Symbol Spotting in Line Drawings Through Graph Paths Hashing

Anjan Dutta
Computer Vision Centre
Universitat Autònoma de Barcelona
Barcelona, Spain
adutta@cvc.uab.es

Josep Lladós
Computer Vision Centre
Universitat Autònoma de Barcelona
Barcelona, Spain
josep@cvc.uab.es

Umapada Pal
CVPR Unit, Indian Statistical Institute
203 B. T. Road, Kolkata 700108
Kolkata, India
umapada@isical.ac.in

***Abstract*—In this paper we propose a symbol spotting technique through hashing the shape descriptors of graph paths (Hamiltonian paths). Complex graphical structures in line drawings can be efficiently represented by graphs, which ease the accurate localization of the model symbol. Graph paths are the factorized substructures of graphs which enable robust recognition even in the presence of noise and distortion. In our framework, the entire database of the graphical documents is indexed in hash tables by the locality sensitive hashing (LSH) of shape descriptors of the paths. The hashing data structure aims to execute an approximate k-NN search in a sub-linear time. The spotting method is formulated by a spatial voting scheme to the list of locations of the paths that are decided during the hash table lookup process. We perform detailed experiments with various dataset of line drawings and the results demonstrate the effectiveness and efficiency of the technique.**

***Keywords*-Symbol spotting, Graph factorization, Graph paths hashing, Graphics recognition, Shape descriptors.**

I. Introduction

Information spotting is a major branch of indexing and retrieval methods in document image databases. The research community is mainly focused in word spotting for textual documents and symbol spotting for graphical documents. Nowadays, 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 desired output for a particular query should be a ranked list of retrieved symbols in which the true positives should appear at the beginning. Symbol spotting follows the segmentation and recognition paradox, that is a symbol spotting architecture does not use a previous segmentation step followed by a proper recognition method, instead it conceives to coarsely recognize and segment in a single step. This demands certain techniques that can handle the recognition without segmentation and segmentation without recognition at the same time. The problem of symbol spotting in documents for real world situation is more difficult as the documents often suffer from different noises (see fig. 1) resulted in scanning, vectorization, superimposition of the graphic and textual parts. Spotting methods are usually queried by example i.e. the user segments the item he wants to retrieve from the database and this cropped image acts as input of the system. This implies the infinite possibility of the query symbols, which prevents explicit trainings within the spotting architecture. Symbol spotting is highly applicable for real time indexing and retrieval of the dataset containing graphical documents, which demands high efficiency of the method in terms of computation.

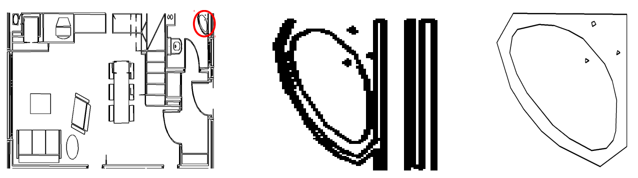
(a) (b) (c)

Figure 1. (a) A floor plan from the FPLAN-POLY database, (b) Zoomed portion of the selected part (within the red circle) in fig. 1a shows the difficulty due to noise, (c) Actual instance of the symbol shown in fig. 1b.

Several attempts have been made for spotting symbols in graphical documents [1]. The algorithm proposed by Messmer and Bunke [2] is among the first few approaches. Graph based methods [3] are also popular, but they often suffer from computational complexity. Among the others, Rusiñol et al. have used a technique of splitting the symbols into several primitives and used region strings [4] and off-the-shelf shape descriptors [5] to represent them. Nguyen et al. [6] used a visual vocabulary built on SCIP descriptors for symbol spotting. Recently Nayef and Breuel [7] proposed a branch and bound algorithm for spotting symbols in documents, where they used geometric primitives as features.

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. On the other hand, subgraph isomorphism is proved to be a NP-hard problem, so handling a large collection of graphical documents using graphs is difficult. In this work we use graph representation in architectural drawing images due to their robust structural description. Our representation considers the critical points detected by the vectorization method as the nodes and the lines joining them as the edges. To avoid the computational burden we propose a method based on the factorization of graphs. The factorization is done by splitting the graphs into a set of all acyclic paths (Hamiltonian paths) between each pair of connected nodes; the paths carry the geometrical information of a structure as attributes. The factorization creates an unified representation of the whole database and at the same time it allows robust detection with certain tolerance to noise and distortion. This also eases the segmentation free recognition which is important for our purpose. In our work, the shape descriptors of paths are compiled in hash tables by the locality-sensitive hashing (LSH) algorithm [8], [9]. The hashing data structure aims to organize the similar paths in the same neighborhood in hash tables and LSH is also proved to perform approximate k-NN search in a sub-linear time. The spotting of the query symbol is then done by a spatial voting scheme, which is formulated in terms of the selected paths from the database. This path selection is done by the approximate search mechanism during the hash table lookup procedure of the paths that compose the query symbol.

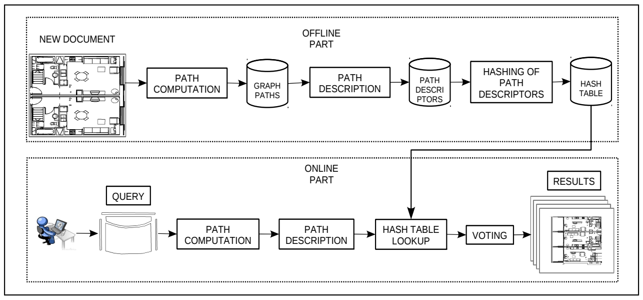

Figure 2. Symbol spotting framework for our method.

Our entire framework can be broadly divided into two parts viz. offline and online (see fig. 2). The offline part includes the computation of all the acyclic graph paths in the database, description of those paths with some off-the-shelf descriptors and hashing of those descriptors using the LSH algorithm (see fig. 3). Each time a new document is being included in the database the entire offline procedure is repeated to create the updated hash table. For each of the documents in the database all the computed paths and their descriptors are stored to reduce the further computation time. On the other hand, the online part includes the querying of the graphic symbol by an end user, computation of all the acyclic paths for that symbol, description of them, hash table lookup for each of the paths in the symbol and a voting scheme which is based on the similarity measure of the paths. The framework is designed to produce a ranked list of retrievals in which the true positive should appear first. The ranking is done based on the total vote values (see Section III) obtained by each retrieval.

The rest of the paper is organized into four sections. In Section II we present the methodology to represent a database in terms of the descriptors of graph paths. Section III describes the spotting method of the system. Section IV contains the detailed experimental results. After that in Section V, we conclude the paper.

II. Graph representation and spotting architecture

We use Zernike moments to describe the graph paths. Zernike moments have been widely utilized in pattern or object recognition, image reconstruction, content-based image retrieval etc. but its direct computation takes huge time. Hence several algorithms have been proposed to speed up the accurate computation process. For line drawings, Lambert and Gao [10] also formulated Zernike moments as computationally efficient line moments.

In order to avoid one-to-one path matching [11], we use the LSH algorithm which performs an approximate k-NN search that efficiently results in a set of candidates that mostly lie in the neighborhood of the query point (path). The LSH was introduced by Indyk and Motwani [8] and later modified by Gionis et al. [9]. It is proved to perform approximate k-NN search in sub-linear time and used for many real time computer vision applications.

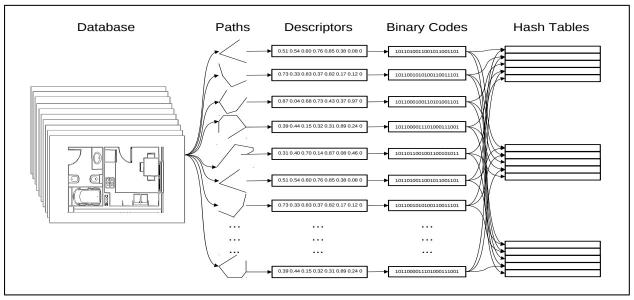

Figure 3. Hashing of paths provokes collisions in hash tables.

Let $G_i = (V_i, E_i)$, $i = 1, ..., n$ be the set of graphs, each related to one document in a database. Let E^* be the set of all acyclic paths in the database and $ZK(p) = (x_1, ..., x_d) \in \Re^d$ be the Zernike moments descriptors of a graph path $p \in E^*$. This point in the d-dimensional space is transformed in a binary vector space by the following function:

$$v(ZK(p)) = (Unary_C(x_1), ..., Unary_C(x_d)) \quad (1)$$

Here $Unary_C(x)$ denotes the unary representation function of x, which results in a binary vector with x number of 1s followed by $C - x$ number of 0s, where C is the highest coordinate value in the Zernike moments space. Thus the distance between two Zernike moments vectors $ZK(p_1)$, $ZK(p_2)$ can then be computed by the Hamming distance between their respective binary representations $v(ZK(p_1))$, $v(ZK(p_2))$. Actually the eqn. (1) allows to embed ZKs into the Hamming cube $H^{d'}$ of dimension $d' = Cd$. The construction of the function in eqn. (1) assumes the positive integer coordinates of ZK, but clearly any coordinates can be made positive by proper translation in $\Re^d$. Also the coordinates can be converted to an integer by multiplying them with a suitably large number and rounding to the nearest integers.

Now let $\mathcal{F}$ be the set of all hash functions $g(x)$ that project the binary points $v(ZK(p))$ to one of the d' coordinates. The set $\mathcal{F}$ can be defined as $\mathcal{F} = \left\{g(x) : \{0,1\}^{d'} \rightarrow \{0,1\} \,|g(x) = x_i, i = 1, ..., d'\right\}$, where x_i is the ith coordinate of x. The final sets of hash functions Gs can be created by randomly selecting at most K such hash functions $g(x)$ and concatenating them sequentially. This actually results in the bucket indices in the hash tables. The LSH algorithm then creates a set $\mathcal{T}$ of L hash tables, each of which is constructed based on different G_i, $(i = 1, ..., L)$. L and K are considered as the parameters to construct the hashing data structures. Now given a query point q the algorithm iterates over all the hash tables in $\mathcal{T}$ retrieving the data points that are hashed into the same bucket. The final list of retrievals is the union of all such matched buckets from different hash tables.

The entire procedure can be better understood with the following example: let ZK_1=$(1, 6, 5)$, ZK_2=$(3, 5, 2)$ and ZK_3=$(2, 4, 3)$ be three different descriptors in a three-dimensional (d=3) space with C=6. Their binary representation after applying the function in eqn. (1) is:

$$v(ZK_1) = 100000\,111111\,111110$$
$$v(ZK_2) = 111000\,111110\,110000$$
$$v(ZK_3) = 110000\,111100\,111000$$

Now lets create a LSH data structure with $L = 3$ and $K = 5$. So we can randomly create 3 hash functions with at most 5 bits in each of them as follows:

$$G_1 = \{g_5, g_{10}, g_{16}\}$$
$$G_2 = \{g_1, g_9, g_{14}, g_{15}, g_{17}\}$$
$$G_3 = \{g_4, g_8, g_{13}, g_{18}\}$$

This defines which components of the binary vector to be considered to create the hash bucket index. For example, applying G_2 to a binary vector results in a binary index concatenating the first, ninth, fourteenth, fifteenth and seventeenth bit values respectively. After applying the above functions to our data we obtain the following bucket indices:

$$G_1(ZK_1) = 011,\ G_2(ZK_1) = 11111,\ G_3(ZK_1) = 0110$$
$$G_1(ZK_2) = 010,\ G_2(ZK_2) = 11100,\ G_3(ZK_2) = 0110$$
$$G_1(ZK_3) = 010,\ G_2(ZK_3) = 11110,\ G_3(ZK_3) = 0110$$

Then for a query $ZK_q = (3, 4, 5)$ we have

$$v(ZK_q) = 111000\,111100\,111110$$
$$G_1(ZK_q) = 011,\ G_2(ZK_q) = 11111,\ G_3(ZK_q) = 0110$$

Thus we obtain ZK_1 as the nearest descriptor to the query since it collides in each of the hash tables.

Similarly, for each of the graph path descriptors in the query symbol we get a set of paths that belong in the database, consequently we get the similarity distances of the paths in the vectorial space. This similarity distance is useful during the voting procedure to spot the symbol and is used to calculate the vote values.

III. Retrieval process

A voting space is defined over each of the images in the database dividing them into grids of three different sizes (10×10, 20×20 and 30×30). Multiresolution grids are used to detect the symbols accurately within the image and the sizes of them are experimentally determined for the best performance. That is, we tested the results with five different grid sizes starting from 10×10 to 50×50 in different combination and obtained best results combining the above mentioned grid sizes. For a particular model path, we select the best matching paths from the hash table by the LSH lookup procedure. For each of the selected paths we accumulate the votes to the 9 neighboring grids of each of the 2 terminal vertices of that path. The vote to a particular grid is inversely proportional to the path distance metric (in this case the Euclidean distance between the Zernike moments descriptors) and is weighted by the Euclidean distance to the centers of the respective grids from the terminal of the selected path. The grids constituting the higher peaks are filtered by the k-means algorithm applied in the voting space with k=2. Finally, the occurrences of the query symbol on the documents are detected by another hierarchical clustering algorithm which clusters the spatial points contributed from all the three grids considered. The total vote values of the grids in each cluster are considered for ranking the retrievals.

IV. Results and Discussion

In order to evaluate the proposed spotting methodology, we present three different experiments. The first experiment is designed to test the method on the images of real world floor plans. The second experiment is done to check the algorithm on moderately large dataset which is a synthetically created benchmark. The last experiments is done to test the efficiency of the method on the images of handwritten sketch like floor plans.

The set of available query symbols for each datasets are used as query to evaluate with the ground truths. For each of the symbols the performance of the algorithm is evaluated in terms of precision (**P**), recall (**R**) and average precision (**AveP**). The interested readers are referred to [12] for the definition of the previously mentioned metrics for symbol spotting problem. To have an idea about the computation time we calculate the per document retrieval time (**T**) for each of the symbols. For each of the datasets the mean of

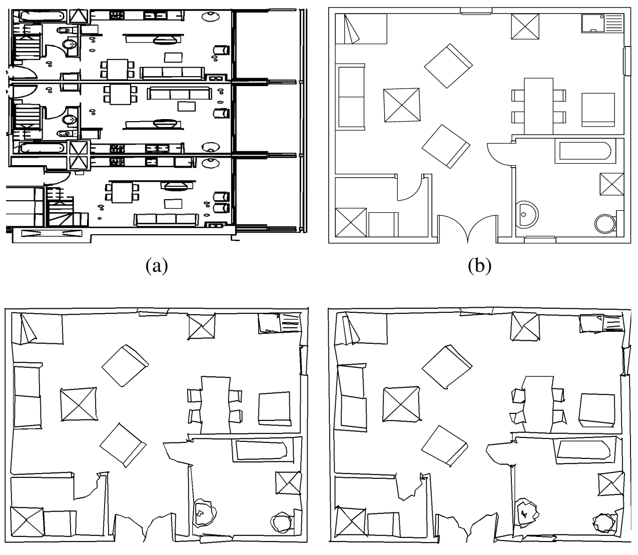

Figure 4. Examples of floor plans from different dataset. (a) A floor plan from FPLAN-POLY dataset, (b) A floor plan from SESYD dataset, (c) The same floor plan in fig. 4b degraded with vectorial noise r=10, (d) The same floor plan in fig. 4b degraded with vectorial noise r=15.

the above mentioned metrics are shown to judge the overall performance of the algorithm.

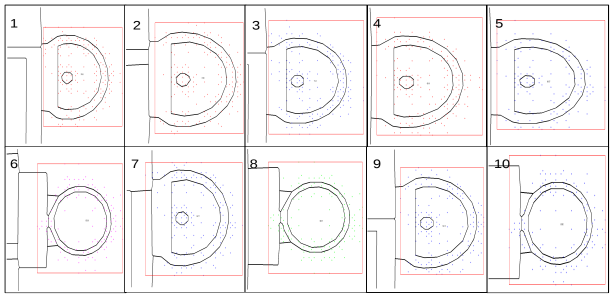

Figure 5. Qualitative results of the method, first 10 retrieved regions obtained by querying the symbol in fig. 6e in the SESYD (floorplans16-01) dataset.

All the experiments described below are done with the Zernike moments descriptors with order 7 (dimension d=36). For LSH the hashing data structures are created with L=10 and K=60. These parameters are experimentally decided to give the best performance. LSH reduces the search space significantly, for example SESYD (floorplans16-01) consists of approximately 1,465,000 paths and after lookup table construction these paths store in 16,000 buckets, so compared to one to one path comparison the search space reduces by 90 times. The time taken to create the hash tables for each of the database is dependent on the number of paths each database contains and also on the parameters L, K, the average time taken is 3 minutes 41 secs (approx.).

Table I
RESULTS WITH SESYD DATASET

Database	P	R	AveP	T
floorplans16-01	41.33	83.56	52.46	0.07
floorplans16-02	52.07	82.90	56.17	0.09
floorplans16-03	55.55	86.42	71.19	0.07
floorplans16-04	61.31	75.02	65.89	0.05
floorplans16-05	62.05	92.57	67.79	0.08
floorplans16-06	53.50	79.81	60.67	0.07
floorplans16-07	69.38	84.85	65.34	0.07
floorplans16-08	56.54	91.09	58.15	0.08
floorplans16-09	59.72	78.67	47.68	0.07
floorplans16-10	57.76	84.76	63.39	0.08
mean	**56.92**	**83.96**	**60.87**	**0.07**

A. Experiment on FPLAN-POLY

We have tested our method with the FPLAN-POLY dataset described in [5]. This dataset is a collection of 42 real floor plans (for example see fig. 4a) and 38 cropped symbols as the queries. The datasets are available in the vectorized form and the vectorization is done by the Qgar[1] software. This experimentation is done to show the efficiency of the algorithm in real images, which could suffer from the noise introduced in the scanning process, vectorization etc.

The recall rate achieved by the method is 93.43% which shows efficiency of the algorithm in retrieving the symbols. The average precision obtained by the method is 79.52% which ensures the occupancy of the true positives in the beginning of the ranked retrieval list. The precision value of the method is 77.87% which is also quite good for any retrieval method. Also the method is efficient in terms of time complexity since the average time taken to spot a symbol per document is 0.18 sec.

B. Experiment on SESYD

We have also tested our method in the SESYD (floorplans) [13] dataset. This dataset contains 10 different sub datasets, each of which contains 100 different synthetically generated floor plans (for example see fig. 4b). All the floor plans in a sub datasets are created upon a same floor plan template by putting different model symbols in different places in random orientations and scales. This experimentation is designed to test scalability of the algorithm i.e. to check the performance of the method in a sufficiently large dataset.

The mean of different measurements for each of the datasets are shown in table I. The recall values for all the datasets are quite good, although the average precisions are lesser than the previous experiments. This is due to the existence of the similar substructures (graph paths) within different symbols, for example, between the symbols in fig. 6c and fig. 6d; between the symbols in fig. 6f and fig. 6g; among the symbols in fig. 6a, 6b, 6i and 6k. These similarities harm the vote values considered for ranking the retrievals. There is an interesting observations regarding the average time taken for the retrieval procedure, which is

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

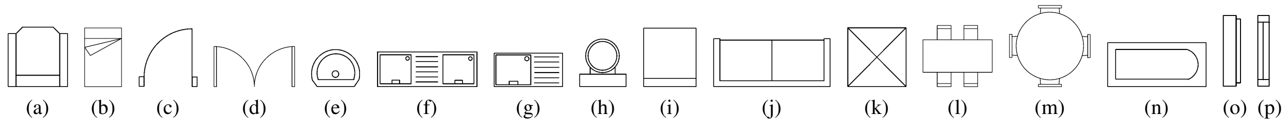
Figure 6. Model symbols in the SESYD dataset.

Table II
RESULTS WITH SESYD-VN DATASET

Radius (r)	P	R	AveP	T
r=5	63.64	92.19	65.27	0.25
r=10	47.49	87.01	56.82	0.26
r=15	34.37	82.16	47.80	0.25

0.07 sec. to retrieve a symbol per document image, that is much lesser than the previous experiment. This is due to the hashing technique which allows the collision of same structural elements and insert them into the same buckets. So even though the search space increases, due to hashing of the graph paths it remains nearly constant for each of the model symbols. This ultimately reduces the per document retrieval time. To get an idea about the performance of the method we present a qualitative results with first 10 retrievals on the SESYD (floorplans16-01) dataset (see fig.5).

C. Experiment on SESYD-VN

Finally, the last experiment is done to test the effectiveness of the algorithm on the hand-drawn sketch like floor plans. For this we select one of the 16 sub datasets of SESYD foorplans and introduce vectorial noise with different levels (for example see fig. 4c, 4d). The vectorial noise is created by randomly shifting the primitive points (critical points detected by the vectorization process) within a circle of radius r. We vary r to get different level of vectorial distortions. For this experiment we have created 3 levels of difficulties (for r = 5, 10, 15). For all the different distortions the same ideal model symbols as in the previous experiments are used as queries. So this experiment also shows the robustness of the method in capturing invariability due to noise.

The results are shown in table II. The recall value for the dataset with minimum distortion $(r = 5)$ is quite good but it decreases with the increment of distortion. The same incident is observed for average precision also. The distortion also introduces many false positives which harms the precision. In this experiment the per document retrieval time of model symbols increases than the previous experiment. This is due to the increment of randomness in the factorized graph paths which decreases the similarity among them. This compels the hashing technique to create large number of buckets and hence ultimately increases the per document retrieval time.

V. CONCLUSIONS

In this paper we have proposed a symbol spotting technique through hashing the shape descriptors of the graph paths. Graphs are very useful in representing graphical documents specially the line drawings but they are computationally expensive. To reduce the computational burden we factorize the graphs into paths, which also helps to incorporate a noise model for symbol spotting. The hashing of the graph paths aims to reduce the search space. We use LSH to perform an approximate k-NN search in sub-linear time. The symbol spotting is then done by a spatial voting procedure. We have tested the method on various noisy datasets and the results are encouraging.

ACKNOWLEDGMENT

This work has been partially supported by the Spanish projects TIN2009-14633-C03-03, TIN2008-04998, CSD2007-00018 and the PhD scholarship 2011FI_B 01022.

REFERENCES

[1] K. Tombre and B. Lamiroy, "Pattern recognition methods for querying and browsing technical documentation," in *Proceedings of 13th CIARP*, LNCS, vol. 5197, pp. 504–518, 2008.
[2] B. T. Messmer and H. Bunke, "A new algorithm for error-tolerant subgraph isomorphism detection," in *IEEE TPAMI*, vol. 20, pp. 493–504, 1998.
[3] J. Lladós, E. Martí, and J. J. Villanueva, "Symbol recognition by error-tolerant subgraph matching between region adjacency graphs," in *IEEE TPAMI*, vol. 23, pp. 1137–1143, 2001.
[4] M. Rusiñol, J. Lladós, and G. Sánchez, "Symbol spotting in vectorized technical drawings through a lookup table of region strings," in *PAA*, vol. 13, pp. 1–11, 2009.
[5] M. Rusiñol, A. Borràs, and J. Lladós, "Relational indexing of vectorial primitives for symbol spotting in line-drawing images," in *PRL*, vol. 31, pp. 188–201, 2010.
[6] T.-O. Nguyen, S. Tabbone, and A. Boucher, "A symbol spotting approach based on the vector model and a visual vocabulary," in *Proceedings of 10th ICDAR*, pp. 708–712, 2009.
[7] N. Nayef and T. M. Breuel, "A branch and bound algorithm for graphical symbol recognition in document images," in *Proceedings of 9th DAS*, pp. 543–546, 2010.
[8] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proceedings of 13th ACM STOC*, pp. 604–613, 1998.
[9] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proceedings of 25th ICVLDB*, pp. 518–529, 1999.
[10] G. Lambert and H. Gao, "Line moments and invariants for real time processing of vectorized contour data," in *IAP*, LNCS, vol. 974, pp. 347–352, 1999
[11] A. Dutta, J. Lladós, and U. Pal, "A bag-of-paths based serialized subgraph matching for symbol spotting in line drawings," in *Proceedings of 5th IbPRIA*, LNCS, vol. 6669, pp. 620–627, 2011.
[12] M. Rusiñol and J. Lladós, "A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices," in *IJDAR*, vol. 12, pp. 83–96, 2009.
[13] M. Delalandre, T. Pridmore, E. Valveny, H. Locteau, and E. Trupin, "Building Synthetic Graphical Documents for Performance Evaluation", in LNCS, vol. 5046, pp. 288–298, 2008.