchical planar graph, a single node can be labeled by the sequence of adjacent angles which sum up to $360°$. Fig. 4 (a) gives an example for such a labeling. This naive approach will cause some problems for hierarchical graph since nodes can have several hierarchically connected neighbors. Thus, the number of possible vectorizations has a strong influence on the node description. Because the number

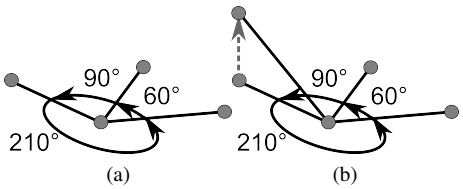


Fig. 4: Example for node labels for graphs based on angles between edges: (a) for planar graphs and (b) for hierarchical graphs. Both will be labeled with $(90, 210, 60)$.

of possibilities is also affected by the level of distortion of the original image, such an approach is not robust to distortion.

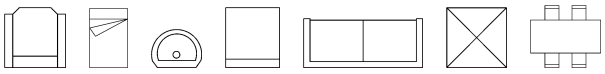


Fig. 5: Model symbols in the SESYD dataset: (a) *armchair*, (b) *bed*, (c) *sink1*, (d) *sofa1*, (e) *sofa2*, (f) *table1*, (g) *table2*.

To reduce the influence of the hierarchical structure and the distortion on the node labeling, we use only edges to nodes that have no predecessor connected with the central node. An example for that can be seen in Fig. 4 (b): though the central node is connected to four nodes, only three edges are used to compute the node label.

To compute the similarity between two node labels, we define an editing distance on these labels. The editing operations are rotating one edge, i.e. lowering one angle and rising another one, removing one edge, i.e. merging two angles, and rotating the whole description. The last operation is cost-free and makes the similarity rotation-invariant. The cost for rotating an edge is set to the angle of rotation. The cost for removing an edge is set to a fixed value.

Using this editing distance, we can define the similarity between nodes. This similarity is based on the nodes and their direct neighborhood, but do not take into account the plausibilities of the nodes. In order to prefer matchings between plausible nodes, we multiply the similarity between two nodes with their plausibilities to get the weight for the corresponding node in the association graph.

III. Experimental Results

Since the work still is in progress, we have conducted a short experiment to check the performance of the algorithm. Our experiments were conducted on the images taken from the SESYD dataset [8][2]. Originally, this dataset contains 10 different subsets and 16 query symbols. These query symbols are available to evaluate our method with the ground truth. Each of the subsets contains 100 synthetically generated floorplans. All the floorplans in a subset are created from the same floorplan template by putting different model symbols in different places in random orientation and scale. For this short experiment we

[2]http://mathieu.delalandre.free.fr/projects/sesyd/symbols/floorplans.html

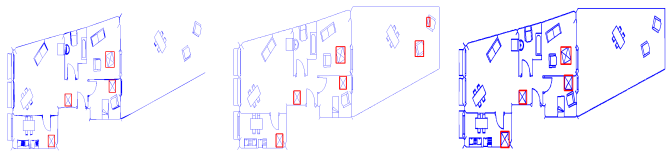


Fig. 6: Results of detecting *table1*, note that all the instances of the symbol *table1* are correctly detected even the ones attached with the walls. In reality these walls are thin and hence less distorted during the vectorization, (b) Results of spotting *table1* by the previous version of the method [1], (c) Results of spotting *table1* by Dutta et al. [7].

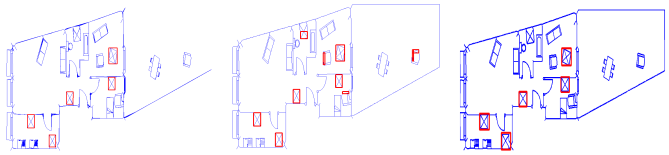


Fig. 7: Results of detecting *table1*, except one all the instances of the symbol *table1* are correctly detected. The one which is not detected is attached with the thick black pixels, (b) Results of spotting *table1* by the previous version of the method [1], (c) Results of spotting *table1* by Dutta et al. [7].

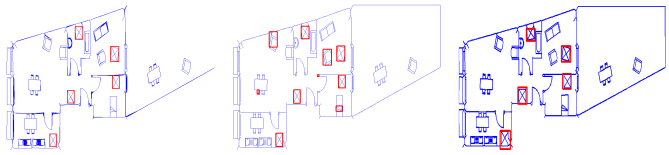


Fig. 8: Results of detecting *table1*, note that all the instances of the symbol *table1* are correctly detected even the one which is connected with thick black pixels, (b) Results of spotting *table1* by the previous version of the method [1], (c) Results of spotting *table1* by Dutta et al. [7].

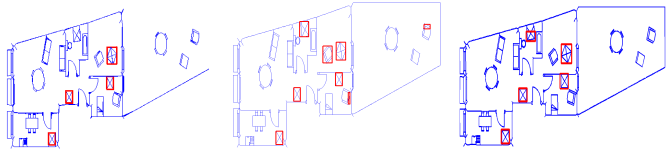
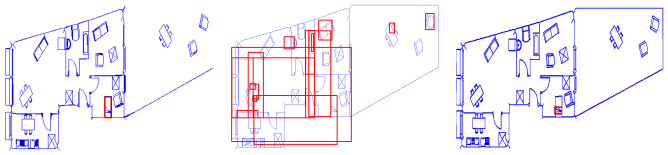


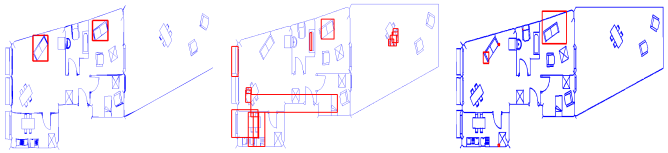
Fig. 9: Results of detecting *table1*, here two of the symbols are not detected and one of them are isolated but heavily distorted by the vectorization algorithm, (b) Results of spotting *table1* by the previous version of the method [1], (c) Results of spotting *table1* by Dutta et al. [7].

have taken the first twenty images from the subset *floorplan16-01* and three model symbols: *bed*, *sofa2* and *table1*. Some



(a) (b) (c)

Fig. 10: Results of detecting *bed*, here the single instance of *bed* is correctly detected, note that in this case the instance is also attached with thin black pixel, (b) Results of spotting *bed* by the previous version of the method [1], (c) Results of spotting *bed* by Dutta et al. [7].



(a) (b) (c)

Fig. 11: (a) Results of detecting *sofa2*, here both the instances are correctly detected among which one of them was partially attached with thick wall, (b) Results of spotting *sofa2* by the previous version of the method [1], (c) Results of spotting *sofa2* by Dutta et al. [7].

qualitative results of spotting the symbols *bed*, *sofa2* and *table1* are shown in Fig. 6 to Fig. 11. The results of the present method is also compared with the previous version [1] and also with a previously proposed symbol spotting method by Dutta et al. [7]. In Fig. 6 to Fig. 11, the sub figures with label (a) show the results obtained by the current method, the sub figures with label (b) show the results obtained by the previous version of the method [1] and those with label (c) show the results obtained by the Dutta et al. [7]. For all the cases we have only considered the same smaller subset of images and query symbols. The quantitative results are listed in the Table I where the first row shows that for current version of the method, the second row shows that for previous version of the method [1], the third row shows graph-matching with a planar graph (with the preprocessing of this paper) and the last row shows the quantitative results by the method proposed by Dutta et al. [7]. The present version of the method has obtained a precision of 100% and recall of 88.75%. High precision proves that all the spotted instances of the symbol is correct. The obtained recall value indicates that the system loses some of the instances and from our investigation we can say that this kind of mis-detections often occur when the symbol is attached to a thick portion of black pixel. Vectorization methods specially perform worse with black thick pixels and creates lot of distortion in the vectorized information. The previous version of the method had obtained the precision and recall respectively as 32.99% and 77.55%, clearly the present version has gained an improvement.

TABLE I: Results obtained by the proposed method and the comparison with the previous version [1] and a previously proposed symbol spotting method [7].

Method	P	R	F
Current version	88.75	100.00	94.04
Previous version [1]	32.99	77.55	46.29
Without Hierarchy	54.55	90.00	67.92
Dutta et al. [7]	69.19	67.28	68.22

IV. Conclusion and Future Works

In this paper we have proposed an extension of the previously proposed hierarchical graph representation. We have introduced plausibilities to the nodes of the hierarchical graph, these plausibilities help to better match the hierarchical sub-structures through the computation of the association graph. The present method still has some scope of improvement, as we have shown in the experimental results that all kind distortions particularly heavy distortions viz. connected to thick black walls and etc still can not be solved. So the future work will address the further improvement of the method regarding noise. With the improvement, the construction of the hierarchical graph for this kind of graph representation is becoming complex and time taking. So another direction of future work will also concern about constructing hierarchical graph for different kind of graph representation. We have investigated that the hierarchical matching algorithm used by us sometime fails to find local optima and hence the solution is erroneous. We further investigated that a little modification of the matching algorithm provides much better results. Therefore improvement of the hierarchical matching will also be considered as a future work.

Acknowledgement

This work has been partially supported by the Spanish projects TIN2009-14633-C03-03, TIN2011-24631, TIN2012-37475-C02-02 and the PhD scholarship 2013FI_B2 00074.

References

[1] K. Broelemann, A. Dutta, X. Jiang, and J. Lladós, "Hierarchical graph representation for symbol spotting in graphical document images," in *Proceedings of S+SSPR*, ser. LNCS, G. Gimelfarb, E. Hancock, A. Imiya, A. Kuijper, M. Kudo, S. Omachi, T. Windeatt, and K. Yamada, Eds. Springer Berlin Heidelberg, 2012, vol. 7626, pp. 529–538.

[2] N. Ahuja and S. Todorovic, "From region based image representation to object discovery and recognition," in *Proceedings of S+SSPR*, ser. LNCS, E. Hancock, R. Wilson, T. Windeatt, I. Ulusoy, and F. Escolano, Eds. Springer Berlin, Heidelberg, 2010, vol. 6218, pp. 1–19.

[3] M. Pelillo, K. Siddiqi, and S. Zucker, "Matching hierarchical structures using association graphs," *IEEE TPAMI*, vol. 21, no. 11, pp. 1105 –1120, nov 1999.

[4] I. Bomze, M. Pelillo, and V. Stix, "Approximating the maximum weight clique using replicator dynamics," *IEEE TNN*, vol. 11, no. 6, pp. 1228 – 1241, nov 2000.

[5] P. L. Rosin and G. A. W. West, "Segmentation of edges into lines and arcs," *Image and Vision Computing*, vol. 7, no. 2, pp. 109–114, 1989.

[6] D. Conte, P. Foggia, C. Sansone, and M. Vento, "Thirty years of graph matching in pattern recognition," *IJPRAI*, vol. 18, no. 3, pp. 265–298, 2004.

[7] A. Dutta, J. Lladós, and U. Pal, "A symbol spotting approach in graphical documents by hashing serialized graphs," *Pattern Recognition*, vol. 46, no. 3, pp. 752–768, 2013.

[8] M. Delalandre, T. Pridmore, E. Valveny, H. Locteau, and E. Trupin, *Building Synthetic Graphical Documents for Performance Evaluation*. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 288–298.