A symbol spotting approach in graphical documents by hashing serialized graphs

Anjan Dutta[a,*], Josep Lladós[a], Umapada Pal[b]

[a] *Computer Vision Center, Universitat Autònoma de Barcelona, Edifici O, Campus UAB, 08193 Bellatera, Barcelona, Spain*

[b] *Computer Vision and Pattern Recognition Unit, Indian Statistical Institute, 203, B.T.Road, Kolkata-108, India*

Abstract

In this paper we propose a symbol spotting technique in graphical documents. Graphs are used to represent the documents and a (sub)graph matching technique is used to detect the symbols in them. We propose a graph serialization to reduce the usual computational complexity of graph matching. Serialization of graphs is performed by computing acyclic graph paths between each pair of connected nodes. Graph paths are one dimensional structures of graphs which are less expensive in terms of computation. At the same time they enable robust localization even in the presence of noise and distortion. Indexing in large graph databases involves a computational burden as well. We propose a graph factorization approach to tackle this problem. Factorization is intended to create a unified indexed structure over the database of graphical documents. Once graph paths are extracted, the entire database of graphical documents is indexed in hash tables by 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. We have performed detailed experiments with various datasets of line drawings and compared our method with the state-of-the-art works. The results demonstrate the effectiveness and efficiency of our technique.

Keywords: Symbol spotting, Graphics recognition, Graph matching, Graph serialization, Graph factorization, Graph paths, Hashing.

*Corresponding author

Email addresses: adutta@cvc.uab.es (Anjan Dutta), josep@cvc.uab.es (Josep Lladós), umapada@isical.ac.in (Umapada Pal)

1. Introduction

Even after the significant advancements in the present digital era, paper documents still make an important contribution in our regular work-flows. Digitization of documents is justified on the basis of portability and preservation issues. However, developing a system for browsing and querying digital documents in an effective way still remains a big challenge. So, efficient indexing mechanisms which organize the information extracted by the analysis of document images are essential in order to improve accessibility to these large collections of digital documents. Indexing and retrieval of textual documents involves the conversion of the printed text image into ASCII characters using OCR as the first step. This facilitates the retrieval and querying of information in the document image by textual queries. However nowadays there are some trends to handle textual documents without explicitly recognizing it by OCR [1]. This is either due to reasons of complexity, or all the information in a document can not be represented by typewritten characters. One benefit to be noted about textual data is its single dimensionality that may be sorted, which is not available for graphical objects because of their bi-dimensional nature.

Information spotting is a major branch of indexing and retrieval methods. It can be defined as locating given query information in a large collection of relevant data. In document analysis, the research community is mainly focused on word spotting for textual documents [1, 2] and symbol spotting for graphic-rich documents [3, 4]. Here it is posited that the textual information can also be given a symbolic representation and approached by a symbol spotting technique. In this work we have concentrated on symbol spotting in graphical documents. Architectural line drawings are used as an experimental framework. Symbol spotting can be defined as the identification of a set of regions of interest from document images which are likely to contain an instance of a certain queried symbol using an inexpensive method. Example applications of symbol spotting include 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 can be considered a variant of content based image retrieval (CBIR) applications.

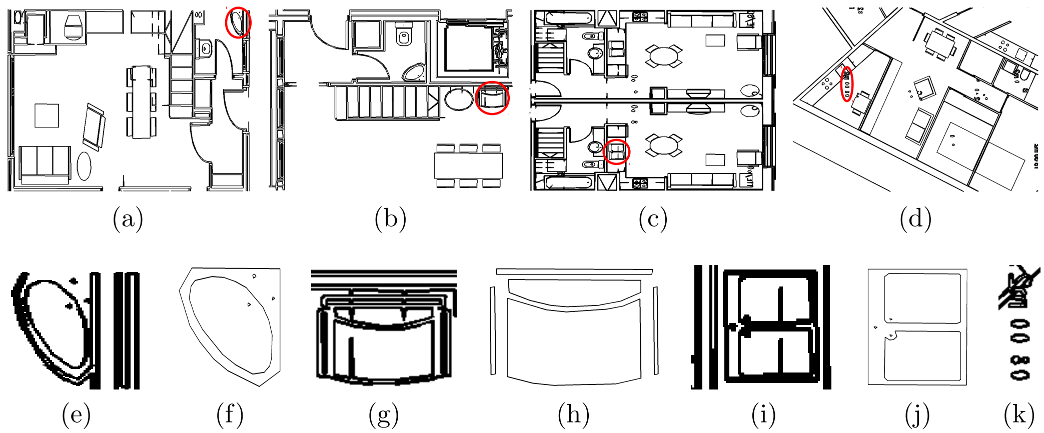


Figure 1: (a)-(d) Examples of floorplans from a real floorplan (FPLAN-POLY) database, (e),(g),(i),(k) Zoomed portions of the selected parts respectively shown in Figures 1a-1d show the difficulty of recognition due to noise and superimposition of textual and graphical information, (f),(h),(j) Actual instances of the symbols shown in (e),(g),(i) respectively.

The main differences are that CBIR approaches retrieve the atomic images on a large scale leaving the user with the task of locating the real relevant information within the provided results, whereas symbol spotting methodologies give direct access to the relevant information. Such applications that return direct passages of interests within documents instead of complete documents, are known as *focused retrieval* systems [5]. In short, CBIR methods can be defined as undertaking image to image matching, whereas focused retrieval is more similar to image to region of interest (ROI) or object location searching.

Symbol spotting follows the segmentation-recognition paradigm, that is a symbol spotting architecture does not use a previous segmentation step followed by a proper recognition method, instead it proceeds 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 are arbitrarily oriented and often suffer from noise (see Figure 1) resulting from scanning, vectorization, superimposition of the graphic and textual parts etc. Spotting methods are usually queried by example i.e. the user segments the item to retrieve from the database and this cropped image acts as input of the system. This implies infinite possibilities of the query symbols, which prevents explicit training within the spotting architecture. Symbol spotting is highly applicable for real-time indexing and retrieval from a dataset containing graphical documents, which demands high efficiency of the method in terms of computation.

Graphs are very suitable data structures to represent graphical documents, especially line drawings. They allow to capture structural properties of points, lines, junctions, regions etc. For that reason they have been widely chosen by the research community as the basic tool to represent graphical structures [6]. Hence, in line drawings represented by graphs, the problem of symbol spotting can then be formulated as a subgraph matching problem, where graph theory offers robust approaches to compute it efficiently. In this paper we propose a symbol spotting technique based on a graph representation of graphical documents, especially various kinds of line drawings. When graphs are attributed by geometric information, this also supports various affine transformations viz. translation, rotation, scaling etc. In this work, our representation considers the critical points detected by the vectorization method [7] as the nodes and the lines joining them as the edges. On the other hand, subgraph isomorphism is proved to be a NP-hard problem [8], so handling a large collection of graphical documents using graphs is difficult. To avoid computational burden, we propose a method based on the factorization of graphs. Informally, *graph factorization* can be defined as the method to extract graph sub-structures from larger graphs. This is helpful to find common subgraph structures from larger collections of graphs to define indexing keys in terms of such common subgraphs. This indexing structure is supposed to reduce the search space by clustering similar subgraphs. In our case, factorization is performed 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 which are considered as attributes. The decomposition of a graph into graph paths can be seen as a *serialization* process where the complex two-dimensional graph structure is converted to a one-dimensional string to reduce computational complexity, usually present in subgraph matching algorithms. In this work we follow both factorization and serialization to create an inexpensive and unified structure. Graph factorization creates a unified representation of the whole database and at the same time it allows for robust detection with a certain tolerance to noise and distortion. This also eases the segmentation-free recognition which is important for our purpose.

In this work, the shape descriptors of paths are compiled into hash tables by the Locality-Sensitive Hashing (LSH) algorithm [9, 10]. The hashing data structure aims to organize similar paths in the same neighborhood into hash tables. The spotting of the query symbol is then undertaken by a spatial voting scheme, which is formulated in terms of the selected paths from the database.

However, the graph paths are indexed independently, ignoring any spatial relationship between them. Actually keeping the spatial relationship is not important for us since we consider all the acyclic paths between each pair of connected nodes. Actually this fact better helps to incorporate the structural noise keeping the spatial relationship among paths. This way the spatial relationship is maintained as the smaller paths are always subpaths of some longer paths and longer paths contain more global structural information.

Since the method represents a database of graphical documents in terms of unified representation of factorized substructures, it can handle a larger database of documents which is important for real-world applications. Moreover, the factorized substructures allow the method to handle structural noise up to a certain limit of tolerance. The proposed method does not work with any kind of pre-segmentation and training, which makes it capable of handling any possible combination of query symbols.

The rest of the paper is outlined as follows: In Section 2 we survey the related work in the literated concerning symbol spotting. We present our proposed methodology in Section 3, followed by a series of experiments in Section 4. Section 5 concludes the paper with discussions on future works.

2. Related work

The focus of this paper is twofold. First, it concentrates on symbol spotting in graphical documents. Second, as a methodological contribution, we propose an efficient subgraph matching approach based on graph paths. In this section, we review the literature in both fields.

2.1. Symbol spotting

Nowadays symbol spotting has experienced a growing interest among the graphics recognition community. The major existing research can be classified into five broad families as in [11], which are listed in Table 1, and those families are reviewed as follows.

Hidden Markov Models (HMMs). HMMs are powerful tools to represent dynamic models which vary in terms of time or space. Their major advantage in space series classification results from their ability to align a pattern along their states using a probability density function (pdf) for each state, that estimates the probability of a certain part of the pattern belonging to the state. HMMs have been successfully applied for off-line handwriting recognition [12, 13], where the characters represent pattern changes in space whilst moving from left to right. Also, HMMs have been applied to the problems of image classification and shape recognition [14]. Müller and Rigoll [15] proposed pseudo 2-D HMMs to model the two-dimensional arrangements of symbolic objects. This is one of the first few approaches we can find for symbol spotting, where the document is first partitioned by a fixed sized grid. Then each small cell acts as an input to a trained 2-dimensional HMM to identify the locations where the symbols from the model database is likely to be found. Previously, HMMs were also applied to word spotting, and this work is an adaptation of HMMs for 2D shapes. The method does not need pre-segmentation, and also it could be used in noisy or occluded conditions, but since it depends on the training of a HMM, it loses one of the main assumptions of symbol spotting methodologies.

Graph-based approaches. The methods based on graphs rely on the structural representation of graphical objects and propose (sub)graph matching techniques to spot symbols in the documents. Graph matching can be solved with a structural matching approach in the graph domain or solved by a statistical classifier in the embedded vector space of the graphs. In both cases these techniques include an error model which allows inexact graph matching to tolerate structural noise in documents. There are an adequate number of methods based on graphs [16–24]. In general the structural properties of the graphical entities are encoded in terms of attributed graphs and then a subgraph matching algorithm is proposed to localize or recognize the symbol in the document in a single step. The (sub)graph matching algorithms conceive some noise models to incorporate image distortion, which is defined as inexact (sub)graph matching. Since (sub)graph matching is an NP-hard problem [8], these algorithms often suffer from a huge computational burden. Among the methods available, Messmer and Bunke in [16] represented graphic symbols and line drawings by Attributed Relational Graphs (ARG). Then the recognition process of the drawings was undertaken in terms of error-tolerant subgraph isomorphisms from the query symbol graph to the

Table 1: Different families of symbol spotting research with their advantages and disadvantages.

Family	Method	Advantages	Disadvantages
HMM	[15]	segmentation-free; Robust in noise	Needs training
Graph based	[16–24]	Simultaneous symbol segmentation and recognition	Computationally expensive
Raster features	[25, 26]	Robust symbol representation; Computationally fast	Ad-hoc selection of regions; Inefficient for binary images
Symbol signatures	[27, 28]	Simple symbol description; Computationally fast	Prone to noise
Hierarchial symbol representation	[29]	Linear matching is avoided by using an indexing technique	Dendogram structure is strongly dependent on the merging criterion.

drawing graph. Lladós et al. in [17] proposed Region Adjacency Graphs (RAG) to recognize symbols in hand drawn diagrams. They represented the regions in the diagrams by polylines where a set of edit operations is defined to measure the similarity between the cyclic attributed strings corresponding to the polylines. In [18], Barbu et al. presented a method based on frequent subgraph discovery with some rules among the discovered subgraphs. Their main application is the indexing of different graphical documents based on the occurrence of symbols. Qureshi et al. [19] proposed a two-stage method for symbol recognition in graphical documents. In the first stage the method only creates an attributed graph from the line drawing images and in the second stage the graph is used to spot interesting parts of the image that potentially correspond to symbols. Then in the recognition phase each of the cropped portions from the images are passed to an error tolerant graph matching algorithm to find the queried symbols. Here the procedure of finding the probable regions restricts the method only to work for some specific symbols, which violates the assumption of symbol spotting. Locteau et al. [20] present a symbol spotting methodology based on a visibility graph. There they apply a clique detection method, which corresponds to a perceptual grouping of primitives to detect regions of particular interest. In [21] Rusiñol et al. proposed a symbol spotting method based on the decomposition of line drawings into primitives of closed regions. An efficient indexing methodology was used to organize the attributed strings of primitives. Nayef and Breuel [23] proposed a branch and bound algorithm for spotting symbols in documents, where they used geometric primitives as features. Recently Luqman et al. [22] also proposed a method based on fuzzy graph embedding for symbol spotting, a priori they also used one pre-segmentation technique as in [19] to get the probable regions of interest which may contain the graphic symbols. Subsequently, these ROIs are then converted to fuzzy structural signatures to find out the regions that contain a symbol similar to the queried one. At last, very recently, Le Bodic et al. [24] proposed substitution-tolerant subgraph isomorphism to solve symbol spotting in technical drawings. They represent the graphical documents with RAG and model the subgraph isomorphism as an optimization problem. The whole procedure is performed for each pair of query and document. Moreover, since the method works with RAG, it is not efficient for the symbols having open regions (for example, Figure 12c,12d) or regions with discontinuous boundary.

Raster features. Some of the methods work with low-level pixel features for spotting symbols. To reduce the computational burden they extract the feature descriptors on some regions of the documents. These regions may come from a sliding window or spatial interest point detectors. These kinds of pixel features robustly represent the region of interest. Apart from those methods mentioned, other methods find some probable regions for symbols by examining the loop structures [19] or just use a text/graphic separation to estimate the occurrence of the symbols [25]. After ad-hoc segmentation, global pixel-based statistical descriptors [25, 26] are computed at each of the locations in sequential order and compared with the model symbols. A distance metric is also used to decide the retrieval ranks and to check whether the retrievals are relevant or not. The one-to-one feature matching is a clear limitation of this kind of methods and also the ad-hoc segmentation step only allows it to work for a limited set of symbols.

Symbol signatures. Like the previous category, this group of methods [27, 28, 30] also works with ad-hoc segmentation, but instead of pixel features they compute the vectorial signatures, which better represent the structural properties of the symbolic objects. Here vectorial signatures are the combination of simple features viz. number of graph nodes, relative lengths of graph edges etc. These methods are built on the assumptions that the symbols always fall into a region of interest and compute the vectorial signatures inside those regions. Since symbol signatures are highly affected by image noise, these methods do not work well in real-world applications.

Hierarchial symbol representation. Some of the methods [29] work with the hierarchical definition of symbols, in which they hierarchically decompose the symbols and organize the symbols' parts in a network or dendogram structure. Mainly, the symbols are split at the junction points and each of the subparts are described by a proprietary shape descriptor. These subparts are again merged by a measure of density, building the dendogram structure. Then the network structures are traversed in order to find the regions of interests of the polylines where the query symbol is likely to appear.

To conclude the literature review, some of the challenges of symbol spotting can be highlighted from the above state-of-the-art reviews. First, symbol spotting is concerned with various graphical documents viz. electronic documents, architectural floorplans etc., which in reality suffer from noise that may come from various sources such as low-level image processing, intervention of text, etc. So efficiently handling structural noise is crucial for symbol spotting in documents. Second, an example application of symbol spotting is to find any symbolic object from a large amount of documents. Hence, the method should be efficient enough to handle a huge database. Third, symbol spotting is usually invoked by querying a cropping symbol from some document, which acts as an input query to the system. So it implies infinite possibilities of the query symbols, and indirectly restricts the possibility of training in the system. Finally, since symbol spotting is related to real-time applications, the method should have a low computational complexity. We chose these five important aspects (segmentation, robustness in noise, training free, computational expenses, robustness with a large database) of symbol spotting to specify the advantages and disadvantages of the key research, which is listed in Table 2. The above literature review reveals the lack of solutions for addressing the above challenges altogether. This fact motivates us to propose a symbol spotting technique which can handle the above limitations of the existing methods.

Table 2: Comparison of the key works of symbol spotting.

Method	segmentation-free	Robust in noise	Training free	Computationally efficient	Robust with large database
Müller and Rigoll [15]	Yes	Yes	No	Yes	-
Messmer and Bunke [16]	Yes	-	-	No	No
Lladós et al. [17]	Yes	-	Yes	No	No
Barbu et al. [18]	Yes	-	Yes	No	No
Qureshi et al. [19]	No	-	Yes	No	No
Locteau et al. [20]	Yes	No	Yes	Yes	No
Rusiñol et al. [21]	Yes	-	Yes	No	Yes
Rusiñol et al. [31]	Yes	-	Yes	Yes	Yes
Tabbone et al. [25]	No	No	Yes	Yes	-
LeBodic et al. [24]	Yes	No	Yes	No	No
Our method	Yes	Yes	Yes	Yes	Yes

2.2. Graph matching approaches for symbol recognition

In addition to the above state-of-the-art of symbol spotting research, since our work is concerned with graph representation and matching, we would like to mention some of the key works in the area of graph matching, which are very relevant to our work. In general, graph matching has a long list of methods applied to various kinds of pattern recognition techniques. The interested reader is referred to [6] for more details. In the literature there are approaches to reduce the computational complexity of graph based methods and graph serialization is one of them. Serialization aims to reduce the computational complexity of expensive graph matching methods, for that reason it is often used for many computer vision problems [32–34]. All this research is based on the matching of strings which are often extracted from the graph representing the images, objects etc. The factorization of graphs into graph paths creates a one-dimensional structure of complex two-dimensional graphs and reduces the computational complexity. Originally, the factorized substructures of graphs are often used to represent bigger graphs in graph kernels [35]. Even the idea of graph paths is already used as a graph kernel in [36, 37] and it also simulates the idea of a random walk in a graph structure.

The above facts motivate us to work on serialization of graphs.

3. Proposed method

Our graph representation considers the critical points detected by the vectorization method as the nodes and the lines joining them as the edges. For our purpose we use the vectorization algorithm proposed by Rosin and West [7]. To avoid the computational burden we propose a method based on the factorization of graphs. The factorization is performed 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 helps to create 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. We have already mentioned that factorization of graphs is used in kernel based methods and it's principle motive was to cope with distortions. But the kernel based method can not utilize the power of indexation which is important for our case as we concentrate in spotting symbols in bigger datasets efficiently. So indexing the serialized subgraphical structures is a crucial part for our application. Our method takes the advantage of the error tolerance as proposed by the kernel based methods and at the same time the advantage of the indexation strategy to make the searching efficient. In our work, the shape descriptors of paths are compiled in hash tables by the Locality-Sensitive Hashing (LSH) algorithm [9, 10]. The hashing data structure aims to organize similar paths in the same neighborhood in hash tables and LSH is also proved to perform an approximate k-NN search in sub-linear time. The spotting of the query symbol is then performed by a spatial voting scheme, which is formulated in terms of the selected paths from the database. This path selection is performed by the approximate search mechanism during the hash table lookup procedure for the paths that compose the query symbol. The method is dependent on the overall structure of the paths. This technique is able to handle the existence of spurious nodes. And since we consider all the acyclic paths between each pair of connected nodes, the detection or recognition of a symbol is totally dependent on the overall structure of the majority of paths. This way the method is able to handle the problem of spurious nodes and edges. So the introduction of spurious edges and nodes only increases the computational time in the offline part without hampering the performance.

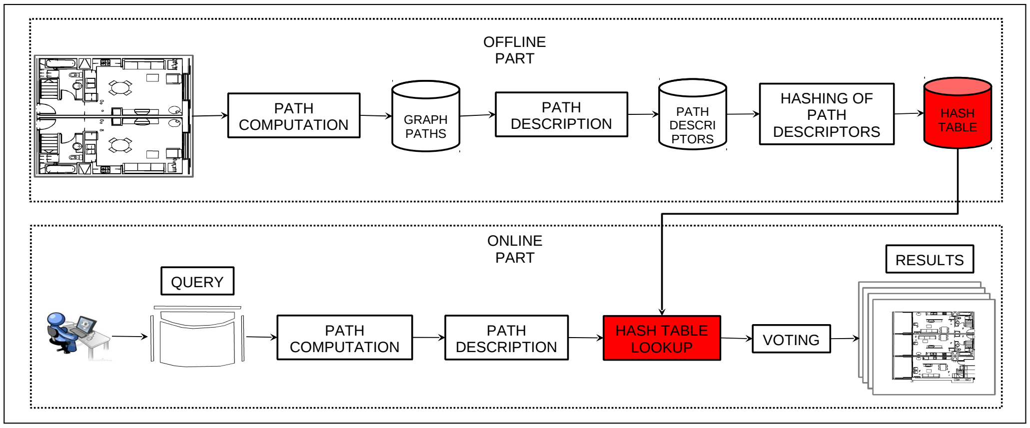


Figure 2: Symbol spotting framework for our method.

3.1. Framework

Our entire framework can be broadly divided into two parts viz. offline and online (see Figure 2). The algorithms are respectively shown in Algorithm 3.1 and Algorithm 3.2. The offline part (Algorithm 3.1) includes the computation of all the acyclic graph paths in the database, description of those paths with some proprietary descriptors and hashing of those descriptors using the LSH algorithm (see Figure 3). Each time a new document is included in the database, the offline steps for this document are repeated to update the hash table. To reduce the time complexity of the offline part the path and description information of the previously added documents are stored. On the other hand, the online part (Algorithm 3.2) includes the querying of the graphic symbol by an end user, the computation of all the acyclic paths for that symbol and description of them by the same method. Then a hash table lookup for each of the paths in the symbol and a voting procedure, which is based on the similarity measure of the paths, are also performed on the fly to undertake the spotting in the documents. The framework is designed to produce a ranked list of retrievals in which the true positive should appear first. The ranking is performed based on the total vote values (see subsection 3.4) obtained by each retrieval.

Let us now describe the key steps of our framework in the following subsections.

Algorithm 3.1 Hash table creation

Require: A set $Doc = \{D_1, \ldots, D_n\}$.

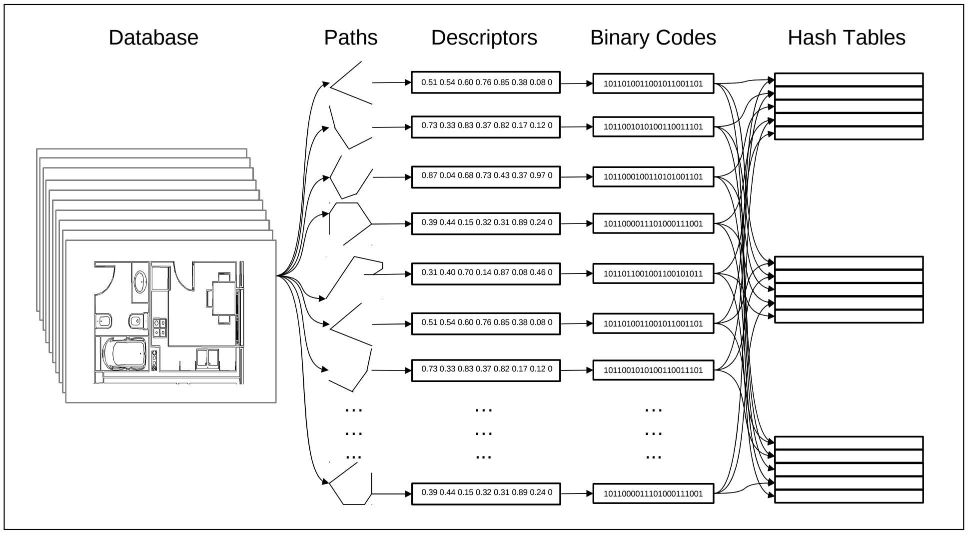


Figure 3: Hashing of paths provokes collisions in hash tables.

Ensure: A set $\mathcal{T}$ of hash tables.
  //Let f_{all} be the set of all path descriptors.
  //Initialize f_{all}
  $f_{all} \Leftarrow \oslash$
  for all D_i of Doc **do**
    $P_i \Leftarrow$ acyclic paths (D_i)
    for all p of P_i **do**
      $f \Leftarrow$ descriptors of (p) // Zernike moments or Hu moment invariants
      $f_{all} \Leftarrow f_{all} \cup f$
    end for
  end for
  //Create the set of hash tables
  $\mathcal{T} \Leftarrow \text{LSH}(f_{all})$

3.2. Path description

Let $Doc = \{D_1, D_2, ..., D_n\}$ be the set of all documents in a database, and $G_i = (V_i, E_i)$ be the graph for the document D_i.

Definition 1. *A* graph $G_i = (V_i, E_i)$ *is the ordered pair comprising of the set of vertices* V_i *and edges* E_i*, here the set* V_i *contains all the critical points detected by the vectorization method in a document and* E_i *contains the edges joining the vertices in the document.*

Our graphs are node-labeled and are denoted by $G_i = (V_i, E_i, L_v)$.

Definition 2. *Let Σ be the set of all labels for the nodes in a graph. A graph $G_i = (V_i, E_i)$ is called* node-labeled *and denoted as $G_i = (V_i, E_i, L_v)$, if there is a function $L_v : V_i \to \Sigma$, in this case $\Sigma = \mathbb{N}^2$, where the labels for each of the nodes is its position in terms of a two-dimensional coordinate system.*

Definition 3. *Given a graph $G_i = (V_i, E_i, L_v)$, a* graph path *p_k between two connected nodes v_r and v_s in the graph is defined as the ordered sequence of vertices $(v_r, ..., v_s)$ starting from v_r to v_s.*

Definition 4. *An* embedding *function f of a graph path is defined as a function $f : P \to \mathbb{R}^n$, defined in the space of a graph path converts a path to an n-dimensional feature space.*

Let $P_i = \{p_1, p_2, ..., p_{n_i}\}$ be the set of all graph paths in the document D_i, where n_i is the total number of paths the document D_i contains. Therefore $P = \cup_i P_i$ is the set of all paths from all the documents in Doc. From the definition of a graph path, a path p_k can be represented as an ordered sequence of nodes i.e. $p_k = [(x_1, y_1), (x_2, y_2), ...] = p_k(x, y)$. So formally speaking, given a path $p_k(x, y)$ and a shape descriptor $f : P \to \mathbb{R}^n$ defined over the space of all graph paths, applying f to each of the graph paths in P will generate a feature vector of dimension n. Below is the brief description of the shape descriptors used in this work. We define the embedding function f by means of Zernike moments and Hu moment invariants.

3.2.1. Embedding function based on Zernike moments

Zernike moments are robust shape descriptors which were first introduced in [38] using a set of complex polynomials. They are expressed as A_{mn} as follows:

$$A_{mn} = \frac{m+1}{\pi} \int_x \int_y p_k(x, y)[V_{mn}(x, y)]^* dxdy, \text{ where } x^2 + y^2 \leq 1 \quad (1)$$

where $m = 0, 1, 2, ..., \infty$ and defines the order, $p_k(x, y)$ is the path being described and $*$ denotes the complex conjugate. While n is an integer (that can be positive or negative) depicting the angular dependence, or rotation, subject to the conditions $m - |n| = even$, $|n| \leq m$ and $A^*_{mn} = A_{m,-n}$ is true.

The Zernike polynomials $V_{mn}(x, y)$ can be expressed in polar coordinates as follows:

$$V_{mn}(x, y) = V_{mn}(r, \theta) = \sum_{s=0}^{\frac{m-|n|}{2}} (-1)^s \frac{(m-s)!}{s!(\frac{m+|n|}{2} - s)!(\frac{m-|n|}{2} - s)!} exp(in\theta) \quad (2)$$

The final descriptor function $f_{Zernike}(p_k)$ for p_k is then constructed by concatenating several Zernike coefficients of the polynomials. Zernike moments have been widely utilized in pattern or object recognition, image reconstruction, content-based image retrieval etc. but its direct computation takes a large amount of time. Realizing this disadvantage, several algorithms [39] have been proposed to speed up the accurate computation process. For line drawings, Lambert et al. [40, 41] also formulated Zernike moments as computationally efficient line moments. But in our case the computation is performed based on the interpolated points of the vectorized data using fast accurate calculations.

3.2.2. Embedding function based on Hu moment invariants

The set of seven Hu invariants of moments proposed in [42] involving moments up to order three, are widely used as shape descriptors. In general the central $(r + s)$th order moment for a function $p_k(x, y)$ is calculated as follows:

$$\mu_{rs} = \sum_x \sum_y (x - \bar{x})^r (y - \bar{y})^s \quad (3)$$

The function $f_{Hu}(p_k)$ describing p_k is then constructed by concatenating the seven Hu invariants of the above central moments. The use of centroid $c = (\bar{x}, \bar{y})$ allow the descriptor to be translation invariant. A normalization by the object area is used to achieve invariance to scale. The geometric moments can also be computed on the contour of the objects by only considering the pixels of the boundary of the object. As in the case of Zernike moments, these moments can also be calculated in terms of line moments [40, 41] for the objects represented by vectorized contours, which are obviously efficient in terms of computation.

3.3. Locality sensitive hashing (LSH)

In order to avoid one-to-one path matching [43], 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). LSH is used to perform contraction of the search space and quick indexation of the data. LSH was introduced by Indyk and Motwani [9] and later modified by Gionis et al. [10]. It has been proved to perform an approximate k-NN search in sub-linear time and used for many real-time computer vision applications.

Let $f(p_k) = (f_1, ..., f_d) \in \mathbb{R}^d$ be the descriptors of a graph path p_k in the d-dimensional space. This point in the d-dimensional space is transformed in a binary vector space by the following function:

$$v(f(p_k)) = (Unary_C(f_1), ..., Unary_C(f_d)) \tag{4}$$

Here if C is the highest coordinate value in the path descriptor space then $Unary_C(f_p)$ is a $|C|$ bit representation function where $|f_p|$ bits of 1's are followed by $|C - f_p|$ bits of 0's. Thus, the distance between two path vectors $f(p_1)$, $f(p_2)$ can be computed by the Hamming distance between their respective binary representations $v(f(p_1))$, $v(f(p_2))$. Actually, eqn.(4) allows the embedding of the descriptors fs into the Hamming cube $H^{d'}$ of dimension $d' = Cd$. The construction of the function in eqn.(4) assumes the positive integer coordinates of f, but clearly any coordinates can be made positive by proper translation in $\mathbb{R}^d$. 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 $g : \{0,1\}^{d'} \rightarrow \{0,1\}$ be a function which projects a point $v \in \{0,1\}^{d'}$ to any of its d' coordinate axes, and $\mathcal{F}$ be a set of such hash functions $g(v)$, which can be formally defined as:

$$\mathcal{F} = \{g(v) | g(v) = v_i, i = 1, ..., d'\}$$

where v_i is the ith coordinate of v. The final set of hash functions Gs can be created by randomly selecting at most K such bitwise hash functions $g(v)$ and concatenating them sequentially. This actually results in 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 Gs. L and K are considered as the parameters to construct the hashing data structures. Then given a descriptor f_q of a query path (point), 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 $f_{p_1} = (1, 6, 5)$, $f_{p_2} = (3, 5, 2)$ and $f_{p_3} = (2, 4, 3)$ be three different descriptors in a three-dimensional ($d = 3$) space with $C = 6$. Their binary representation after applying the function in eqn. (4) is:

$$
\begin{aligned}
v(f_{p_1}) &= 100000\,111111\,111110 \\
v(f_{p_2}) &= 111000\,111110\,110000 \\
v(f_{p_3}) &= 110000\,111100\,111000
\end{aligned}
$$

Now let us create an 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:

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

This defines which components of the binary vector will 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:

$$
\begin{aligned}
G_1(f_{p_1}) = 011,\ G_2(f_{p_1}) = 11111,\ G_3(f_{p_1}) = 0110 \\
G_1(f_{p_2}) = 010,\ G_2(f_{p_2}) = 11100,\ G_3(f_{p_2}) = 0110 \\
G_1(f_{p_3}) = 010,\ G_2(f_{p_3}) = 11110,\ G_3(f_{p_3}) = 0110
\end{aligned}
$$

Then for a query $f_{p_q} = (3, 4, 5)$ we have

$$
\begin{aligned}
v(f_{p_q}) &= 111000\,111100\,111110 \\
G_1(f_{p_q}) = 011,\ G_2(f_{p_q}) &= 11111,\ G_3(f_{p_q}) = 0110
\end{aligned}
$$

Thus, we obtain f_{p_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 to 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.

3.4. Voting scheme

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 to have the best performance. It was mentioned earlier that the voting is performed in the online step of the system when the user query is accepted with a model symbol S_m. We factorize the graph representing S_m in the same way as the documents and let us say $P_{S_m} = p_1^{S_m}, ..., p_t^{S_m}$ be the set of all paths of S_m and $F_{S_m} = f_{p_1^{S_m}}, ..., f_{p_t^{S_m}}$ be the set of descriptors for the paths in P_{S_m}. The searching in the hash table is then performed in a path by path manner and consecutively the voting is performed in the image space. For a particular model path, $p_l^{S_m} \in P_{S_m}$, the LSH lookup procedure returns a union of several buckets (this is how the LSH is constructed). Let us say B_l be the union of all buckets returned when queried with a path $p_l^{S_m}$. In the next step, for each path $f_{p_{B_i}} \in B_l$ we accumulate the votes to the nine neighboring grids of each of the two terminals of $f_{p_{B_i}}$ (see Figure 4). 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 (in Figure 4 the centers of the grids are shown in red) 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. Here we only keep the cluster having the higher votes, all the higher voted points from all the three grids are then considered for spatial clustering. Here we compute the distances among all these points and use this distance matrix to cluster the points hierarchically. Here we use a threshold th_1 to cut the dendogram and have the clusters. The selection of th_1 is performed experimentally to give the best performance. Each of the clusters of points is considered as a retrieval; the total vote values of the grids in each cluster are considered for ranking the retrievals.

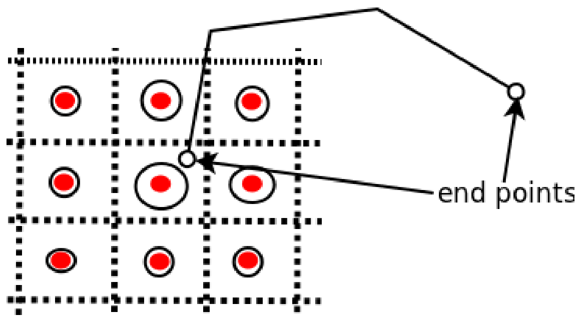


Figure 4: Illustration of voting: For each of the selected paths from the hash table, we accumulate the votes to the nine nearest grids of each of the 2 terminal vertices of that path.

Algorithm 3.2 Spotting of query symbols in documents

Require: A model symbol (S_m) with the set of path descriptors $f_{p_1^{S_m}}, \ldots, f_{p_t^{S_m}}$
  and a set $\mathcal{T}$ of hash tables.
Ensure: A ranked list $ROI = \{R_1, R_2, \ldots\}$ of regions of interest.
  //Search for the nearest buckets
  for all $f_{p_i^{S_m}}$ of $f_{p_1^{S_m}}, \ldots, f_{p_t^{S_m}}$ **do**
    $B_i \Leftarrow$ nearest bucket of $f_{p_i^{S_m}} \in \mathcal{T}$
    //Calculate the matching scores
    for all $f_{p_{B_j}}$ of B_i **do**
      $MS_{i,j} \Leftarrow$ matching score of $(f_{p_i^{S_m}}, f_{p_{B_j}})$
    end for
  end for
  //Define and initialize the voting space
  for all $D_k \in Doc$ **do**
    // Grids of three different sizes
    for all $gsize$ of $\{[10 \times 10], [20 \times 20], [30 \times 30]\}$ **do**
      $G_{gsize}^{D_k} \Leftarrow \oslash$ //Grids on documents
      $GV_{gsize}^{D_k} \Leftarrow \oslash$ //Vote values for the grids
    end for
  end for

//Voting
for all B_i of $B_1, \ldots, B_t$ **do**
  for all $f_{p_{B_j}}$ of B_i **do**
    $D \Leftarrow$ document of $f_{p_{B_j}}$
    $[pt_1, pt_2] \Leftarrow$ two end points of $f_{p_{B_j}}$
    for all $gsize$ of $\{[10 \times 10], [20 \times 20], [30 \times 30]\}$ **do**
      for all pt_i of $[pt_1, pt_2]$ **do**
        $G_D(1:9) \Leftarrow$ Nine neighbouring grids of pt_i
        $CG_{gsize}(1:9) \Leftarrow$ Centres of $G^D(1:9)$
        $GDist(1:9) \Leftarrow$ distance between ($CG_{gsize}(1:9)$,pt_i)
        $GV^D_{gsize}(G^D_{gsize}(1:9)) \Leftarrow GV^D_{gsize}(G^D_{gsize}(1:9)) + GDist(1:9) \times \frac{1}{MS_{i,j}}$
      end for
    end for
  end for
end for
//Spotting
$S \Leftarrow \oslash$
for all $D_k \in Doc$ **do**
  for all $gsize \in \{[10 \times 10], [20 \times 20], [30 \times 30]\}$ **do**
    $[Class^{D_k}_{gsize}(h), Class^{D_k}_{gsize}(l)]$=kmeans($GV^{D_k}_{gsize}, 2$)
    //mean($GV^{D_k}_{gsize}(Class^{D_k}_{gsize}(l))$)$\leq$mean($GV^{D_k}_{gsize}(Class^{D_k}_{gsize}(h))$),
    //where $GV^{D_k}_{gsize}(Class^{D_k}_{gsize}(h))$ are the higher voted grids
  end for
  $G^{D_k}_{all} \Leftarrow G^{D_k}_{[10\times10]}(Class^{D_k}_{[10\times10]}(h)) \cup G^{D_k}_{[20\times20]}(Class^{D_k}_{[20\times20]}(h)) \cup G^{D_k}_{[30\times30]}(Class^{D_k}_{[30\times30]}(h))$
  $\{(s_1, total\_votes(s_1)), (s_2, total\_votes(s_2)), \ldots\} \Leftarrow$ spatial clustering($G^{D_k}_{all}$)
  $S \Leftarrow S \cup \{(s_1, total\_votes(s_1)), (s_2, total\_votes(s_2)), \ldots\}$
end for
$ROI = sort(S, key = total\_votes)$

4. Experimental results

In this section we present the results of several experiments. The first experiment is designed to see the efficiency between the Zernike moments and the Hu moment invariants in a comparative way to represent the graph paths. The second experiment is undertaken to show the variation of symbol spotting results by varying the L and K parameters of the hash table creation. Then a set of experiments is performed to test efficiency of the proposed method to spot the symbols on documents. For that we use four different sets of images with varying difficulties. The last experiment is performed to see the possibility of applying the proposed method to any other information spotting methodologies; for that we test the method with handwritten word spotting in some real historical handwritten documents. Next we present a comparative study with a state-of-the-art method. For all these experiments we mainly use two publicly available databases of architectural floorplans: FPLAN-POLY[1] [31] and SESYD (floorplans)[2] [44]. The FPLAN-POLY dataset is a collection of 42 real floorplans (for example see Figure 7a) and 38 cropped symbols as the queries. The datasets are available in a vectorized form and the vectorization is performed by the Qgar[3] software. Conversely, the SESYD (floorplans) dataset contains 10 different sub-datasets, each of which contains 100 different synthetically-generated floorplans (Figure 7b). All the floorplans in a sub-datasets are created based upon the same floorplan template by putting different model symbols in different places in random orientations and scales. Depending upon the need of particular experiments, we introduce some noise models to test the robustness of the method.

4.1. Zernike moments versus Hu moment invariants

This test aims to compare the two description methods used to describe graph paths. Finally, based on this experiment, the best method is used in the remaining experiments. We compare the performance of the presented algorithm by using both description methods. To undertake this experiment, we consider the FPLAN-POLY database and perform the path description with Hu moment invariants and Zernike moments with different orders (6 to 10). In Figure 5 we show a precision recall curve showing the performance with different descriptions. This shows that the Zernike moments with any order outperforms the Hu moment invariants, on average there is a gain of 6.3% precision for a given recall value. Finally, in this experiment, Zernike moments with order 7 give the best trade-off in terms of performance. This gives the imperative to perform the rest of the experiments with Zernike moments descriptors with order 7.

---

[1] `http://www.cvc.uab.es/~marcal/FPLAN-POLY/index.html`
[2] `http://mathieu.delalandre.free.fr/projects/sesyd/floorplans.html`
[3] `http://www.qgar.org/`

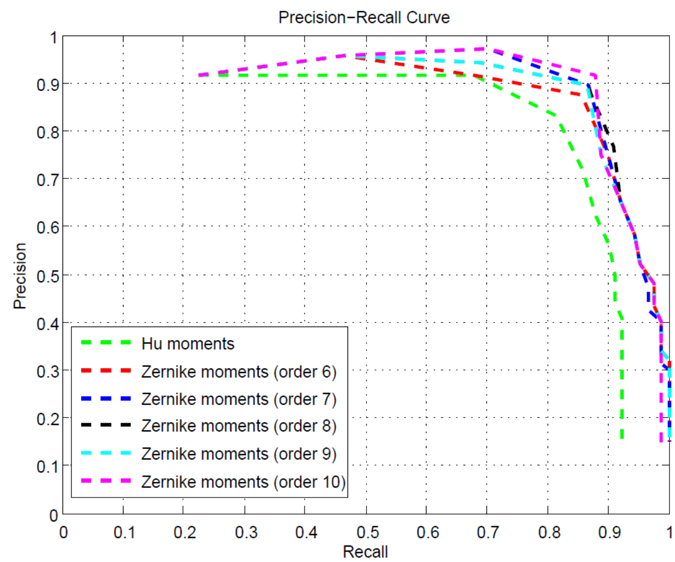
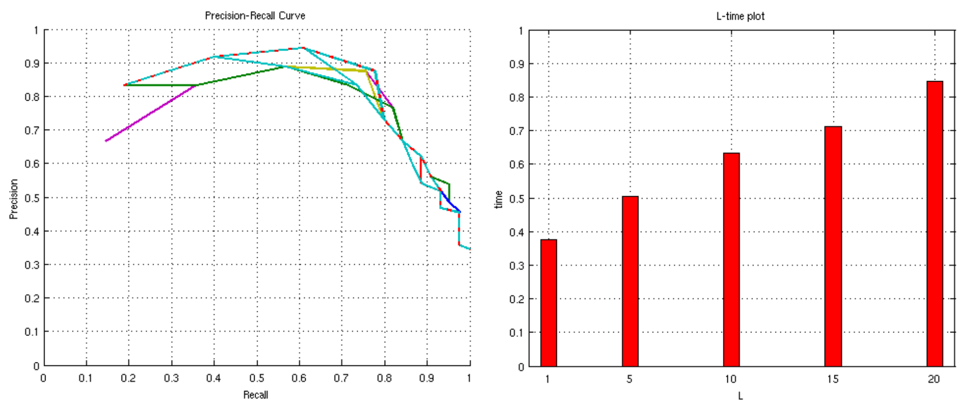


Figure 5: Precision-Recall plot showing the performance of the spotting method with the Hu moment invariants and Zernike moments of order 6 to 10.

4.2. Experiments on the influence of parameters L and K

Literally K is the maximum number of bits of the binary indices of different buckets in a table. So increasing K will increase the number of random combinations of the bit positions which ultimately increases the number of buckets in each of the hash tables. This creates tables in which many buckets with only a few instances appear, which separates the search space poorly. On the other hand, decreasing K will merge different instances incorrectly. The number of hash tables (L) is another parameter to play with, which indicates the number of tables to create for a database. Increasing L will increase the search space, since LSH considers the union of all the tables, so after a certain limit, increasing the number of tables will not improve the performance but only increase the retrieval time. So choosing the proper combination of L and K for a particular experiment is very important for efficient results.

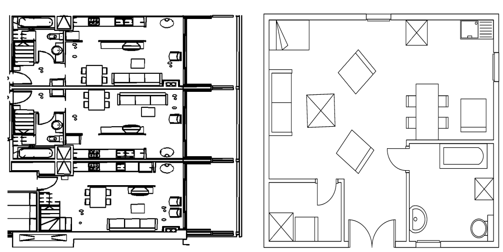
In this experiment we chose a set of 10 floorplans from the FPLAN-POLY dataset and created the hashing data structures by varying L from 1 to 20 and K from 40 to 80. The performance of the spotting method is shown in terms of the precision-recall curves in Figure 6a, which shows similar performance



(a) (b)

Figure 6: (a) The precision-recall plot of the spotting method by varying L 1 to 20 and K 40 to 80. (b) The plot of the time taken by the method to retrieve symbols for different values of L.

for all the settings. But the time taken by the spotting method increases proportionally with the increment of L (Figure 6b).

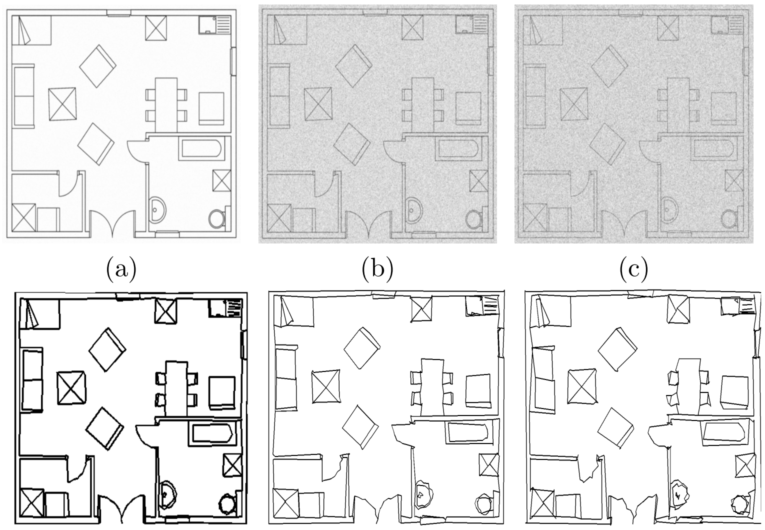


(a) (b)

Figure 7: Example images from (a) FPLAN-POLY dataset (b) SESYD dataset.

4.3. Symbol spotting experiments

In order to evaluate the proposed spotting methodology, we present four different experiments. The first experiment is designed to test the method on the images of real-world floorplans. The second experiment is performed to check the algorithm on a moderately large dataset which is a synthetically created benchmark. Then the experiments are performed to test the efficiency of the method on the images of handwritten sketch-like floorplans.



(a) (b) (c) (d) (e) (f)

Figure 8: Examples of degraded floorplans. (a)-(c) The same floorplan in Figure 7b degraded with Gaussian noise (m=0.1, σ=0.01), (m=0.3, σ=0.05) and (m=0.5, σ=0.09) respectively, (d)-(f) The same floorplan in Figure 7b degraded with vectorial noise r=5, 10 and 15 respectively.

Lastly we conducted some experiments to test the method on some noisy images, where the kind of noise is very similar to the noise introduced by scanning or any other low-level pre-processing.

The set of available query symbols for each dataset are used as queries to evaluate 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**). In general, the precision (**P**) and recall (**R**) are computed as:

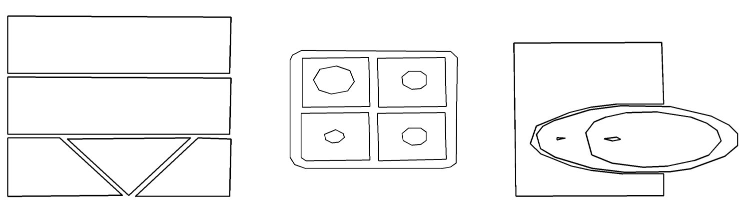
$$P = \frac{|ret \cap rel|}{|ret|}; R = \frac{|ret \cap rel|}{|rel|} \tag{5}$$

Here in eqn. 5, the precision and recall measures are computed on the whole set of retrievals returned by the system. That is, they give information about the final performance of the system after processing a query and do not take into account the quality of ranking in the resulting list. But IR systems return results ranked by a confidence value. The first retrieved items are the ones the system believes that are more likely to match the query. As the system provides more and more results, the probability to find non-relevant items increases. So in this paper the precision value is computed as the $P(r_{max})$ i.e. the precision attained at r_{max}, where r_{max} is the maximum recall attained by the system and average precision is computed as:

$$AveP = \frac{\sum_{n=1}^{n=|ret|} P(n) \times r(n)}{|rel|} \tag{6}$$

where $r(n)$ is an indicator function equal to one, if the item at rank n is a relevant instance or zero otherwise. The interested reader is referred to [45] for the definition of the previously mentioned metrics for the symbol spotting problem. To examine the computation time we calculate the per document retrieval time (**T**) for each of the symbols. For each of the datasets the mean of the above mentioned metrics are shown to judge the overall performance of the algorithm.

All the experiments described below are performed 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 are stored in 16,000 buckets, so compared to a one-to-one path comparison, the search space is reduced by a factor of 90.



(a) (b) (c)

Figure 9: Examples of model symbols from the FPLAN-POLY dataset used for our experiment.

4.3.1. Experiment on FPLAN-POLY with real-world images

We have tested our method with the FPLAN-POLY dataset. This experiment is undertaken to show the efficiency of the algorithm on real images,

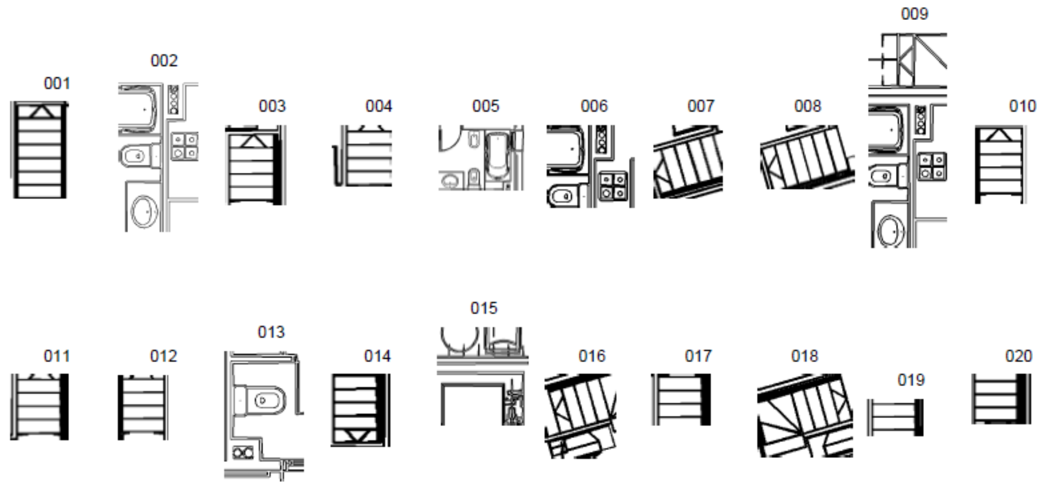


Figure 10: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol in Figure 9a in the FPLAN-POLY dataset.

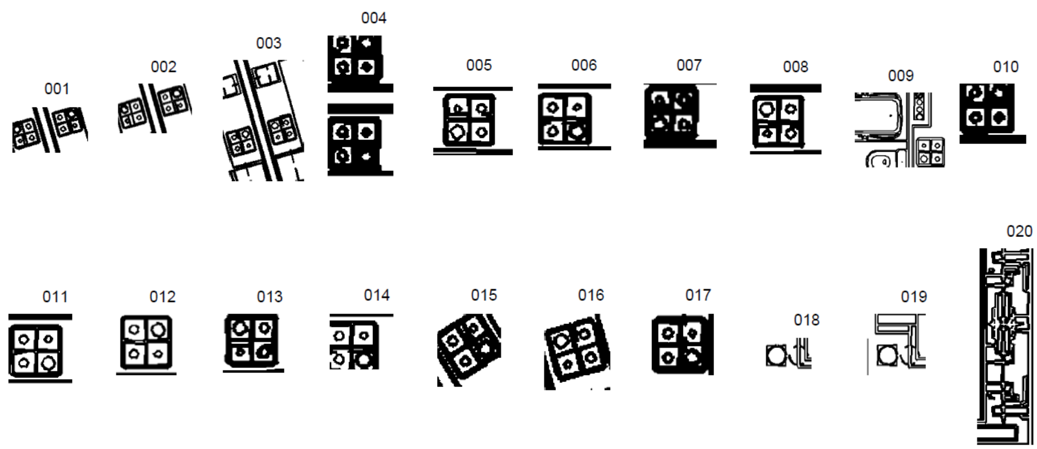


Figure 11: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol in Figure 9b in the FPLAN-POLY dataset.

which could suffer from the noise introduced in the scanning process, vectorization etc.

The recall rate achieved by the method is 93% which shows the efficiency of the algorithm in retrieving the true symbols. The average precision obtained by the method is 79.52% which ensures the occupancy of the true positives at the beginning of the ranked retrieval list. The precision value

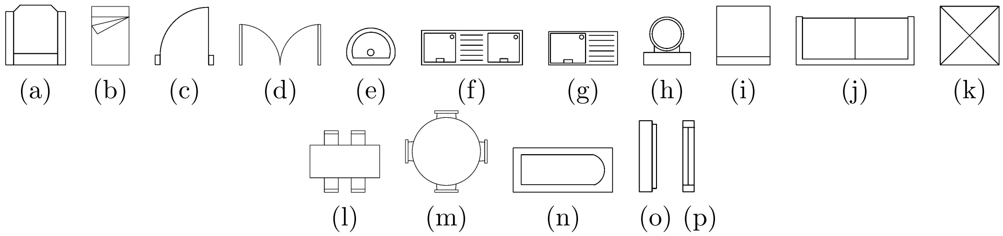


Figure 12: Model symbols in the SESYD dataset.

of the method is 77.87% which is more than 50% better than the precision reported by the latest state-of-the-art method [21] on this dataset. This signifies that the false positives are ranked worse than the correct results. This fact is also clear from Figure 10, 11, where we show the qualitative results obtained by the 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.

4.3.2. Scalability experiment on SESYD

We have also tested our method on the SESYD (floorplans) dataset. This experiment is designed to test the scalability of the algorithm i.e. to check the performance of the method on a dataset which is sufficiently large.

Table 3: Results with SESYD dataset

Database	**P**	**R**	**AveP**	**T**
floorplans16-01	41.33	82.66	52.46	0.07
floorplans16-02	45.27	82.00	56.17	0.09
floorplans16-03	48.75	85.52	71.19	0.07
floorplans16-04	54.51	74.92	65.89	0.05
floorplans16-05	53.25	91.67	67.79	0.08
floorplans16-06	52.70	78.91	60.67	0.07
floorplans16-07	52.78	83.95	65.34	0.07
floorplans16-08	49.74	90.19	58.15	0.08
floorplans16-09	51.92	77.77	47.68	0.07
floorplans16-10	50.96	83.01	63.39	0.08
mean	**50.32**	**83.06**	**60.87**	**0.07**

The mean measurements for each of the sub-datasets are shown in Table

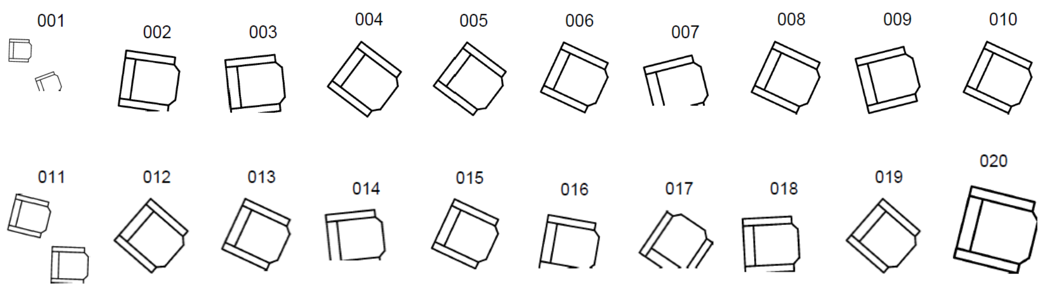


Figure 13: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol shown in Figure 12a in the SESYD (floorplans16-01) dataset.

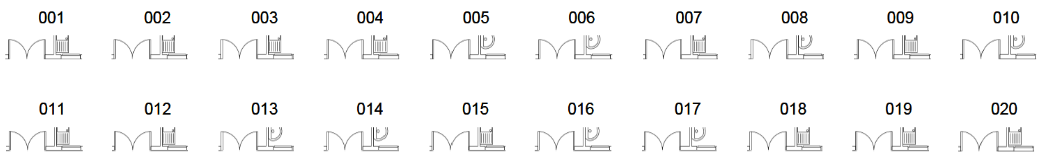


Figure 14: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol shown in Figure 12d in the SESYD (floorplans16-05) dataset.

3. The recall values for all the sub-datasets are quite good, although the average precisions are less than in the previous experiments. This is due to the existence of the similar substructures (graph paths) among different symbols (for example, between the symbols in Figure 12c and Figure 12d between the symbols in Figure 12f and Figure 12g, among the symbols in Figures 12a, 12b, 12i and 12k and etc). These similarities negatively affect the vote values considered for ranking the retrievals. There is an interesting observation regarding the average time taken for the retrieval procedure, which is 0.07 sec. to retrieve a symbol per document image, which is much less than the previous experiment. This is due to the hashing technique, which allows for the collision of the same structural elements and inserts 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, in Figures 13, 14, 15 and 16, we present some qualitative results on the SESYD dataset.

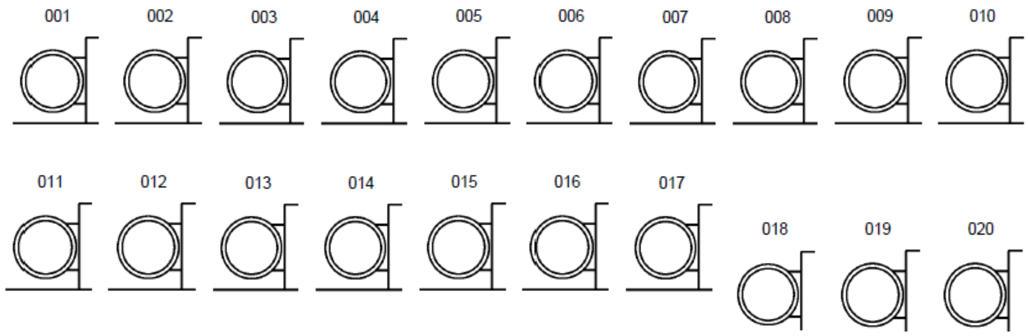


Figure 15: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol shown in Figure 12h in the SESYD (floorplans16-05) dataset.

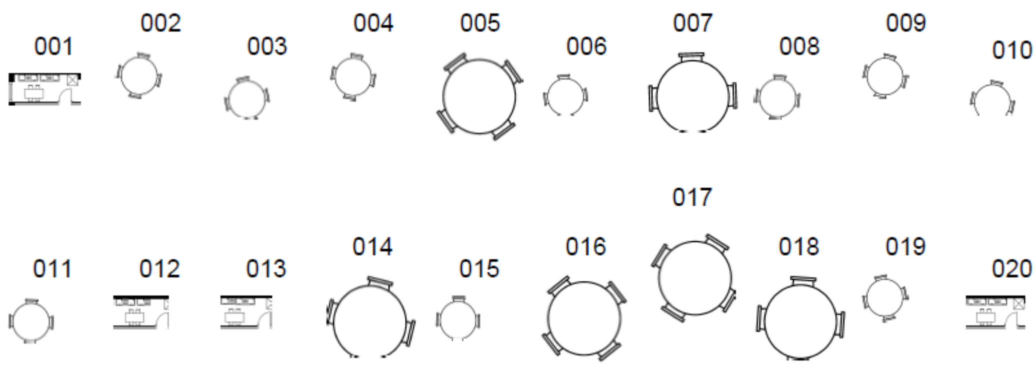


Figure 16: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol shown in Figure 12m in the SESYD (floorplans16-01) dataset.

4.3.3. Experiment on SESYD-VN to test vectorial distortion

This experiment is undertaken to test the effectiveness of the algorithm on the handwritten sketch-like floorplans. For this we select one of the 16 sub-datasets of the SESYD foorplans and introduces vectorial noise with different levels (see Figures 8d, 8e, 8f). 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 difficulty (for r = 5, 10, 15). For all the different distortions the same model symbols are used as queries.

Table 4: 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

The measurements of the method are shown in Table 4. 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 when compared to 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 a large number of buckets and hence ultimately increases the per document retrieval time.

Table 5: Results with SESYD-GN dataset

mean (m)	**variance** (σ)	**P**	**R**	**AveP**	**T**
0.1	0.01	24.36	94.86	74.07	0.25
	0.05	21.79	89.46	60.07	0.35
	0.09	15.38	67.77	42.85	1.47
0.2	0.01	24.36	94.87	73.43	0.26
	0.05	20.00	82.19	48.93	1.16
	0.09	15.38	65.44	30.97	1.58
0.3	0.01	24.10	93.34	65.79	2.12
	0.05	14.62	69.11	40.81	2.30
	0.09	12.05	54.12	25.62	3.15
0.4	0.01	15.89	72.45	36.32	1.95
	0.05	11.79	50.64	17.97	2.11
	0.09	11.54	43.78	15.29	2.49
0.5	0.01	9.74	34.56	10.00	0.52
	0.05	8.20	29.94	6.69	0.74
	0.09	9.23	36.07	11.14	0.84

4.3.4. *Experiment on SESYD-GN with noisy images*

The last symbol spotting experiment is performed to test the efficiency of the algorithm on noisy images, which might be generated in the scanning process. For this, we also selected one of the 16 sub-datasets of SESYD floorplans and introduced Gaussian noise at different levels (see Figure 8a, 8b, 8c) with the mean (m) of 0.1 to 0.5 with step 0.1 and with variance (σ) 0.01 to 0.09 with step 0.04, which generates a total 15 sets of images with different levels of noise. Practically, the increment of variance introduced more pepper noise into the images, whereas the increment of the mean introduced more and more white noise, which will detach the object pixel connection. Here we do not apply any kind of noise removal technique other than pruning, which eliminates isolated sets of pixels.

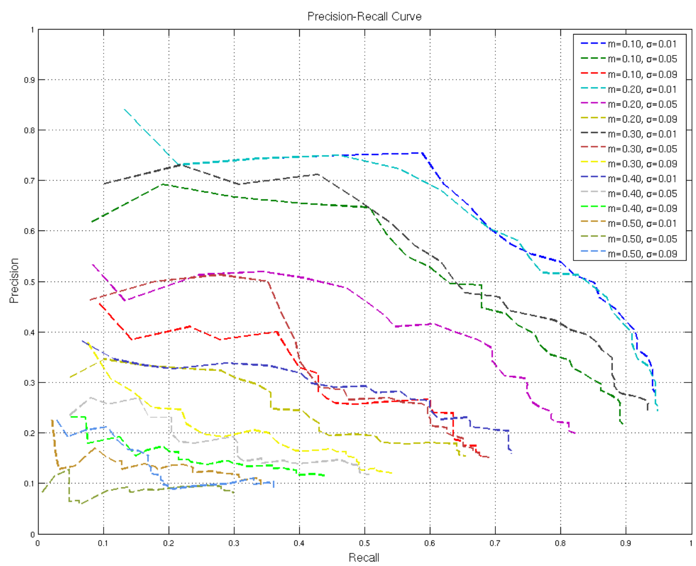
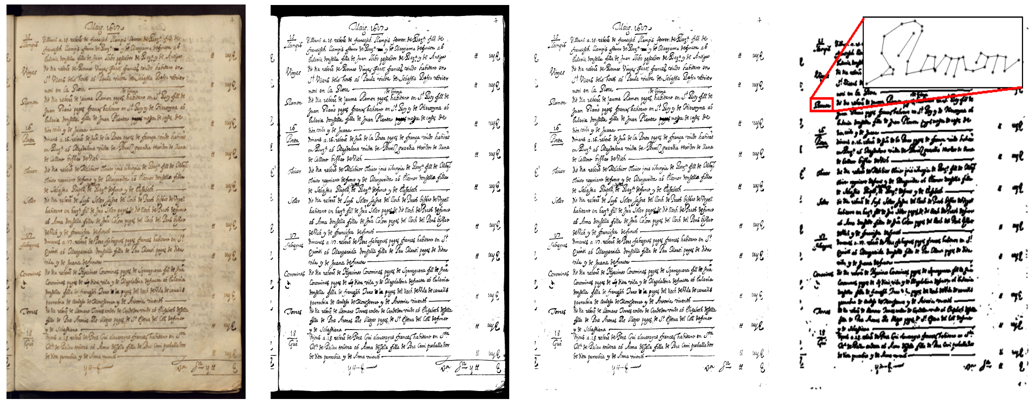


Figure 17: Precision-Recall plot generated by the spotting experiments with different levels of Gaussian noise.

The mean measures of metrics are shown in Table 5 and the performance of the method is shown in Figure 17 in terms of the precision recall curves. Clearly, from the precision-recall curves, the impact of variance is more than that of the mean. This implies that with the introduction of more and more random black pixels, there is a decrease in the performance, which is due to the distortion in the object pixels that substantially affects the vectorization methods and changes the local structural features of the graph paths. On



(a) (b) (c) (d)

Figure 18: An image from the marriage register from the fifth century from the Barcelona cathedral, (a) The original image, (b) The binarized image of 18a, (c) The image in 18b after preprocessing (eliminating black border created due to scanning), (d) Graph constructed from the image in 18c: the inset also shows the zoomed part of a word 'Ramon'.

the other hand, the increment of the mean introduces white pixel noise which ultimately separates an object into different parts and which facilitates the loss of the local structural information. Increase in Gaussian noise introduces local distortions (both with black and white pixels) which introduces extra points, as well as discontinuity during the vectorization process. These random points increase the time for computing the paths and also the number of buckets due to the random structure of them. Since the increment of the mean after a certain stage breaks a component into several pieces, the vectorization results in simple structures of isolated components. These structures are quite similar, since in most of the cases they are the straight lines or simple combination of straight lines which further decrease the retrieval time as they reduce the number of buckets. This explains the increase of retrieval time up to a certain stage and then again the decrease. The increment of both mean and standard deviation of the Gaussian noise creates a lot of discontinuity within the structure of objects; this creates lot of spurious parts after vectorization. These parts are not distinctive among different symbolic objects, which explains the irregular shape of the precision recall curves with the increase of noise.

4.4. Experiment on handwritten word spotting

This experiment is performed to demonstrate the possibility of applying our method to any other kind of information spotting system. For that we have chosen a handwritten word spotting application which also has received some popularity amongst the research community. The experiment is performed on a set of 10 unsegmented handwritten images taken from a collection of historical manuscripts from the marriage register of the Barcelona cathedral (see Figure 18). Each page of the manuscripts contains approximately 300 words. The original larger dataset is intended for retrieval, indexing and to store in a digital archive for future access. We use skeletonization based vectorization to obtain the vectorized documents. Before skeletonization, the images undergo preprocessing such as binarization by Otsu's method [46] and removal of the black borders generated in the scanning process. Then we construct the graph from the vectorial information and proceed by considering this as a symbol spotting problem. The retrieval results of the method on the handwritten images are promising, which is also clear from the qualitative results shown in Figure 19. This shows a very good retrieval of the word "de" with almost perfect segmentation. We also observe some limitations of the method in spotting handwritten words, among them, when a particular query word is split into several characters or components, the method is more prone to retrieve the character, which is more discriminative with respect to the other characters in the word. This is due to the non-connectivity of the word symbol, which reduces the overall structural information. Another important observation is that the computation of paths takes a substantial amount of time for the handwritten documents, since handwritten characters contain many curves. This generate more and more spurious critical points in the images, which ultimately affects the path computation time.

4.5. Discussions

In order to compare the performance of the proposed method with other methods, we compare our results with three state-of-the-art methods respectively proposed by Luqman et al. [22], Rusiñol et al. [31] and Qureshi et al. [19]. The method put forward by Luqman et al is based on graph embedding, the method due to Rusiñol et al. is based on the relational indexing of primitive regions contained in the symbol and that proposed by Qureshi et al. is based on graph matching, where the methods due to Luqman et al. and Qureshi et al. [19, 22] use a pre-segmentation technique to find the regions of

Figure 19: The first 120 retrievals of the handwritten word 'de' in the Marriage documents of the Barcelona Cathedral.

interest, which probably contain the graphic symbols. Generally this kind of localization method works to find some region containing loops and circular structures etc. Then a graph matching technique is applied either directly in the graph domain or in the embedded space to each of the regions in order to match the queried symbol. The method proposed by Rusiñol et al. [31] works without any pre-segmentation. For experimentation, we considered the images from a sub-dataset of SESYD, The sub-dataset contains 200 images of floorplans. The mean measurements at the recall value of 90.00% are shown in Table 6 and the performance of the algorithm is shown in terms of the precision-recall plot in Figure 20. Clearly, the proposed method dominates over the existing methods. For any given recall, the precision given by our method is approximately 12% more than that reported by Qureshi et al. [19], 10% more than that indicated by Rusiñol et al. [31] and 6% more than that resulted by Luqman et al. [22], which is a substantial improvement.

Finally, we use our algorithm as a distance measuring function between a pair of isolated architectural symbols, let us say, S_1 and S_2. In this case we do not perform any hashing, instead we simply factorize the symbols into

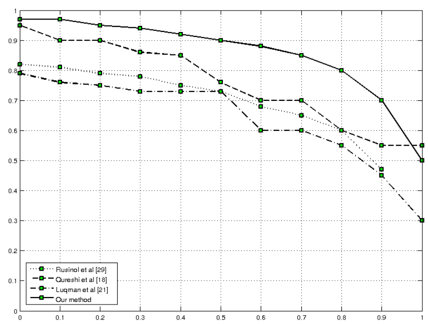


Figure 20: Precision-Recall plot generated by the spotting methods proposed by Luqman et al. [22], Qureshi et al. [19], Rusiñol et al. [31] and our proposed method.

Table 6: Comparison with the state-of-the-art methods

Methods	**P**	**R**	**AveP**	**T**
Qureshi et al. [19]	45.10	90.00	64.45	1.21
Rusiñol et al. [31]	47.89	90.00	64.51	-
Luqman et al. [22]	56.00	90.00	75.70	-
Our method	70.00	90.00	86.45	0.07

graph paths and describe them with some shape descriptors as explained in subsection 3.2. Then we use these descriptors to match a path of, say, symbol S_1, to the most identical path of S_2. So the total distance between the symbols S_1 and S_2 is the sum of such distances:

$$\sum_{p_i \in S_1} \min_{p_j \in S_2} dist(p_i, p_j)$$

We use this total distance to select the nearest neighbours of the query symbol. It is expected that for a pair of identical symbols, the algorithm will give a lower distance than for a non-identical symbol. This experiment is undertaken to compare our method with various symbol recognition methods available in the literature. When using the GREC2005 [47] dataset for our experiments, we only considered the set with 150 model symbols. The results are summarized in Table 7. We have achieved a 100% recognition rate for clear symbols (rotated and scaled) which shows that our method can efficiently handle the variation in scale and rotation. Our method outperforms the GREC participants (results obtained from [47]) for degradation models 1, 2, 3 and 5. The recognition rate decreases drastically for models 4 and 6, this is because the models of degradation lose connectivity among the foreground pixels. So after the vectorization, the constructed graph can not represent the complete symbol, which explains the poorer results.

Table 7: Results of symbol recognition experiments

Database	**Recognition rate**
Clear symbols (rotated & scaled)	100.00
Rotated & degraded (model-1)	96.73
Rotated & degraded (model-2)	98.67
Rotated & degraded (model-3)	97.54
Rotated & degraded (model-4)	31.76
Rotated & degraded (model-5)	95.00
Rotated & degraded (model-6)	28.00

In general the symbol spotting results of the system on the SESYD database are worse than the FPLAN-POLY (see Table 8). This is due to the existence of more similar symbols in the collection, which often create confusion amongst the query samples. But the average time for retrieving the symbols per document is much lower than the FPLAN-POLY database. This is because of the hashing technique that allows collision of the same structural elements and inserts 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, which ultimately reduces the per document retrieval time.

Table 8: Comparative results on two databases FPLAN-POLY & SESYD

Database	**P**	**R**	**AveP**	**T**
FPLAN-POLY	77.87	93.43	79.52	0.18
SESYD	50.32	83.06	60.87	0.07

Our system also produces some erroneous results (see Figures 10(002, 005, 006, 013, 015) and Figures 21(001, 002, 003, 004, 014, 019)) due to the appearance of similar substructures in nearby locations. For example the

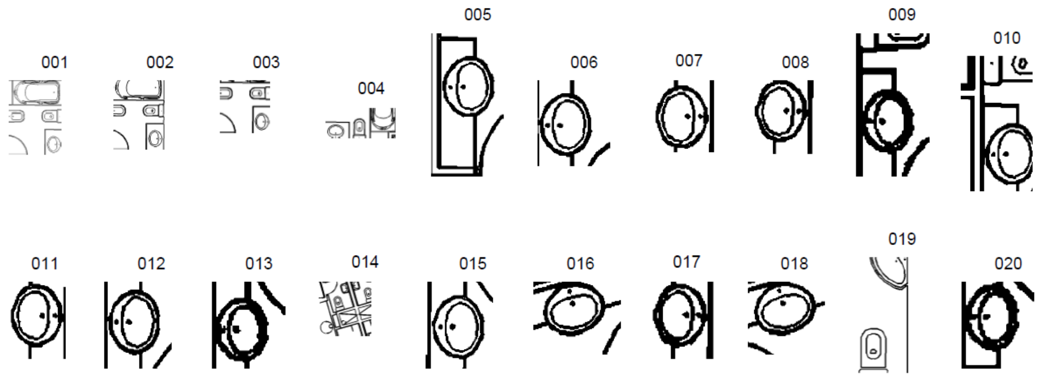


Figure 21: Qualitative results of the method: first 20 retrieved regions obtained by querying the symbol 9c in the FPLAN-POLY dataset.

symbol in Figures 9a contains some rectangular box like subparts. The paths from these derived substructures of the symbol resemble some commonly occurring substructures (walls, mounting boxes etc.) in a floorplan. This creates a lot of false votes, which explains the retrieval of the false instances in Figure 10. Similarly, the subparts of the symbol in Figure 9c resemble the subparts of some architectural symbols. This explains the occurrence of the false retrievals in Figure 21.

5. Conclusions

In this paper we have proposed a graph based approach for symbol spotting in graphical documents. We represent the documents with graphs where the critical points detected in the vectorized graphical documents are considered as the nodes and the lines joining them are considered as the edges. The document database is represented by the unification of the factorized substructures of graphs. Here the graph substructures are the acyclic graph paths between each pair of connected nodes. The factorized substructures are the one-dimensional (sub)graphs which give efficiency in terms of computation and since they provide a unified representation over the database, the computation is substantially reduced. Moreover, the paths give adaptation to some structural errors in documents with a certain degree of tolerance. We organize the graph paths in hash tables using the LSH technique, this helps to retrieve symbols in real-time. We have tested the method on different datasets of various kinds of document images.

The main contribution of the method is to deal with a large graph dataset, whereby dealing with graphs demands more computational complexity. This has become possible for the factorization technique of graphs which creates an efficient indexation structure with LSH on top of the database. LSH makes the organization efficient and the retrieval faster due to the binary representations of the descriptors. The method has performed quite well in real-world images, this is also due to the factorization of graphs, which allows structural noise to a certain level and is very useful for real images. This fact is also proved when the method performed well with vectorial noise, in this case of course the performance decreases with the increase of the noise level. The method performs worse with the increase of Gaussian noise. This kind of noise introduces lot of spurious points and also disconnections throughout the vectorization process, which affects the structural attributes of graph paths and reduces the performance. Also for our experiments we have created some distorted floorplan datasets represented with graph (SESYD-GN, SESYD-VN) and we believe the research community will be benefited of the graph datasets used in the experiments of this paper.

The proposed method works with the vectorized information of the document and the graph representations are created from vectorized documents. This implies that the method is highly dependent on the vectorization procedure. If the vectorization is not robust to noise, even after having some tolerance to it, the method performs poorly with it, which is clear in the experiment with the pixel (Gaussian) noise. Moreover, when a new document is included in the database, the system needs to repeat the creation of the hash table i.e. a part of the offline procedure, which could be considered as an overhead of the whole system.

It is true that the consideration of the graph paths between each pair of connected nodes creates redundant information but we have argued that path redundancy is needed to deal the structural noise in the documents. To reduce the number of redundant paths, we can further think of mutually exclusive factorization of the graph paths. But this is not straight forward, moreover, in that case we should take care on the stability of the path structure. To do that, we can factorize the graphs hierarchically depending on the curvature of the graph nodes. So, these need further investigations and experiments which will be our future research issue.

6. Acknowledgement

The authors want to thank to the anonymous reviewers for their comments which were really helpful for the betterment of the article. The authors are grateful to Dr. Marçal Rusiñol for running his method and providing the results for the method comparison. This work has been partially supported by the Spanish projects TIN2009-14633-C03-03, TIN2008-04998, CSD2007-00018 and the PhD scholarships 2011FI_B 01022, 2012FI_B1 00174 provided by the Catalan Government research agency AGAUR.

References

[1] T. M. Rath, R. Manmatha, Word image matching using dynamic time warping, in: IEEE International Conference on Computer Vision and Pattern Recognition, Vol. 2, IEEE Computer Society, Los Alamitos, CA, USA, 2003, pp. 521–527.

[2] S. Lu, L. Li, C. L. Tan, Document image retrieval through word shape coding, IEEE Transactions on Pattern Analysis and Machine Intelligence 30 (2008) 1913–1918.

[3] J. Lladós, E. Valveny, G. Sánchez, E. Martí, Symbol recognition: Current advances and perspectives, in: D. Blostein, Y.-B. Kwon (Eds.), Graphics Recognition Algorithms and Applications, Vol. 2390 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2002, pp. 104–128.

[4] K. Tombre, B. Lamiroy, Pattern recognition methods for querying and browsing technical documentation, in: 13th Iberoamerican Congress on Pattern Recognition, CIARP 2008, LNCS, vol. 5197, Springer-Verlag, 2008.

[5] S. R. Joty, S. Sadid-Al-Hasan, Advances in focused retrieval: A general review, in: 10th IEEE International Conference on Computer and Information Technology (ICCIT 2007), 2007, pp. 1–5.

[6] D. Conte, P. Foggia, C. Sansone, M. Vento, Thirty years of graph matching in pattern recognition, International Journal of Pattern Recognition and Artificial Intelligence 18 (3) (2004) 265–298.

[7] P. L. Rosin, G. A. West, Segmentation of edges into lines and arcs, Image and Vision Computing 7 (2) (1989) 109 – 114.

[8] K. Mehlhorn, Graph algorithms and NP-completeness, Springer-Verlag New York, Inc., New York, NY, USA, 1984.

[9] P. Indyk, R. Motwani, Approximate nearest neighbors: towards removing the curse of dimensionality, in: Proceedings of the thirtieth annual ACM symposium on Theory of computing, STOC '98, ACM, New York, NY, USA, 1998, pp. 604–613.

[10] A. Gionis, P. Indyk, R. Motwani, Similarity search in high dimensions via hashing, in: Proceedings of the 25th International Conference on Very Large Data Bases, VLDB '99, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999, pp. 518–529.

[11] M. Rusiñol, Geometric and structural-based symbol spotting. application to focused retrieval in graphic document collections, Ph.D. thesis (2009).

[12] A. El-Yacoubi, M. Gilloux, R. Sabourin, C. Suen, An hmm-based approach for off-line unconstrained handwritten word modeling and recognition, Pattern Analysis and Machine Intelligence, IEEE Transactions on 21 (8) (1999) 752–760.

[13] S. España-Boquera, M. Castro-Bleda, J. Gorbe-Moya, F. Zamora-Martinez, Improving offline handwritten text recognition with hybrid hmm/ann models, IEEE Transactions on Pattern Analysis and Machine Intelligence 33 (4) (2011) 767–779.

[14] Y. He, A. Kundu, 2-d shape classification using hidden markov model, Pattern Analysis and Machine Intelligence, IEEE Transactions on 13 (11) (1991) 1172–1184.

[15] S. Müller, G. Rigoll, Engineering drawing database retrieval using statistical pattern spotting techniques, in: Graphics Recognition Recent Advances, Vol. 1941 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2000, pp. 246–255.

[16] B. Messmer, H. Bunke, Automatic learning and recognition of graphical symbols in engineering drawings, in: R. Kasturi, K. Tombre (Eds.),

Graphics Recognition Methods and Applications, Vol. 1072 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1996, pp. 123–134.

[17] J. Lladós, E. Martí, J. J. Villanueva, Symbol recognition by error-tolerant subgraph matching between region adjacency graphs, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (2001) 1137–1143.

[18] E. Barbu, P. Heroux, S. Adam, E. Trupin, Frequent graph discovery: Application to line drawing document images, Electronic Letters on Computer Vision and Image Analysis 5 (2) (2005) 47–54.

[19] R. Qureshi, J.-Y. Ramel, D. Barret, H. Cardot, Spotting symbols in line drawing images using graph representations, in: W. Liu, J. Lladós, J.-M. Ogier (Eds.), Graphics Recognition. Recent Advances and New Opportunities, Vol. 5046 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2008, pp. 91–103.

[20] H. Locteau, S. Adam, Éric Trupin, J. Labiche, P. Héroux, Symbol spotting using full visibility graph representation, in: Proceedings of 7th International Workshop of Graphics Recognition, 2007.

[21] M. Rusiñol, J. Lladós, G. Sánchez, Symbol spotting in vectorized technical drawings through a lookup table of region strings, Pattern Analysis and Applications 13 (2009) 1–11.

[22] M. Luqman, T. Brouard, J.-Y. Ramel, J. Lladós, A content spotting system for line drawing graphic document images, in: Pattern Recognition (ICPR), 2010 20th International Conference on, 2010, pp. 3420–3423.

[23] N. Nayef, T. M. Breuel, A branch and bound algorithm for graphical symbol recognition in document images, in: Proceedings of Ninth IAPR International Workshop on Document Analysis System (DAS,'2010), 2010, pp. 543–546.

[24] P. L. Bodic, P. Hèroux, S. Adam, Y. Lecourtier, An integer linear program for substitution-tolerant subgraph isomorphism and its use for symbol spotting in technical drawings, Pattern Recognition 45 (12) (2012) 4214 – 4224.

[25] S. Tabbone, L. Wendling, K. Tombre, Matching of graphical symbols in line-drawing images using angular signature information, International Journal on Document Analysis and Recognition 6 (2) (2003) 115–125.

[26] T.-O. Nguyen, S. Tabbone, A. Boucher, A symbol spotting approach based on the vector model and a visual vocabulary, in: Document Analysis and Recognition, 2009. ICDAR '09. 10th International Conference on, 2009, pp. 708–712.

[27] P. Dosch, J. Lladós, Vectorial Signatures for Symbol Discrimination, Springer Berlin / Heidelberg, 2004, Ch. Vectorial Signatures for Symbol Discrimination, pp. 154–165.

[28] W. Zhang, L. Wenyin, A new vectorial signature for quick symbol indexing, filtering and recognition, in: Proceedings of the Ninth International Conference of Document Analysis and Recognition, Vol. 1, 2007, pp. 536 –540.

[29] D. Zuwala, S. Tabbone, A Method for Symbol Spotting in Graphical Documents, Springer Berlin / Heidelberg, 2006, pp. 518–528.

[30] M. Rusiñol, J. Lladós, Symbol Spotting in Technical Drawings Using Vectorial Signatures, Springer Berlin / Heidelberg, 2006, Ch. Symbol Spotting in Technical Drawings Using Vectorial Signatures, pp. 35–46.

[31] M. Rusiñol, A. Borràs, J. Lladós, Relational indexing of vectorial primitives for symbol spotting in line-drawing images, Pattern Recognition Letters 31 (3) (2010) 188–201.

[32] J.-M. Jolion, Some experiments on clustering a set of strings, in: E. Hancock, M. Vento (Eds.), Graph Based Representations in Pattern Recognition, Vol. 2726 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2003, pp. 214–224.

[33] C. Solnon, J.-M. Jolion, Generalized vs set median strings for histogram-based distances: Algorithms and classification results in the image domain, in: F. Escolano, M. Vento (Eds.), Graph-Based Representations in Pattern Recognition, Vol. 4538 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2007, pp. 404–414.

[34] J. Ros, C. Laurent, J.-M. Jolion, A bag of strings representation for image categorization, Journal of Mathematical Imaging and Vision 35 (2009) 51–67.

[35] N. Shervashidze, S. V. N. Vishwanathan, T. H. Petri, K. Mehlhorn, K. M. Borgwardt, Efficient graphlet kernels for large graph comparison, ReCALL 5 (2008) 488–495.

[36] F.-X. Dupé, L. Brun, Edition within a Graph Kernel Framework for Shape Recognition, Springer-Verlag, Berlin, Heidelberg, 2009, pp. 11–20.

[37] F.-X. Dupé, L. Brun, Hierarchical Bag of Paths for Kernel Based Shape Classification, Vol. 5342 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2010, pp. 227–236.

[38] M. R. Teague, Image analysis via the general theory of moments, J. Opt. Soc. Am. 70 (8) (1980) 920–930.

[39] K. Hosny, Fast computation of accurate zernike moments, Journal of Real-Time Image Processing 3 (2008) 97–107.

[40] G. Lambert, H. Gao, Line moments and invariants for real time processing of vectorized contour data, in: Image Analysis and Processing, Vol. 974 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 1995, pp. 347–352.

[41] G. Lambert, J. Noll, Discrimination properties of invariants using the line moments of vectorized contours, in: Pattern Recognition, 1996., Proceedings of the 13th International Conference on, Vol. 2, 1996, pp. 735–739 vol.2.

[42] M.-K. Hu, Visual pattern recognition by moment invariants, Information Theory, IRE Transactions on 8 (2) (1962) 179–187.

[43] A. Dutta, J. Lladós, U. Pal, A bag-of-paths based serialized subgraph matching for symbol spotting in line drawings, in: Proceedings of 5th Iberian Conference on Pattern Recognition and Image Analysis, (IbPRIA2011), Gran Canaria, 2011, pp. 620–627.

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

[45] M. Rusiñol, J. Lladós, A performance evaluation protocol for symbol spotting systems in terms of recognition and location indices, International Journal on Document Analysis and Recognition 12 (2) (2009) 83–96.

[46] N. Otsu, A threshold selection method from gray-level histograms, Systems, Man and Cybernetics, IEEE Transactions on 9 (1) (1979) 62–66.

[47] P. Dosch, E. Valveny, Report on the second symbol recognition contest, in: W. Liu, J. Llads (Eds.), Graphics Recognition. Ten Years Review and Future Perspectives, Vol. 3926 of Lecture Notes in Computer Science, Springer Berlin / Heidelberg, 2006, pp. 381–397.