A Product graph based method for dual subgraph matching applied to symbol spotting

Anjan Dutta and Josep Lladós
Computer Vision Center
Universitat Autònoma de Barcelona
Barcelona, Spain
Email: {adutta,josep}@cvc.uab.es

Horst Bunke
Inst. of Comp. Sc. and Appl. Maths
University of Bern
Bern, Switzerland
Email: bunke@iam.unibe.ch

Umapada Pal
CVPR Unit
Indian Statistical Institute
Kolkata, India
Email: umapada@isical.ac.in

***Abstract*—Product graph has been shown to be an efficient way for matching subgraphs. This paper reports the extension of the product graph methodology for subgraph matching applied to symbol spotting in graphical documents. This paper focuses on the two major limitations of the previous version of product graph: (1) Spurious nodes and edges in the graph representation and (2) Inefficient node and edge attributes. To deal with noisy information of vectorized graphical documents, we consider a *dual graph* representation on the original graph representing the graphical information and the product graph is computed between the *dual graph*s of the *query graph*s and the *input graph*. The *dual graph* with redundant edges is helpful for efficient and tolerating encoding of the structural information of the graphical documents. The adjacency matrix of the product graph locates similar path information of two graphs and exponentiating the adjacency matrix finds similar paths of greater lengths. Nodes joining similar paths between two graphs are found by combining different exponentials of adjacency matrices. An experimental investigation reveals that the recall obtained by this approach is quite encouraging.**

I. Introduction

Product graph was introduced for computing the random walk graph kernel for measuring the similarity between two graphs. Recently it has been used for subgraph matching and applied for spotting symbols on graphical documents [1]. In this paper we propose an extension and improvement of the product graph. Particularly, this work mainly focuses on the two major limitations of the previous version of the method: (1) Spurious nodes and edges that are generated during low level image processing and (2) Erroneous node and edge attributes. Of course, these problems are application and representation dependent, but there is no doubt that the proposed solution is more robust to distortion and noise, which will be further useful for other applications and representations.

Graph representation of graphical documents involves some low level image processing viz. binarization, skeletonization, vectorization etc. For example, we use Qgar[1] for vectorizing the given binary images. This particular vectorization generates critical points and connectivity information between them. For representing a document with a graph, we consider the critical points as the nodes and the lines joining them as the edges (Fig. 1). The main problem in this kind of low level image processing is the addition of noisy information. As an example, Fig. 2(b) shows the graph representation of the symbol in Fig. 2(a) under the previously mentioned representation scheme, whereas Fig. 2(c) shows the ideal graph representation of the symbol. Here we can see an example of the introduction of numerous spurious nodes near the junctions and corners. Note that such a vectorization can also generate spurious and discontinuous edges. This kind of structural noise always creates problems for matching or comparing (sub)graphs. It is true that with a different kind of graph representation it may be possible to solve the problem more efficiently, but dealing with this kind of distortions or noise at the graph level is interesting for other domain also as it gives more robustness in the matching method.

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

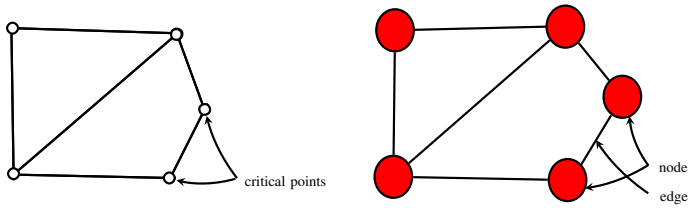


Fig. 1: Each critical point detected by the vectorization technique is considered as a node and the straight lines joining them are considered as the edges.

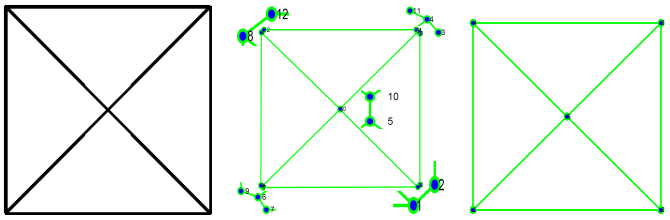


Fig. 2: Difference between a real graph representation and an ideal graph representation: (a) an architectural symbol, (b) the graph representation of the symbol in (a) after doing the vectorization (note the spurious nodes near the corners and junctions), (c) the ideal graph representation of the symbol in (a), showing how the graph representation should be.

Product graphs have been introduced for computing random walk graph kernels [2]. In our previous work, we introduced product graph for subgraph matching and applied it

for spotting symbols in graphical documents [1]. Formally, a symbol spotting method can naturally be formulated as a sub-graph matching problem where a *query symbol* is considered as *model* or *query graph* and the big target document can be considered as a *target* or *input graph*. For more information on symbol spotting methods based on graph representation please refer to [3], where a literature review is given. Product graph provides an efficient comparison technique between a subgraph and a graph in terms of different substructures (in this case paths) which allows the entire subgraph matching process to be computed online. The product graph finds the similar pair of nodes in the *input graph* with a pair of nodes in the *query graph* and join them with an edge. In this way the similar pair of nodes in these two graphs are supposed to be connected with edges, this information can be obtained from the adjacency matrix of the product graph. Now, exponentiating the adjacency matrix provides the information about similar pair of nodes which are connected with paths of greater lengths. This was the main motivation of the work in [1] but one of the problems was selecting efficient node and edge attributes, especially when the graphs contain noise or distortions like spurious nodes and edges, and the possibility of discontinuous edges. To solve this problem in this paper we introduce the *dual graph* DG with redundant edges of the original graph G and consider the product graph of the *dual graph*s. Here the *dual graph* DG of the original graph G is a graph that has a vertex corresponding to each edge of G and an edge joining two neighboring edges in G. In this work we use a *dual graph* with redundant edge to cope with the distortions and noise as explained in Section II. The variation of *dual graph* provides us robust node and edge labels and with this information the product graph is better suited for subgraph matching applied to symbol spotting.

The rest of the paper is organized into three sections. In Section II, we present the methodology to represent the graphical documents with *dual graph* with redundant edges, computing the node and edge labels and computing the product graphs to spot the symbol on documents. Section III contains a description of our current experimental results. After that, in Section IV, we conclude the paper and discuss future directions of work.

II. Methodology

Let $G_M = (V_M, E_M)$ be the graph representing the *model* or *query symbol* and $G_I = (V_I, E_I)$ be the graph representing the *input* or *target image*, where V_M and V_I are the set of vertices, in this case the critical points, and E_M and E_I are the set of edges connecting the critical points. Now the subgraph matching is intended to find the different instances or occurrences of G_M in G_I.

A. Dual graph

Let $DG_M = (DV_M, DE_M, \alpha_M, \beta_M^1, \beta_M^2)$ and $DG_I = (DV_I, DE_I, \alpha_I, \beta_I^1, \beta_I^2)$ be the *dual graph*s on the original graphs G_M and G_I respectively, where $DV_M = E_M$ and $DV_I = E_I$. Clearly, each node in DG_M, DG_I represents an edge in G_M, G_I respectively. Here onwards, in this paper, we will be denoting a node in the *dual graph* which joins two nodes, say, v_i and v_j in the original graph as dv_{ij}. Now the

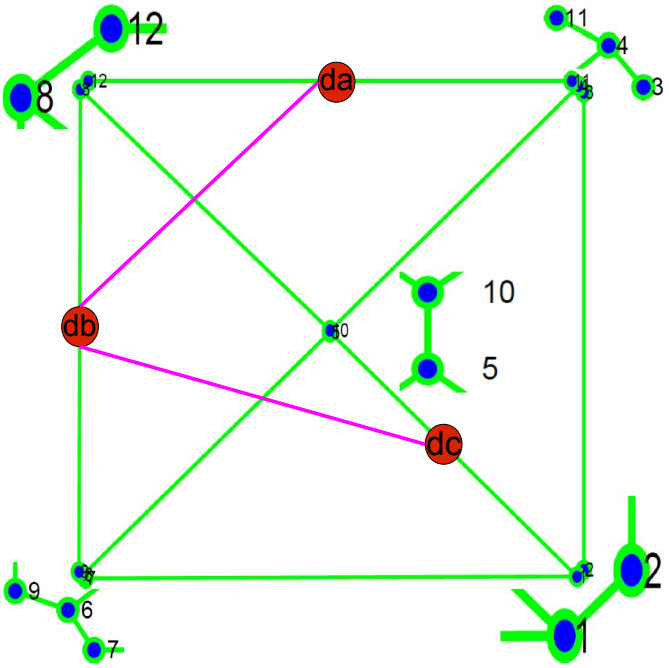


Fig. 3: Details of adding redundant edges in the *dual graph* considering only three nodes da, db and dc and $n = 3$: The graphdist$(da, db) = 2$ and graphdist$(db, dc) = 3$, that is why there exist edges (da, db) and (db, dc) but since graphdist$(da, dc) = 4$, there is no edge (da, dc) (Note the spurious nodes, edges near the corners and junctions in the original graph that are plotted in green color).

edge sets of DG_M and DG_I are defined as follows:

$$DE_I = \{(dv_{ij}, dv_{kl}) : dv_{ij}, dv_{kl} \in DV_I \text{ are adjacent}\}$$

$$DE_M = \{(du_{ij}, du_{kl}) : du_{ij}, du_{kl} \in DV_M \text{ are adjacent}\}$$

We introduce the redundant edges to our *dual graph*s and update DE_I and DE_M as follows (see Fig. 3):

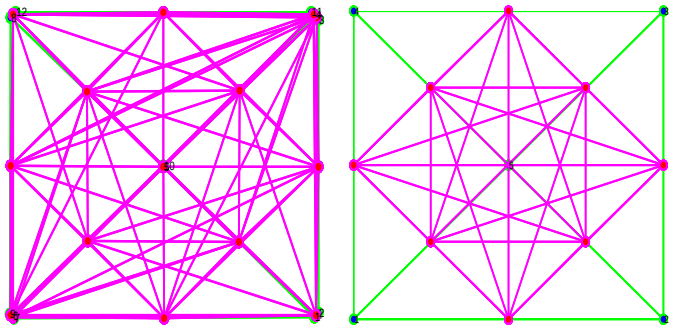
$$DE_I = DE_I \cup \{(dv_{ij},dv_{kl}) : \text{graphdist}(dv_{ij}, dv_{kl}) \leq n, \\ n \in \mathbb{N} \text{ and } dv_{ij}, dv_{kl} \in DV_I\}$$

$$DE_M = DE_M \cup \{(du_{ij},du_{kl}) : \text{graphdist}(du_{ij}, du_{kl}) \leq n, \\ n \in \mathbb{N} \text{ and } du_{ij}, du_{kl} \in DV_M\}$$

Here graphdist(du, dv) stands for the minimum number edges between the nodes du and dv in a *dual graph*. Now onwards in this paper, by *dual graph* we mean the *dual graph* with redundant edges.

$\alpha_I : DV_I \to \mathbb{R}^7$ is a node labeling function and is defined as $\alpha_I(dv_{ij}) =$ Hu moments invariants [4] of the acyclic graph paths between v_i and v_j of length less than or equal to $m \in \mathbb{N}$, $dv_{ij} \in DV_I$. Similarly, $\alpha_M(du_{ij}) =$ is the same node labeling function but defined in DG_M. It is to be mentioned that for each path, we get a vector of dimension seven and hence for a node dv in a *dual graph* DG, we get a set of Hu moments invariants, let us denote the set as $Hu(dv)$. $\beta_I^1 : DE_I \to \mathbb{R}$ is an edge labeling function and is defined as:

$$\beta_I^1(dv_{ij}, dv_{kl}) = \min \angle(dv_{ij}, dv_{kl}), (dv_{ij}, dv_{kl}) \in DE_I.$$



(a) (b)

Fig. 4: Original graph and redundant *dual graph* representation (with $n = 3$) of (a) a real symbol and (b) an ideal symbol. The original graph is plotted with green color and its corresponding *dual graph* with redundant edges is plotted with magenta color (for details see Fig. 3).

$\beta_I^2 : DE_I \rightarrow \mathbb{R}$ is another edge labeling function and is defined as:

$$\beta_I^2(dv_{ij}, dv_{kl}) = \frac{\text{mdist}(dv_{ij}, dv_{kl})}{\max(\text{length}(dv_{ij}), \text{length}(dv_{kl}))}, \\ (dv_{ij}, dv_{kl}) \in DE_I.$$

here $\text{mdist}(dv_{ij}, dv_{kl})$ is the length of the line joining the midpoint of the edges dv_{ij} and dv_{kl}. β_M^1 and β_M^2 are defined respectively as β_I^1 and β_I^2 but in DE_M.

B. Product graph

Product graph $G_P = (V_P, E_P)$ of DG_I and DG_M relates the *input graph* DG_I and the model graph DG_M with the node and edge sets. The properties or the conditions are included in the set definitions as follows:

$$V_P = \{(du_{ij}, dv_{ij}) : du_{ij} \in DV_M, dv_{ij} \in DV_I, \\ \alpha_M(du_{ij}) \simeq \alpha_I(dv_{ij}), \\ \beta_M^1(du_{ij}, du_{kl}) \simeq \beta_I^1(dv_{ij}, dv_{kl}) \text{ and} \\ \beta_M^2(du_{ij}, du_{kl}) \simeq \beta_I^2(dv_{ij}, dv_{kl})\}$$

and given the above set of nodes, the edge set E_P will be:

$$E_P = \{((du_{ij}, dv_{ij}), (du_{kl}, dv_{kl})) : (du_{ij}, du_{kl}) \\ \in DE_M, (dv_{ij}, dv_{kl}) \in DE_I\}$$

We use the parameters t_α, t_{β_1} and t_{β_2} for measuring the node and edge similarities as follows:

$$\alpha_M(du_{ij}) \simeq \alpha_I(dv_{ij}) \Leftrightarrow |\alpha_M(du_{ij}) - \alpha_I(dv_{ij})| \leq t_\alpha$$

$$\beta_M^1(du_{ij}, du_{kl}) \simeq \beta_I^1(dv_{ij}, dv_{kl}) \Leftrightarrow \\ |\beta_M^1(du_{ij}, du_{kl}) - \beta_I^1(dv_{ij}, dv_{kl})| \leq t_{\beta_1}$$

$$\beta_M^2(du_{ij}, du_{kl}) \simeq \beta_I^2(dv_{ij}, dv_{kl}) \Leftrightarrow \\ |\beta_M^2(du_{ij}, du_{kl}) - \beta_I^2(dv_{ij}, dv_{kl})| \leq t_{\beta_2}$$

Here

$$|\alpha_M(du_{ij}) - \alpha_I(dv_{ij})| = \sum_{p_1 \in Hu(du_{ij})} \min_{p_2 \in Hu(dv_{ij})} d(p_1, p_2)$$

where $d(.,.)$ denotes the Euclidean distance.

From the above definition of product graph it is clear that an edge in the product graph G_P corresponds to the similar connected node pairs in DG_M and DG_I. A simple synthetic example is given in Fig. 5, where we have two different graphs with discrete labels, one having node labels $\{a, b, c, d, e\}$ and the other with node labels $\{x, y, z\}$. For simplicity, let us ignore the edge labels. Now, if we define the node label similarities as $a = x$, $b = y$ and $c = z$, we get the product graph in Fig. 5. Here, each edge in the product graph denotes a connected node pair.

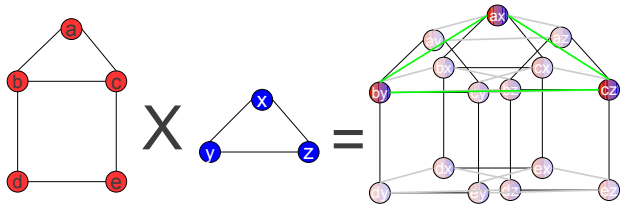


Fig. 5: Demonstration of product graph computation with a synthetic example.

C. Powers of adjacency matrix

If we multiply the adjacency matrix E_P of the product graph G_P once with itself, we get the E_P^2. The (i, j)th element of E_P^2 denotes the number of paths of length two between nodes i and j. Since matrix multiplication is associative we can exponentiate E_P as follows to get E_P^n:

$$E_P^n = E_P.(E_P)^{n-1} = (E_P)^{n-1}.E_P$$

It is well known that the (i, j)th element of E_P^n always denotes the number of paths of length n between i and j [2]. Since an edge in E_P denotes the similarity between respective connected node pairs, paths of longer length denote series of similar nodes between DG_M and DG_I. This information is helpful for getting nodes in DG_I that are similar to nodes in DG_M and it can be utilized by integrating different weighted exponentiations of E_P into a new matrix A as follows:

$$A = \sum_{i=1}^{nl} \epsilon^i \times E_P^i, 0 < \epsilon \leq 1 \text{ s.t.} \sum_{i=1}^{nl} \epsilon^i = 1.$$

where nl is the maximum number of times the matrix E_P is being exponentiated.

These weights to different path lengths are inspired by the traditional random walk graph kernel [2]. This way of weightings reduces the influence of paths of greater lengths because they often contains redundant information.

Now let us take an example of the matrix A as follows:

$$A = \begin{array}{c|cccccc} & (du_1,dv_1) & \cdots & (du_k,dv_l) & (du_{k+1},dv_{l+1}) & \cdots & (du_m,dv_n) \\ (du_1,dv_1) & 0 & \cdots & 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (du_{l-1},dv_{k-1}) & 0 & \cdots & 0 & v_1 & \cdots & 0 \\ (du_l,dv_k) & 0 & \cdots & v_2 & 0 & \cdots & 0 \\ (du_{l+1},dv_{k+1}) & 0 & \cdots & 0 & v_3 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ (du_m,dv_n) & 0 & \cdots & 0 & 0 & \cdots & 0 \end{array}$$

TABLE I: Symbol-wise mean results with the Product graph methodology: the results under the "New version" list the results of the current method whereas the results under the "Old version" list the previous results.

	New version					Old version				
Symbol	**P**	**R**	**F**	**AveP**	**T (secs)**	**P**	**R**	**F**	**AveP**	**T (secs)**
armchair	47.86	99.69	64.67	47.61	14.05	15.34	78.13	25.65	16.21	0.56
bed	79.37	100.00	88.50	80.90	8.71	26.49	100.00	41.89	27.72	0.43
sofa2	32.23	100.00	48.75	34.60	7.62	15.23	45.00	22.76	15.43	0.39
table1	91.98	100.00	95.83	92.42	14.42	66.07	78.72	71.84	67.14	0.78
table3	7.82	100.00	14.51	7.22	97.63	3.57	87.50	6.86	3.87	1.26

where $m < n$ and $1 < k, l \leq m, n$.

Let us further assume for simplicity that only v_1,v_2 and v_3s are the real numbers greater than zero and all the other values in A are zero. As, for example, $v_1 \neq 0$, this particularly signifies that $du_{l-1} \simeq dv_{k-1}$ and $du_{l+1} \simeq dv_{k+1}$ and also $(du_{l-1}, du_{l+1}) \simeq (dv_{k-1}, dv_{k+1})$ according to the node and edge similarity defined before. Now it can be noted that if two nodes u, v in a graph $G = (V, E)$ are connected with more than one paths, they supposed to have more and more random walks of different lengths which should be reflected in the matrix A. Following the above explanation it is to be mentioned that similar nodes in DG_M and DG_I should be connected with different paths in the product graph G_P. So the non zero entry in the combined weighted matrix A identifies the similar nodes in DG_I with DG_M and with this the occurrences of the graph DG_M can be found in DG_I. The similar pair of nodes in the redundant *dual graph*s should be connected in the matrix A as above and the dissimilar pair of nodes should get the zero entries, as in the exponentiation of the adjacency matrix E_P, the existence of solitary graph edge must be diminished. The connected sub-component in the matrix A (in this case component with non zero entries viz. v_1, v_2 and v_3) can be found by searching maximal subgroup of entries that are mutually reachable (`graphconncomp` function in the `matlab`). Each component is then regarded as a single instance of the *query graph* in the *input graph* and can be found according to the position of the second nodes of the vertices of A.

III. Experimental results

Our experiments were conducted on the SESYD dataset[2]. This dataset contains 10 different subsets and 16 query symbols. 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. In this experiment we have only considered a subset of 100 images (floorplans16-01). The symbol-wise quantitative results (precision (**P**), recall (**R**), f-measure (**F**), average precision (**AveP**) and time (**T**)) for five symbols are shown in the Table I under the column "New version". The recall values for all the five symbols are very high, which proves that the method can find most of the true occurrences in the images. The precision values for some symbols are quite low though, and this is because of the occurrence of false positives. Also in the same table we have listed the results of the previous version of the product graph in the "Previous version" column, which shows in the newer version we have achieved substantial improvement in terms of both precision and recall but the time complexity is substantially high. This is because in the previous version the node and edge labels were quite simple. Furthermore, the presence of redundant edges in the *dual graph* increase the number of comparison and hence the time complexity. Note that in both the version of product graph the node and edge labeling are computed online, since the time complexity of the newer version is higher the node and edge labeling can be done offline to reduce the online time. Qualitative results are shown in Fig. 7 to Fig. 11. All the experiments are done with the parameter values set as: $t_\alpha = 0.025 \times |Hu(dv)|$, where $|Hu(dv)|$ is the number of paths connecting the two terminals of dv, $t_{\beta_1} = 6$, $t_{\beta_2} = 0.2$, $m = 15$, $n = 3$ and $nl = 10$, these parameters are chosen to give the optimal performance.

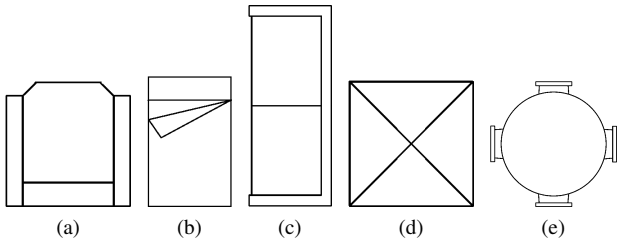


Fig. 6: Model symbols used to perform experiments:(a) *armchair*, (b) *sofa2*, (c) *table1*, (d) *table3*.

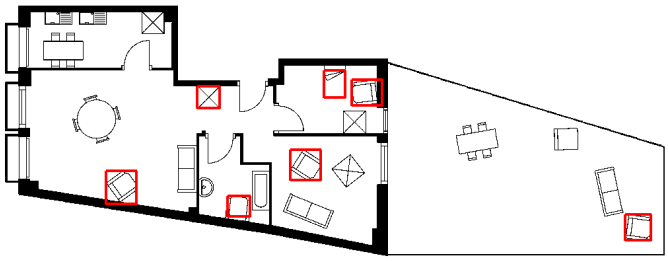


Fig. 7: Qualitative results of spotting *armchair* which shows correct detection of all the occurrences of *armchair*, at the same time it includes two false detection of *bed* and *table1* respectively.

IV. Conclusions

In this paper we extend the product graph methodology by using the *dual graph* rather than the original representation as a basis. It turns out that this *dual graph* representation provides a robust way to deal with noise and distortions in the structural information. The product graph exhibits similar paths

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

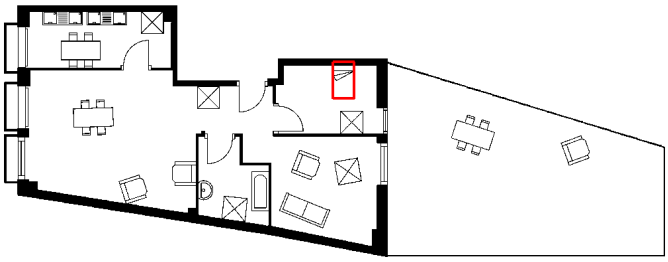


Fig. 8: Qualitative results of spotting *bed* which shows correct detection of the only one occurrence of *bed*, note there is no false detection.

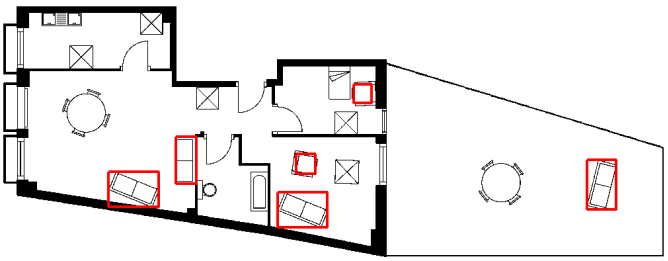


Fig. 9: Qualitative results of spotting *sofa2* which shows correct detection of all the occurrences of *sofa2* and also it false detection of two instances of *armchair*. This is because of the square regions in *armchair* resemble with the square region in *sofa2*.

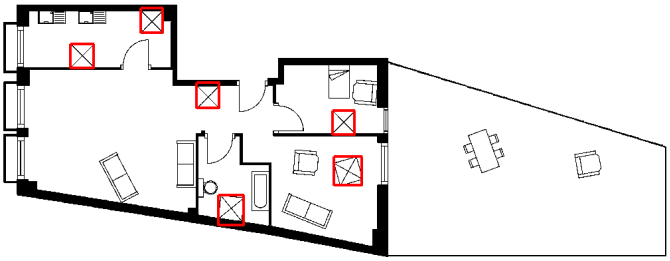


Fig. 10: Qualitative results of spotting *table1* which shows all the correct detection.

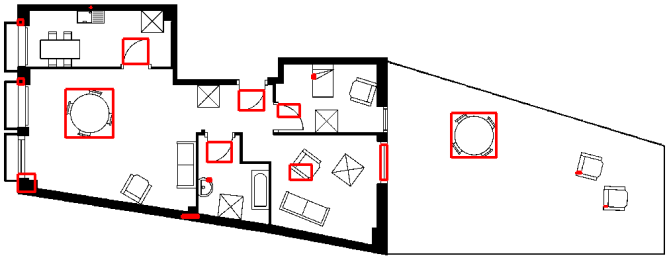


Fig. 11: Qualitative results of spotting *table3* which shows all the correct detection but a lot of false positives most of them resemble with the smaller subpart of the symbol *table3*.

and exponentiation of the product graph's adjacency matrix allows one to extract similar paths of any desired length. So these path similarities help to get similar nodes in the *target* and *input graph*. The recall values obtained by the method are very encouraging albeit we experienced lot of false positives at the same time. A closer analysis reveals that this kind of false positives are generated due to the tottering between the nodes in the product graph [5]. So our future work will address the removal of tottering from exponentiated adjacency matrices [6], [7].

REFERENCES

[1] A. Dutta, J. Gibert, J. Lladós, H. Bunke, and U. Pal, "Combination of product graph and random walk kernel for symbol spotting in graphical documents," in *Proceedings of 21st International Conference of Pattern Recognition*, 2012, pp. 1663–1666.

[2] T. Gärtner, P. A. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *COLT*, 2003, pp. 129–143.

[3] 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.

[4] M.-K. Hu, "Visual pattern recognition by moment invariants," *Information Theory, IRE Transactions on*, vol. 8, no. 2, pp. 179–187, 1962.

[5] P. Mahé, N. Ueda, T. Akutsu, J.-L. Perret, and J.-P. Vert, "Graph kernels for molecular structure-activity relationship analysis with support vector machines," *Journal of Chemical Information and Modeling*, vol. 45, no. 4, pp. 939–951, 2005.

[6] F. Aziz, R. Wilson, and E. Hancock, "Backtrackless walks on a graph," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 6, pp. 977–989, 2013.

[7] H. Stark and A. Terras, "Zeta functions of finite graphs and coverings," *Advances in Mathematics*, vol. 121, no. 1, pp. 124 – 165, 1996.