

Engineering Drawings Recognition Using a Case-based Approach

Luo Yan

Department of Computer Science
City University of Hong Kong
luoyan@cs.cityu.edu.hk

Liu Wenyin

Department of Computer Science
City University of Hong Kong
csluwy@cityu.edu.hk

Abstract

In this paper, we propose a framework for engineering drawings recognition using a case-based approach. The key idea of our scheme is that, interactively, the user provides an example of one type of graphic object in an engineering drawing, then the system learns the graphical knowledge of this type of graphic object from the example and uses this learned knowledge to recognize or search for similar graphic objects in engineering drawings. The scheme emphasizes the following three distinct characteristics: automatism, run-time-ness, and robustness. We summarized five types of geometric constraints to represent the generic graphical knowledge. We also developed two algorithms for case-based graphical knowledge acquisition and knowledge-based graphics recognition, respectively. Experiments have shown that our proposed framework is both efficient and effective for recognizing various types of graphic objects in engineering drawings.

1. Introduction

The problem of automated recognizing engineering drawings can be divided into the following three levels:

1. Recognition of low-level graphic primitives
2. Recognition of high-level graphic objects
3. Higher-level intelligent interpretation and analysis

For the first level, the usual solution is transforming a raster image to a vector form and obtaining the low-level graphic primitives, such as lines, circles, arcs, etc. The second level performs the recognition of high-level graphic objects, which are composed of low-level primitives. At the third level, much more intelligent interpretation and analysis can be carried out, such as indexing, browsing, content-based retrieval, etc.

Currently, the second level is still a challenge since there are so many types of graphic objects in so many domains, such as logic circuit diagrams, mechanical drawings, architectural drawings, etc. Although many approaches to recognition of particular graphic objects have been reported in the literature, they have many inconveniences and disadvantages.

Most current graphics recognition approaches require each graphic pattern be known, analyzed, and represented

prior to the recognition process. It is especially true that a large number of samples are required for pre-training prior to recognition in statistical approaches (e.g., Template Matching approach [1] and Neural Network approach [2]). Similarly, the syntaxes and structures of the patterns are required to be pre-defined prior to recognition in syntactic and structural approaches [3]. Thus, most current approaches require much human involvement such as producing templates, training models, defining features, etc., prior to recognition. These tasks presently not only require professional knowledge but also are quite time-consuming and labor-intensive.

Besides the above disadvantages, the recognition ability of current approaches is also very limited. For statistical approaches, when the ground truth's dimension or orientation changes they usually will be unable to recognize them. For syntactic and structural approaches, features, syntaxes, and other knowledge about the graphic pattern are usually hard-coded in the algorithms. Hence, they can only deal with a limited set of specific and known graphic objects in a particular domain. In order to recognize new objects, the same analysis-design process needs to be repeated.

It is fine to hard-code for those very common low-level graphic primitives (e.g., lines, circles, and arcs) in the recognition algorithms. However, there are many different types of high-level graphic objects in many different domains. Even within a single domain, e.g., circuit diagrams, the number of graphic objects in common usage can be very large. Hence, it is non-realistic to hard-code all of them in the recognition algorithms. A generic method that can automatically learn new graphic objects for run-time recognition is strongly desired.

In this paper, we propose a new, case-based scheme for graphics recognition, which includes:

1. Graphical knowledge representation
2. Case-based knowledge acquisition
3. Knowledge-based graphics recognition

In the rest of this paper, we will first discuss the detail of our recognition scheme, followed by our proposed method for graphical knowledge representation and the two algorithms for case-based knowledge acquisition and knowledge-based graphics recognition, respectively. Finally, the experiment results and concluding remarks will be presented.

2. Our Scheme of Graphics Recognition

We carefully analyze the experience of how human beings recognize a particular graphic object and propose our scheme of graphics recognition as follows. We first decompose the whole graphic object into a few primitive components and analyze each component's attributes and spatial relationships (e.g., constraints) among them. Then, we store all these information as the knowledge of this type of graphic object. When recognizing a graphic object, we test if it conforms to the knowledge of a specific graphic type in the following steps. First, we select a component from the object and assume that it is one component of this graphic type according to its attributes. Next, we use this component and the constraints to figure out the nearby components and test whether they do exist at the right positions in the object. The similar reasoning procedure will be continued if the expected components are found. Otherwise, we select another component to restart the reasoning procedure. If all expected components are found at the expected positions, the recognition is successful and the object is recognized as of this type.

From the above analysis, precise and proper knowledge representation of graphic objects is the key point in our scheme. The spatial relationship is a very important feature of a graphic object, which contains some invariant information, e.g., topological invariants. When the dimension and/or the orientation of the graphic object change, this feature does not change and is therefore used in our scheme to represent graphical knowledge. Many approaches have been developed to represent this feature, such as Attributed Relation Graph (ARG, [4]), 2-D Strings [5], and Region Adjacent Graph (RAG, [6]). However, all of them are not suitable for our reasoning procedure. Thus, in this paper we propose a new method to represent the graphical knowledge, which will be discussed in the next section.

3. Graphical Knowledge Representation

First of all, we define the primitive components of a graphic object. We summarized three types of primitive components: line, circle, and arc. Each of them has 2 or 4 attributes as shown in the following table.

Line	P ₁	P ₂		
Circle	Center	Radius		
Arc	Center	Radius	Start-angle	End-angle

Since our scheme is actually a reasoning-based recognition procedure, we focus on the relative spatial relationships among the primitive components. That is, given one component we can use the graphical knowledge to figure out all other components in the object one by one. Hence, we only store the relative values to represent

the graphical knowledge. We use geometric constraints in our graphical knowledge representation. Geometric constraint is not a new concept, which has been widely used in CAD systems (e.g., [7]). However, in CAD systems (e.g., [7], [8] and [9]), the geometric constraints are defined, extracted, and specified by professional and experienced users. In our scheme, we want to make a computer extract the useful geometric constraints completely and automatically, and use these geometric constraints to recognize new graphic objects. Thus, our definition of geometric constraints should be different from that in CAD systems.

We analyzed more than 300 types of graphic objects to summarize the geometric constraints. First, we summarize three main geometric constraints, Intersection, Parallelism, and Perpendicularity, for linear components, and then two more for circles and arcs. They will be presented in details in the following sub-sections, respectively.

3.1. Intersection

Usually, the point of intersection is at the middle of a line, but this is not convenient for the reasoning procedure. Hence, we first decompose the lines by their points of intersection. After decomposition, the points of intersection are all at the end of lines. Thus two lines are intersected only in four types, as illustrated in Fig.1.



Fig.1 Four types of Intersection

We use a parameter *type* to represent this information and use another parameter *angle* to store the angle between the two lines. We set

$$\text{angle} = \cos(\alpha) = \frac{L_1 \cdot L_2}{|L_1| |L_2|}$$

In this definition, the parameter *angle* itself is not sufficient to fully specify the spatial relationship between two intersected lines since the angle has a direction. Thus, we use another parameter *direction* to describe this information. Consider $L_1(x_1, y_1, 0)$ and $L_2(x_2, y_2, 0)$, which are 2D vectors in 3D space, their cross product

$$L_1 \times L_2 = \left(\begin{vmatrix} y_1 & 0 \\ y_2 & 0 \end{vmatrix}, \begin{vmatrix} 0 & x_1 \\ 0 & x_2 \end{vmatrix}, \begin{vmatrix} x_1 & y_1 \\ x_2 & y_2 \end{vmatrix} \right) = (0, 0, L_z)$$

is perpendicular to the plane formed by L1 and L2, and its direction complies with the Right Hand Rule. Thus we can determine the direction by calculating L_z . In addition, we use a parameter *length* to describe the relative length of L2 to L1 ($\text{length} = |L_2|/|L_1|$).

3.2. Parallelism

Similar to Intersection, the Parallelism constraint is also described using four parameters. The first one is $distance = D(L_1, L_2) / |L_1|$, where $D(A, B)$ means the distance between A and B. The second one, $direction$, is used to describe whether L_2 is on the left or right to L_1 and the computing method is similar to the definition in Intersection. We use two additional parameters to describe their relative position and length, as shown in Fig.2.

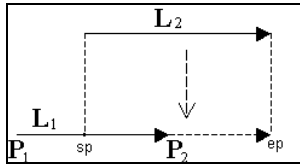


Fig.2 Start-point and end-point in Parallelism

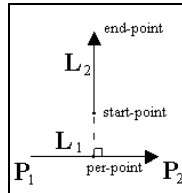


Fig.3 Perpendicularity

In Fig.2, L_1 and L_2 are parallel to each other, in which sp and ep are the projections of the endpoints of L_2 on L_1 . The following two parameters are set for L_2 accordingly:

$$start - point = \frac{sp - L_1.P_2}{L_1.P_1 - L_1.P_2}$$

$$end - point = \frac{ep - L_1.P_2}{L_1.P_1 - L_1.P_2}$$

3.3. Perpendicularity

For the Perpendicularity constraint in which two lines are intersected, we can use Intersection to represent it. In this paper, we only define the Perpendicularity in which two lines are not intersected:

1. $length = |L_2| / |L_1|$
2. $per - point$ is the perpendicular point of L_2 on L_1
3. $start - point = D(per - point, L_2.P_1) / |L_2|$
4. $end - point = D(per - point, L_2.P_2) / |L_2|$

When we calculate parameters $start - point$ and $end - point$, we set a sign to the value of them. We set it positive if the point is on the left-hand side of L_1 and negative the right-hand side. The computing method is similar to computing $direction$ in Intersection and Parallelism. For example, in Fig.3, the values of $start - point$ and $end - point$ are both positive.

3.4. Circles and arcs

We use the following four parameters to represent a circle constrained with a line (L) as illustrated in Fig.4.

1. $center - point = (cp - L.P_2) / (L.P_1 - L.P_2)$

2. $distance = D(center - point, cp) / |L|$
3. $direction : center - point$ is on left or right
4. $radius = |radius| / |L|$

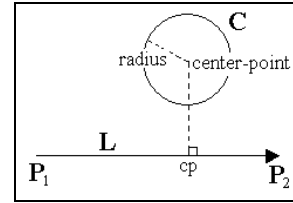


Fig.4 Geometric constraint for a circle and a line

For arcs, two additional parameters (start-angle and end-angle) are required to describe the constraint.

4. Algorithms

This section presents the Case-based Knowledge Acquisition Algorithm (CKAA) and the Knowledge-based Graphics Recognition Algorithm (KGRA) we used in the proposed framework.

4.1. CKAA

As its name tells, this algorithm is used to learn the graphical knowledge, i.e., to extract the useful geometric constraints from the graphic object examples. For easy understanding, we use a graph $G(V, E)$ to represent the graphical knowledge in which the vertexes are the vectors' sequence numbers and the edges are the geometric constraints between the vectors. In our algorithm, we first extract the Intersection constraints since its frequency in graphic objects is the highest and its computation is easier than others. For example, in Fig.5, (a) is a graphic object that consists of 6 vectors and (b) is the result after extracting all the Intersection constraints. Since we will use the graphical knowledge for the reasoning procedure, all vectors should be connected by geometric constraints. However, only Intersection constraints sometimes cannot connect all vectors, e.g., in Fig.5(b). We test this un-connectedness in $G(V, E)$ and search for the other two types of geometric constraints to connect all vectors in the graphic object and guarantee the connectedness of $G(V, E)$. Fig.5(c) is the result after adding a Parallelism constraint between Vector 0 and Vector 2.

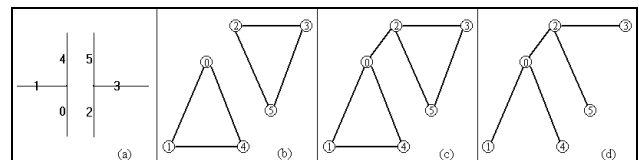


Fig.5 Intermediate results of CKAA

Although a connected $G(V,E)$ is complete for the graphical knowledge representation, it is not concise and convenient for the reasoning (recognition) procedure. Hence, we propose a reduction procedure on $G(V,E)$ using the Spanning Tree. After reduction, all the edges are necessary for graphical knowledge representation, e.g., Fig.5(d). Finally, if the graphic object contains circles or arcs, we add necessary geometric constraints for them. The following is the detail of CKAA:

Input: the vectors of a graphic object example
Output: a set of geometric constraints, $K(c_1, c_2, \dots, c_n)$
<ol style="list-style-type: none"> 1. Extract all Intersection constraints and construct $G(V,E)$ 2. If $G(V,E)$ is connected then goto Step 5 3. Divide $G(V,E)$ into a few connected sub graphs, G_1, G_2, \dots, G_n, which are not connected to each other 4. Search for any Parallelism or Perpendicularity that can connect the sub graphs. If found then add it to $G(V,E)$ and goto Step 2. If not, stop (failure) 5. Construct a spanning tree (ST) on $G(V,E)$ 6. Add geometric constraints for circles and arcs 7. Stop (success)

4.2. KGRA












As we discussed in Section 2, our recognition scheme KGRA is a reasoning procedure based on the graphical knowledge learned from examples. However, there are still some problems in practice. First of all, the directions of lines are random. Thus, when searching for a similar line, we consider the two directions. Similarly, when selecting a line to start the reasoning procedure, we test it twice using the two directions. We also set some tolerances in matching, e.g., length tolerance and angle tolerance. For the case of absence and redundancy of some vectors during the searching procedure according to the known graphical knowledge, we also set a tolerance range in the reasoning procedure. The following is the detail of KGRA.

Input:	SV: the vectors of a graphic object KD: the knowledge database TL: the set of tolerances
Variables:	CT: the temporary constructed tree SM: the set of marks
Output:	RR: the recognition result
<ol style="list-style-type: none"> 1. Select a knowledge tree K, which represents a type of graphic object, from KD. If all knowledge trees have been tried, then stop (failure) 2. Set CT empty and initialize SM 3. Select the next vector V from SV sequentially, which has not been marked in SM. Add it into CT as the root, and mark it in SM to indicate this vector has been used in the reasoning procedure 	

<ol style="list-style-type: none"> 4. Select the next edge E from K sequentially. If all edges have been traced and the number of redundant vectors does not exceed the tolerance range then stop (success) and RR is specified as an object type represented by the current K 5. Calculate the new vector V' using V and E 6. Search for a V'' in SV, which is similar to V' using the tolerances in TL. Both directions of V'' are tested 7. If V'' is found, set it as a child of V in CT and mark it in SM to indicate V'' has been used and goto Step 4 8. If V'' is not found and the number of the absent vectors exceeds the tolerant range, then goto Step 1. Otherwise, goto Step 4

5. Experiment Results

We have implemented a prototype system based on our proposed scheme. The user is asked to indicate an example by selecting a region in an engineering drawing. The system then learns its knowledge in run-time and uses this learned knowledge to recognize other similar graphic objects immediately. This graphical knowledge can also be stored in a knowledge database for future use. Hence, the system's recognition ability will be more and more powerful as the learned knowledge increases. Fig.6 is an example engineering drawing we used to test our system. The dashed rectangles are the selected examples. The recognition rates and time cost (on a PC with PII 450MHz CPU and 128M SDRAM) are listed in the following table, where L is the number of lines and C is the number of circles in the example, Co is the number of constraints generated by CKAA, Re is the number of recognized objects, Ra is the recognition rate, AT is the knowledge acquisition time in microseconds, and RT is the recognition time in milliseconds.

Object	L	C	Co	Re	Ra	AT	RT
	2	1	2	66	100%	32	167
	6	0	5	6	100%	53	23
	7	0	6	3	100%	58	25
	7	0	6	3	100%	57	37
	5	0	4	4	100%	47	28
	5	0	4	6	100%	45	30
	4	0	3	10	100%	36	29
	4	0	3	10	100%	36	28
	1	1	1	2	100%	18	3
	6	0	5	3	100%	45	40
	7	0	6	3	100%	47	50

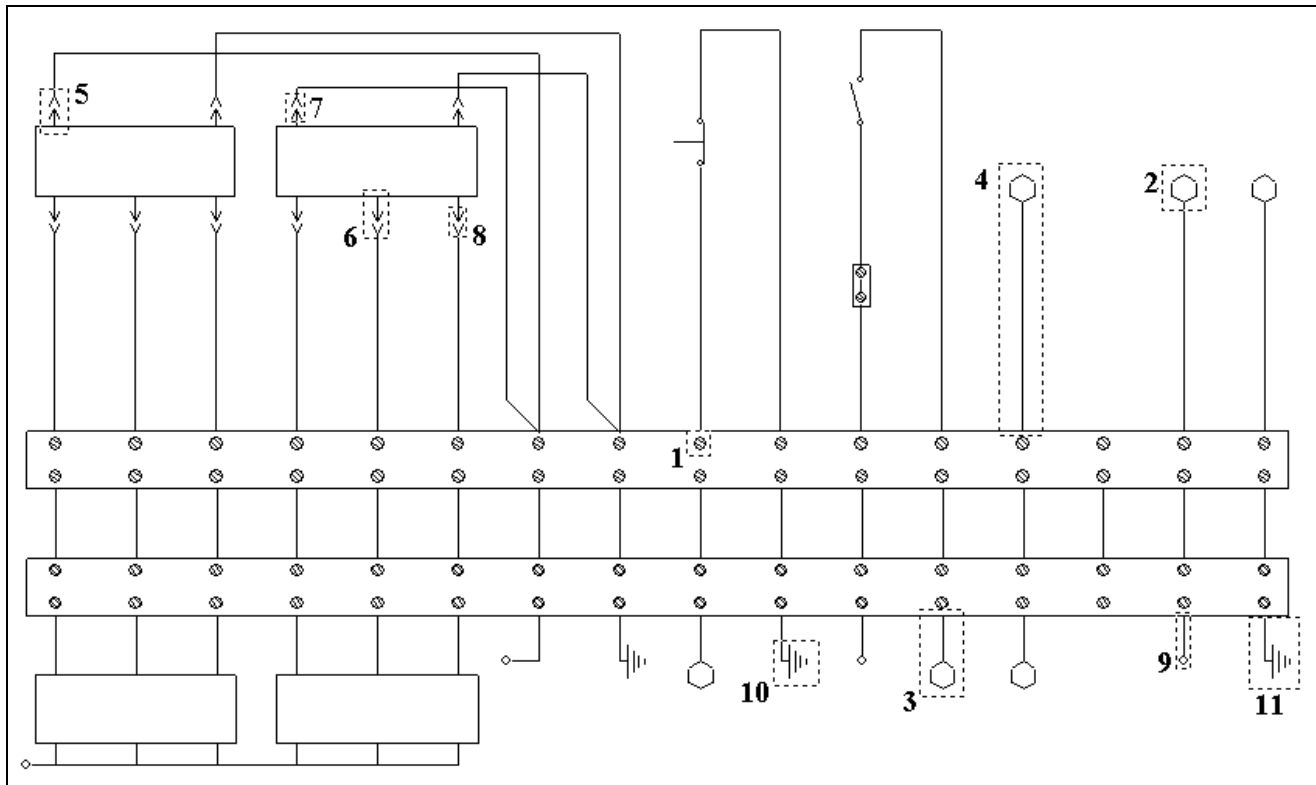


Fig.6 Some examples of graphics recognition using our prototype system on an engineering drawing

6. Conclusions and Future Work

In this paper we have proposed a new scheme for engineering drawings recognition using a case-based approach. We summarized five geometric constraints for representing the graphical knowledge. We also developed two algorithms for knowledge acquisition and graphics recognition. Experiments have shown that the proposed framework is both efficient and effective for recognizing various types of graphic objects in engineering drawings. However, there are still much work need to do for enhancing and enriching this framework. For example, our prototype system currently can only learn the graphical knowledge from single example. We believe that our scheme can also fit the cases of multiple examples from both positive and negative examples. In the future work, we will add the user's feedback, e.g., manual corrections to those mis-recognitions, to our algorithm, which will make our scheme more powerful for solving the engineering drawings recognition problem.

References

- [1]. Gader P, Forester B, Ganzberger M, Mitchell B, Whalen M, Yocum T, "Recognition of Handwritten Digits Using Template and Model Matching", *Pattern Recognition*, Vol.24, No.5, pp.421-431, 1991.
- [2]. Lee SW, Kim YJ, "A New Type of Recurrent Neural Network for Handwritten Character Recognition", *Proc. 3rd ICDAR*, Vol.1, 1995.
- [3]. den Hartog JE, ten Kate TK, Gerbrands JJ, "Knowledge-based Interpretation of Utility Map", *Computer Vision and Image Understanding*, Vol.3, No.1, pp.105-117, 1996.
- [4]. Li C, Yang B, Xie W, "On-line Hand-sketched Graphics Recognition Based on Attributed Relation Graph Matching", *Proc. 3rd Int. Conf. on Intelligent Control and Automation*, pp.2549-2553, 2000
- [5]. Chang SK, Shi QY, Yan CW, "Iconic Indexing by 2-D Strings", *IEEE Trans. on PAMI*, Vol.9, No.3, pp.413-428, 1987.
- [6]. Llado J, Marti E, Jose J, "Symbol Recognition by Subgraph Matching Between Region Adjacency Graphs", *IEEE Trans. on PAMI*, Vol.23, No.10, pp.1137-1143, 2001.
- [7]. Lee JY, Kim K, "Geometric Reasoning for Knowledge-based Parametric Design using Graph Representation", *Computer-Aided Design*, Vol.28, No.10, pp.831-841, 1996.
- [8]. Ait-Aoudia S, Hamid B, Moussaoui A, Saadi T, "Solving Geometric Constraints by a Graph-Constructive Approach", *Proc. Int. Conf. on Information Visualization*, pp.250-255, 1999.
- [9]. Fudos I, Hoffmann CM, "A Graph-Constructive Approach to Solving Systems of Geometric Constraints", *ACM Trans. on Graphics*, Vol.16, No.2, pp.179-216, 1997.