Hierarchical graph representation for symbol spotting in graphical document images

Klaus Broelemann[1], Anjan Dutta[2], Xiaoyi Jiang[1], and Josep Lladós[2]

[1] Department of Mathematics and Computer Science, University of Münster, Germany
[2] Computer Vision Center, Universitat Autonoma de Barcelona, Spain

Abstract. Symbol spotting can be defined as locating given query symbol in a large collection of graphical documents. In this paper we present a hierarchical graph representation for symbols. This representation allows graph matching methods to deal with low level vectorization errors and, thus, to perform a robust symbol spotting. To show the potential of this approach, we conduct an experiment with the SESYD dataset.

Keywords: hierarchical graph representation, graph matching, maximal clique finding, symbol spotting, graphics recognition

1 Introduction

Symbol spotting has experienced a growing interest among the graphics recognition community. It can be defined as locating a given query graphical symbol into a set of graphical document images. Example applications of symbol spotting are finding a mechanical part in a database of engineering drawings or retrieving invoices of a provider from a large database of documents by querying a particular logo. The problem of symbol spotting in documents for real world situation is difficult as the documents often suffer from different noises. Graphs are very effective tool to represent any graphical elements, especially line drawings. Moreover, when graphs are attributed by geometric information, this supports various affine transformation viz. translation, rotation, scaling etc. Hence, in line drawings represented by graphs, the problem of symbol spotting can be formulated as a subgraph matching problem, where graph theory offers robust approaches to solve it. This explains our motivation to work with graphs.

The list of approaches proposed for spotting symbols in graphical documents is long [10]. The current paper only mentions the recent works dealing with the graph representations: Nayef and Breuel [7] proposed a branch and bound algorithm for spotting symbols in documents, where they used geometric primitives of images as features. Luqman *et al.* [6] proposed a graph embedding based subgraph spotting method applied to symbol spotting. Here the candidate regions containing symbols are filtered out beforehand using some criteria of loop. Recently Dutta *et al.* [5] proposed graph factorization based symbol spotting methods for architectural floorplans. Of course, the above set of algorithms deal

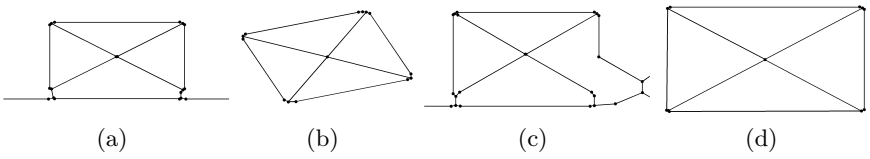

Fig. 1. Examples for low-level segmentation errors.

with some kind of error tolerance when matching the subgraphs but they seldom can handle disconnection between nodes i.e. when two nodes are disconnected but supposed to belong to same graph. In case of such disconnection usually the methods just loose the connectivity which reduce some topological feature of the graph. So handling these kind of distortions are the inspiration of proposing a hierarchical representation of graph where we deal with different kind of errors propagated from the lower level to the graph level.

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 Figure 1). Our present work deals with this kind of distortion in the graph level, to do that we propose hierarchical representation of graphs. The hierarchical representation of graphs allows to incorporate the various segmentation errors hierarchically. The main motivation of our work comes from [1], where the authors used the hierarchical representation of the segmented image regions and later used an approximated maximal clique finding algorithm to match the two hierarchical representation. In particular, the aforementioned work was applied to match two different natural images.

The rest of the paper is organized into four sections. In Section 2 we present the hierarchical representation of the graphs to represent a database in terms of the descriptors of graph paths. Section 3 describes the hierarchical graph matching methods we used. Section 4 contains the detailed experimental results. After that, in Section 5, we conclude the paper and discuss future work.

2 Hierarchical graph representation

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.

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.

2.1 Vectorization

Graph representation of documents follows some pre-processing 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 done it with the Rosin-West algorithm [8] which is implemented in the Qgar package [3]. This particular 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, artefacts 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 artefacts 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.

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.

2.2 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 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. These possibilities are connected hierarchically. This allows us to embed the constraint not to match two hierarchically connected

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

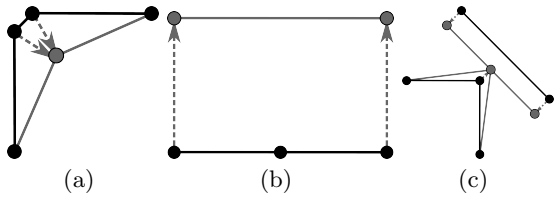

Fig. 2. Three cases for simplification. Displayed are the original nodes and edges (black) and the simplified nodes and their edges (gray): (a) Merge nodes (b) Remove dispensable node (c) Merge node and edge.

nodes into the graph matching and, thus, only accept one alternative. In section 3 we will give further details for the graph matching and this constraint.

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 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. Figure 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 Figure 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 Figure 2 (c).

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.

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. Formally, we define a

hierarchical graph H as a tuple $H = (V, E_N, E_H, \sigma)$ with the neighborhood edges $E_N \subseteq V \times V$ and the hierarchical edges $E_H \subseteq V \times V$.

To detect the three previously described cases, we define:

1. function $\delta_1 : V \times V \to \{0, 1\}$ to test for pairs of nearby nodes.
2. function $\delta_2 : V \to \{0, 1\}$ to test for dispensable nodes.
3. function $\delta_3 : V \times E \to \{0, 1\}$ to test for nodes near to edges.

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: $L(u) = \{v \in V_G | v \rightsquigarrow w\}$. Based on these functions and formulations we can define the hierarchical simplification $H = \mathcal{H}(G) = (V, E_N, E_H, \sigma)$ of G by the following rules:
Initial. As initialization for the recursion, G is a subgraph of H, i.e. $V_G \subseteq V$ and for $u, v \in V_G : (u, v) \in E_G \Leftrightarrow (u, v) \in E_N$
Merging. For $u, v \in V$ with $\delta_1(u, v) = 1$ 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: $\sigma(w) = \frac{1}{|L(w)|} \sum_{s \in L(w)} \sigma(s)$

Removing. For a dispensable node $u \in V$ with $\delta_2(u) = 1$ there exist two nodes $v, w \in V_G$ with $(u, v), (u, w) \in E_N$. Since v and w can have hierarchical successors, 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: $\sigma(v_i) = \sigma(\bar{v_i})$, $\sigma(w_j) = \sigma(\bar{w_j})$

Node/Edge merging. For $u \in V, e = (v, w) \in E$ with $\delta_3(u, e) = 1$ 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 $\sigma(w) = \sigma(\bar{w})$

Based on these recursive rules, we construct the smallest hierarchical graph that satisfies these rules, i.e. no additional nodes are added. For our hierarchical graph we defined the testing functions $\delta_1, \delta_2, \delta_3$ by using thresholds: for δ_1 define an upper bound for the distance between two nodes, for δ_3 we do the same for the distance between edge and node. We define δ_2 by a threshold for the relative distance of the dispensable node from the direct line between it's neighbors. In contrast to other definitions like the angle at the dispensable node, this can easily be extended to chains of dispensable nodes.

Pre-processing Depending on the chosen thresholds there can be a huge number of possibilities and, thus, a large hierarchical graph. To reduce the size of the hierarchy, we perform a pre-processing step. The idea is that in some cases the confidence in the simplification is strong enough not to store both versions, e.g. it is not very likely that a one-pixel gap is intentional. For that purpose we perform merging and removing steps on the graph with stricter threshold. With these thresholds we do not create hierarchically connected possibilities, but change the original graph structure.

3 Graph matching

The previous section has presented a hierarchical graph representation for vectorized drawings. In this section we will describe how to make use of this representation for subgraph matching in order to spot symbols. Graph matching has a long history in pattern recognition and there exist several algorithms for this problem [3]. Our approach is based on solving maximal weighted clique problem in association graphs [2]. 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 $H^i = (V^i, E_N^i, E_H^i, \sigma^i)$, $i = 1, 2$, 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) and (v_1, v_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 holds also 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 the same.

We use replicator dynamics [2] 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_s^1 and consider all the hierarchical and normal edges connecting the nodes in V_s^1 as in H^1, this forms a subgraph which we

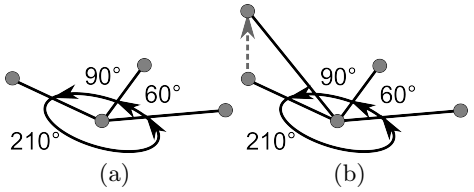

Fig. 3. Example for node labels for graphs based on angles between edges: (a) for planar graphs and (b) for hierarchical graphs. Both will be labeled with $(90, 210, 60)$

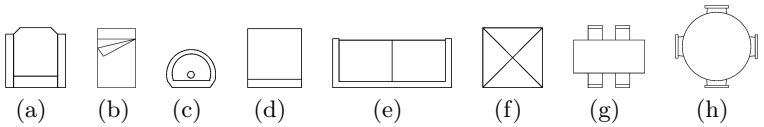

Fig. 4. Model symbols in the SESYD dataset used for our experiment.

can denote as $H_s^1 = (V_s^1, E_{s_N}^1, E_{s_H}^1, \sigma_s^1)$. 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 [1]). Since we perform n iterations, we get a complexity of $\mathcal{O}(n \cdot |A|^2)$

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 shall also reflect geometric structures, we use geometric attributes for the similarity.

In a non-hierarchical plane graph, nodes can be labeled by the angles between adjacent edges. Figure 3 (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 of possibilities is also affected by the level of distortion of the original image, such an approach is not robust to distortion.

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 figure 3 (b): though the central node is connected to four nodes, only three edgesare 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 that is used to weight the nodes of the association graph.

4 Experimental results

We have evaluated the performance of our method on the SESYD (floorplans)[4] database which is a synthetically generated graphical document benchmark [4]. Actually, this dataset contains 10 different subdatasets, each of which consists of 100 different synthetically generated floorplans and 16 model symbols (see Figure 4). For this work we have considered one such subdataset and all the 8 randomly chosen query symbols. All the floorplans in a subdatasets are created on a same floorplan template by putting different model symbols in different places in random orientation and scale. The query symbol is always ideal and does not contain any distortion. The average number nodes in the query graph and the input graph are 12 and 1500 respectively. Since we are focused on the document retrieval aspect of the problem, we use the standard performance measures of precision (**P**), recall (**R**) and F-measure (**F**) for evaluating the performance of our system. For a more detailed discussion on performance evaluation of spotting systems we refer to [9].

The results obtained by our system are presented in Table 1 in a symbol wise manner, which shows that the method is not equally successful for all the symbols, in particular for the simple symbols with trivial nodes, for example sofa1 (Figure4(d)). This is because the nodes of the graph representing those symbols contain similar attributes with the nodes from the background. In general, the precision of the algorithm is quite good which ensures the confidence of the system for retrieving the system. The recall values vary depending on the symbol but in most of the cases it is quite satisfactory. This ensures that most of the instances of the query symbols can be retrieved by the system. To get an idea about the results obtained the system, in Figure 5 we present the symbol spotting results of querying armchair (Figure 4(a)) and table1 (Figure 4(f)).

5 Conclusion and future work

In this paper we have presented a new hierarchical graph representation that enables us to store different possibilities for the vectorization of a drawing in one

[4] http://mathieu.delalandre.free.fr/projects/sesyd/index.html

Table 1. Results with SESYD dataset

Symbol	Precision	Recall	F-measure	Symbol	Precision	Recall	F-measure
armchair	92.71	83.86	88.06	sofa1	32.65	77.45	45.94
bed	23.67	87.17	37.23	sofa2	47.98	81.87	60.50
table1	98.56	97.23	97.89	table2	32.76	79.98	46.48
sink1	82.85	78.98	80.87	table3	23.51	78.23	36.15

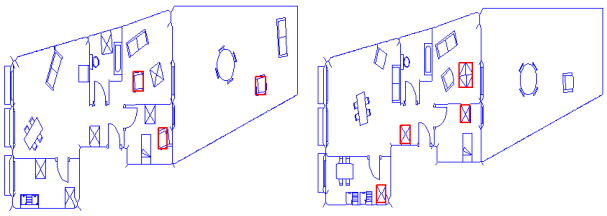

Fig. 5. Qualitative results of retrieving armchair (Figure 4(a)) and table1 (Figure 4(f)).

graph. With this representation, symbol spotting by graph matching can deal with typical vectorization errors. We could show the efficiency in an experiment.

Though our method performs well for most symbols, we still have some problems with too simple symbols. In the future we want to improve the efficiency for simple symbols and apply the approach to free-hand sketches, which have a higher level of distortion.

References

1. N. Ahuja and S. Todorovic. From region based image representation to object discovery and recognition. In Edwin Hancock, Richard Wilson, Terry Windeatt, Ilkay Ulusoy, and Francisco Escolano, editors, *Proceedings of S+SSPR*, volume 6218 of *LNCS*, pages 1–19. Springer Berlin, Heidelberg, 2010.
2. I.R. Bomze, M. Pelillo, and V. Stix. Approximating the maximum weight clique using replicator dynamics. *IEEE TNN*, 11(6):1228 – 1241, nov 2000.
3. D. Conte, P. Foggia, C. Sansone, and M. Vento. Thirty Years Of Graph Matching In Pattern Recognition. *IJPRAI*, 18(3):265–298, 2004.
4. Ma. Delalandre, T. Pridmore, E. Valveny, H. Locteau, and E. Trupin. *Building Synthetic Graphical Documents for Performance Evaluation*, pages 288–298. Springer-Verlag, Berlin, Heidelberg, 2008.
5. A. Dutta, J. Lladós, and U. Pal. Symbol spotting in line drawings through graph paths hashing. In *Proceedings of 11th ICDAR*, pages 982–986, 2011.
6. M. M. Luqman, J. Ramel, J. Llados, and T. Brouard. Subgraph spotting through explicit graph embedding: An application to content spotting in graphic document images. In *Proceedings of 11th ICDAR*, pages 870–874, 2011.
7. N. Nayef and T. M. Breuel. A branch and bound algorithm for graphical symbol recognition in document images. In *Proceedings of Ninth IAPR International Workshop on DAS*, pages 543–546, 2010.
8. P. L. Rosin and G. A. W. West. Segmentation of edges into lines and arcs. *Image and Vision Computing*, 7(2):109–114, 1989.
9. M. Rusiñol and J. Lladós. A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices. *IJDAR*, 12(2):83–96, 2009.
10. K. Tombre and B. Lamiroy. Pattern recognition methods for querying and browsing technical documentation. In *Proceedings of 13th CIARP, LNCS, vol. 5197, Springer-Verlag*, 2008.