ELSEVIER

Computer-Aided Design 37 (2005) 1053–1069

COMPUTER-AIDED DESIGN

www.elsevier.com/locate/cad

A new recognition model for electronic architectural drawings

Tong Lu[a,*], Chiew-Lan Tai[b], Feng Su[a], Shijie Cai[a]

[a]State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210093, People's Republic of China
[b]Department of Computer Science, The Hong Kong University of Science and Technology, Kowloon, Hong Kong, People's Republic of China

Received 19 May 2004; received in revised form 2 November 2004; accepted 8 November 2004

Abstract

Current methods for recognition and interpretation of architectural drawings are limited to either low-level analysis of paper drawings or interpretation of electronic drawings that depicts only high-level design entities. In this paper, we propose a Self-Incremental Axis-Net-based Hierarchical Recognition (SINEHIR) model for automatic recognition and interpretation of real-life complex electronic construction structural drawings. We design and implement a series of integrated algorithms for recognizing dimensions, coordinate systems and structural components. We tested our approach on more than 200 real-life drawings. The results show that the average recognition rate of structural components is about 90%, and the computation time is significantly shorter than manual estimation time.

Keywords: Electronic architectural drawing; Recognition; Interpretation; SINEHIR model

1. Introduction

Construction structural drawings (CSDs) are a set of architectural drawings that describe the layout of various structural objects (e.g. columns, beams, slabs, walls, and holes) in a building and the mechanic design (e.g. steel bars or concrete) of these structural objects. Since they define the mechanic requirement of the entire building, they are the most important kind of architectural drawings for quantity surveying and construction. From CSDs, quantity surveyors can calculate the volume of concrete and the shape and dimension of each steel bar needed. These measurements are usually not directly marked in the CSDs, one needs to first mentally reconstruct the whole 3D model according to one's experience and the domain knowledge. Only after the relationships of all the structural objects in the building have been established, can these quantities be accurately calculated. Unfortunately this manual process is very time-consuming and error-prone.

Most of the previous research on the recognition of architectural drawings has been limited to low-level analysis of paper drawings, such as vectorization and recognition of the dimensions, texts and graphic primitives [1–5]. To our knowledge, no previous work has attempted to recognize the higher-level structural objects and extracts useful information directly from electronic CSDs for real-life construction or other applications; that is, the recognition of CSDs is a new area in graphic recognition and document interpretation. In addition to quantity surveying, the recognition is also useful for other applications, such as 4D modeling, virtual reality, and graphical retrieval system.

We note that there has been extensive research on the recognition and 3D reconstruction of mechanical parts from engineering drawings [6–13]. Due to the differences between engineering and architecture drawings, we will conclude these methods are not suitable for our intended problem [1,2].

Generic recognition and accurate interpretation of electronic CSDs are difficult for the following reasons. (1) Real-life electronic drawings may not strictly follow drafting standards and shortcuts are often used. There are also thresholds and errors in electronic drawings; for example, what is the largest distance tolerance of neighboring or connected graphic primitives? What is the angular tolerance for two parallel lines? How does one recognize structural objects when lines or arcs are missing or extraneous? (2) A CSD only captures the architect's

* Corresponding author.
E-mail addresses: lu_tong@263.net (T. Lu), taicl@cs.ust.hk (C.-L. Tai).

0010-4485//$ - see front matter © 2004 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cad.2004.11.004

intentions and major characteristic of the building, and its precision varies from one architect to another. It is difficult to develop a universal system to recognize all kinds of CSDs and to calculate accurate engineering data from them. (3) Manifold annotations (such as grid lines, dimensions, references and attribute texts) are distributed within a drawing and across multiple drawings. All the annotations, including implicit meanings of the designers, have to be recognized and integrated into a global coordinate system in order to reconstruct an accurate 3D model. (4) Structural objects exist in varied internal forms, for example, a column may be composed of lines, solids, triangles, polylines or blocks. Automatic recognition also faces the problem of adaptability to new symbols and variants of existing objects. Citing Tombre [10], 'architectural design is more or less at the crossroads between engineering and art, which makes precise analysis and reconstruction more difficult'.

Our research group has been working on the recognition of architecture drawings for the last 6 years [14–16]. Our ultimate goals are to, fully automatically, recognize all kinds of structural objects, reconstruct the entire 3D building, and calculate the useful quantities for construction workers directly from real-life electronic CSDs. During this process, very little user interaction or parameterization will be needed. In this paper, we present our successfully designed and implemented automatic recognition system, which is based on a new model called Self-Incremental Axis-Net-based Hierarchical Recognition (SINEHIR). For a typical CSD project, which is made up of about 72,000 graphic primitives (Table 1), our system automatically recognizes about 17,000 structural objects, reconstructs the 3D model, and calculates various engineering data in less than 1 h. The data covers more than 320 tons of steel of accurate lengths and shapes, and 21,480 m^2 area and 2112 m^3 volume data of concrete for various structural objects. A human usually takes at least 1.5 week to interpret and calculate such amounts of data.

The remainder of this paper is organized as follows. Section 2 describes the specifics of electronic CSDs and the overview of the proposed model. Section 3 presents the various recognition algorithms for recognizing dimensions, coordinate systems, columns, beams, walls and slabs. Section 4 gives the experiment results. Finally, in Section 5, we discuss the limitations of our system and offer the conclusions.

Table 1
CSDs of a typical architectural project[a]

CSD name	Type	Floors	Number of drawings	Number of primitives
Level table	Table	1–7	1	1756
Wall drawings	Section and table	1–2, 3–4, 5–6	3	13,670
Column drawings	Section	1–2, 3–4, 5–6	3	11,771
Beam drawings	Plane	2–3, 4–5, 6–7	3	21,280
Slab drawings	Plane	2–3, 4–5, 6–7	3	23,438

[a] A dormitory project in Nanjing.

2. Self-incremental axis-net-based hierarchical recognition model

2.1. Specifics of construction structural drawings

There are two main kinds of architectural drawings. One only expresses the idea of the architect by showing the layout of the design entities, such as windows, doors and walls. The other more complex kind, called CSD, includes a multitude of engineering data representing the mechanic requirements (such as dimensions, structural objects with accurate attributes, internal steel bars or concrete for structural objects), which are needed for quantity surveying and construction in real life. To reconstruct a 3D building and to compute accurate material quantities, we must interpret CSDs, which is the subject of our research.

A typical architectural project has several CSDs. They may be presented as plane drawings, section drawings, elevation drawings or tables for each type of structural components (columns, beams, walls). Table 1 shows an example. A CSD typically contains the following elements:

- *Dimensions*. Dimensions are used to locate structural objects. There may be dimension annotations between two parallel grid lines to indicate their exact distance. A group of parallel grid lines, labeled with serial numbers (A, B-1, B-2, C… or 1, 2/1, 2/2, 3…) is called a *grid group*. Two orthogonal and connected grid groups form an *axes net*. For example, Fig. 1 shows three axes nets, marked as A_1, A_2, and A_3 (the rightmost A_1 has horizontal and vertical grid groups, the center one A_3 has the longest grid lines).
- *Components*. Components comprise two types of structural objects: those that support the weight of the whole building (columns, beams, walls, slabs, internal steels) and the rest (such as holes). Fig. 2 shows a portion of a CSD. Fig. 3 shows the columns to be extracted from Fig. 2; Fig. 4 shows the beams and walls, and Fig. 5 shows the steel bars.
- *Component relations*. Usually no structural components are isolated. The connection mode between components must obey the mechanical constraints to support the whole building. The component relations must be correctly interpreted in order to reconstruct the whole building and accurately calculate the engineering data. Different connection modes between the same two components will generate different engineering data such as area, volume or shapes of the internal steel.

Therefore the main objectives of the recognition and interpretation of CSDs are:

(1) Extracting the dimensions and grids to build axes nets and then a local coordinate system in each CSD, and integrating call the local coordinate systems into a global 3D coordinate system;

Fig. 1. Axes nets in CSD.

(2) Analyzing structural components, interpreting them by their attributes and establishing their relations;
(3) Reconstructing 3D building and accurately calculating all the necessary structural data.

2.2. Overview of the SINEHIR Model

Previous recognition systems for architectural drawings are shape specific, with the details of the graphical constraints hard-coded into the system [1,4]. Developing such a system requires substantial effort. Still, because electronic drawings may not strictly follow drafting standards, and there are often missing or sharing graphical entities, the algorithms of such systems are not robust enough for real-life drawings. In designing a new model for recognizing electronic CSDs, we require that the recognition algorithms be shape-independent as far as possible, based on internal semantic constraints rather than visual graphical constraints. We name our model Self-Incremental Axis-Net-based Hierarchical Recognition model (SINEHIR model). When recognizing a structural component from the graphic primitives, SINEHIR first identifies its characteristic features from the more regular constituents, and then tracks the graphic objects as far as possible under the guidance and constraints of recognized objects and the domain knowledge. For instance, SINEHIR first recognizes the relatively regular objects, like dimensions and grid lines, and then analyzes columns, which are distributed around the internal grid points. Since beams are supported by columns, SINEHIR next tries to recognize segments of beams by

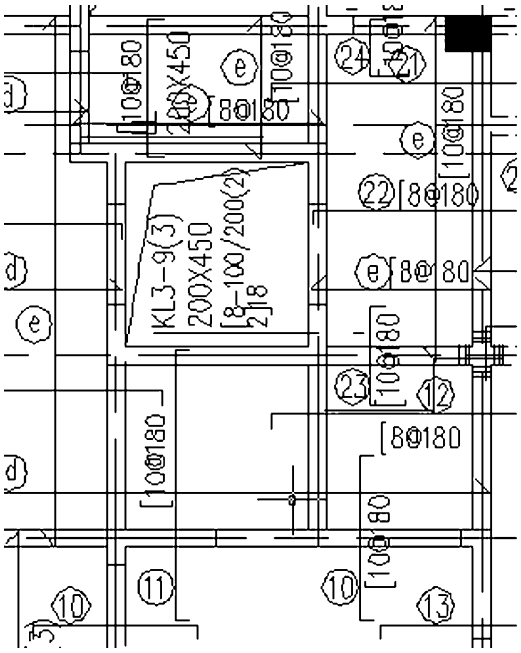

Fig. 2. Part of a CSD.

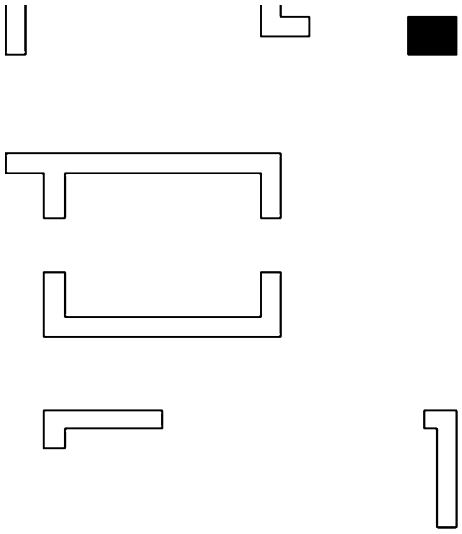

Fig. 3. Columns to be extracted from Fig. 2.

searching between columns. Finally, SINEHIR makes use of structural constraints of recognized structural components and try to track a sequence of structural components with minimal computational overhead. Throughout the recognition process, we incrementally simplify the drawing by removing objects that have been recognized; this eliminates object interference.

SINEHIR applies the following principles to analyze CSDs:

2.2.1. Integrate multiple views based on axes nets

Each CSD of a building provides parameterization information through one or more axes nets and auxiliary dimension sets. The axes nets in different CSDs only correspond to each other logically, not physically. Hence, for 3D reconstruction of the whole building, it is necessary to recognize and integrate all the axes nets within each CSD, and then integrate all of them into a global 3D coordinate system.

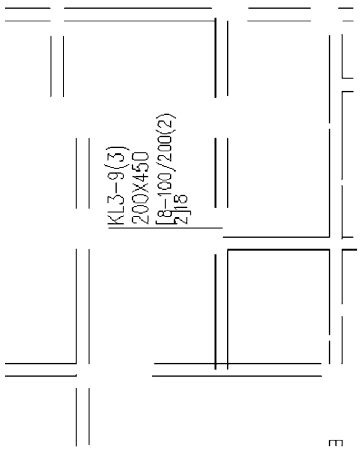

Fig. 4. Beams to be extracted from Fig. 2.

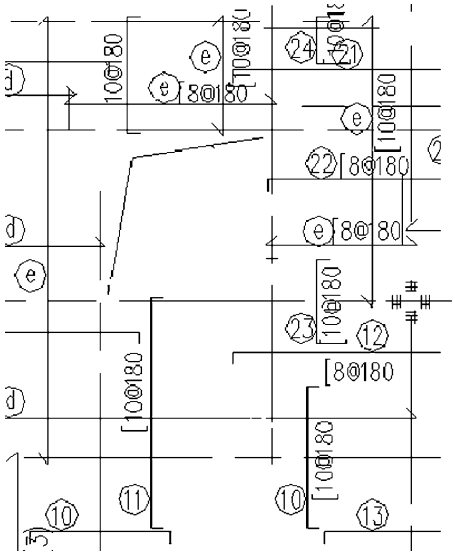

Fig. 5. Steel, grid lines and a hole to be extracted from Fig. 2.

2.2.2. Recognize structural components based on architectural semantics

It is usually not possible to recognize a structural object only from the local geometry information, especially in real-life drawings with imperfections. Imitating the way humans read the drawings, recognition and interpretation of structural components should be guided by structural semantics and their environment.

2.2.3. Integrate structural components using a hierarchical self-incremental approach

The process of recognizing CSDs includes the analysis of graphic primitives, recognition of structural components, and integration of all recognized components in different layers of the building. The order of these processes plays an important role in the incremental construction of the whole building—from columns to beams on each layer of the building, from the bottom to the top layers. Recognitions performed earlier can provide references, proofs and guidance to subsequent recognition.

3. Hierarchical self-incremental recognition method

In this section, we describe the main algorithms of SINEHIR to realize the above-mentioned three main principles of recognition and interpretation of CSDs.

3.1. Recognition of global coordinate system based on axes nets

Dimensions are distributed within individual drawings and across multiple drawings. They form the basic reference for recognition and reconstruction. SINEHIR first recognizes the dimensions and integrate them into axes nets,

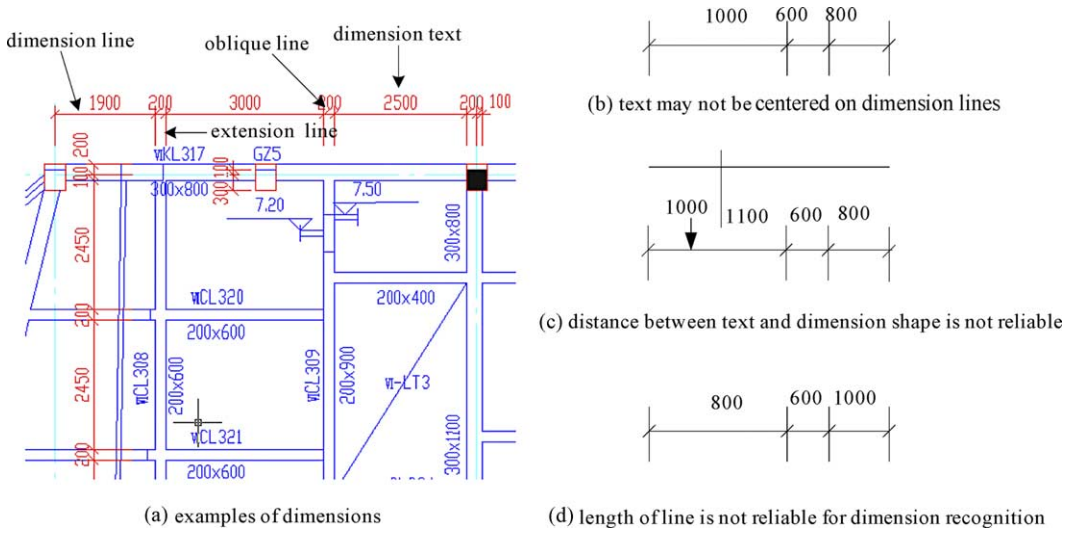

Fig. 6. Dimensions in CSDs.

then reconstructs the local coordinate system for each CSD, and finally integrates them into a global 3D coordinate system. We first define some terms.

- Physical coordinate system: a coordinate system in which the graphic primitives are drawn.
- Logical coordinate system: a coordinate system built from the dimensions or attribute annotations (like height, width, length).
- Axes net logical coordinate system: the logical coordinate system associated with an axes net.
- Local logical coordinate system: the logical coordinate system associated with a CSD.
- Global logical coordinate system: the logical 3D coordinate system of the whole building.

3.1.1. Recognition of dimensions

Jensen [13] considered a *dimension* as composed of graphics entities such as dimension lines, extension lines, leaders, arrowheads and text. In CSDs, a dimension is composed of a dimension line l_{id}, two extension lines l_{ie1} and l_{ie2}, two oblique lines l_{io1} and l_{io2}, and a dimension text t, which can be represented as $d_i\langle l_{id}, l_{ie1}, l_{ie2}, l_{io1}, l_{io2}, t_i\rangle$ (Fig. 6). A *dimension shape* is composed of all the entities of a dimension except the text, i.e. $\langle l_{id}, l_{ie1}, l_{ie2}, l_{io1}, l_{io2}\rangle$. Several dimension recognition methods have been proposed in the literature [17–20]. These methods usually recognize dimensions by starting from the detection of texts or arrowheads. They are not applicable to CSDs for three reasons. (1) Dimensions in CSDs are usually delimited by oblique lines, not arrowheads. (2) It is not reliable to recognize dimension shapes from texts because the dimension texts are not always centered on the dimension lines (Fig. 6(b)). The distance between the text and the dimension shape, and also the length of dimension lines, are not reliable for the recognition of dimension texts. For example, in Fig. 6(c) the text '1100' is nearer to a candidate dimension shape than the correct annotation text '1000'. In Fig. 6(d), the length of the dimension line of the annotated dimension text '800' is much longer than that of '1000'. Fig. 7(a) illustrates a scenario in which recognizing the dimensions either from the texts or from candidate dimension shapes would be difficult. (3) Few of the former research efforts consider accurate geometry reconstruction for quantity surveying or the integration of multiple drawings to build a global 3D coordinate system.

We propose an iterative bidirectional method for dimension recognition. We first recognize the candidate dimension shapes from the short oblique lines, and then group those dimensions that share a dimension line or extension lines into a *dimension group*. Dimensions in a dimension group are ordered sequentially along a dimension line; for example, Fig. 7 shows a group $\langle d_i, d_{i+1}, d_{i+2}\rangle$. Next, for each candidate dimension shape, we collect all its

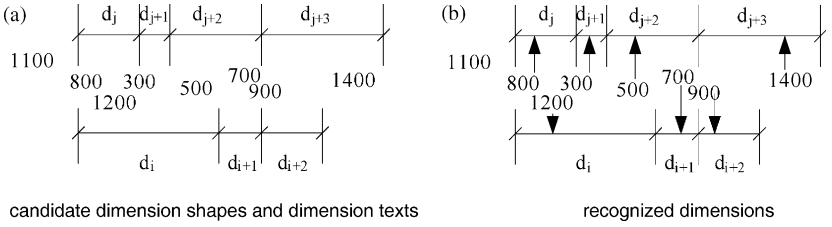

Fig. 7. An example of dimension recognition.

nearby texts, within a given threshold. At the same time, for each dimension text, we collect all the nearby candidate dimension shapes. Finally, the dimension shapes and dimension texts are matched bidirectionally—a new dimension is considered recognized if a candidate dimension shape matches a candidate dimension text, or vice versa. When a dimension is recognized, we remove the pertinent text from the sets of candidate dimension texts of all other dimension shapes, and similarly, remove the pertinent dimension shape from the sets of candidate dimension shapes of all other dimension texts.

Let L be the set of lines in a CSD, and T be the set of texts. We define

$$TextHeight = \frac{1}{n}\sum_{i=1}^{n} height(text_i)$$

where n is the number of texts in a CSD. The algorithm proceeds as follows.

1. Find all the candidate oblique lines: consider each line $l_i\langle(x_{i0}, y_{i0}),(x_{i1}, y_{i1})\rangle$ in L, add the line to the set of candidate oblique lines, OL, if it satisfies the following conditions:
   (a) $length(l_i) < TextHeight$
   (b) Neither (x_{i0}, y_{i0}) nor (x_{i1}, y_{i1}) is the endpoint of any other lines in L.
2. Find all the dimension groups: consider each line l_i in (L-OL), and try to find the dimension group that has l_i as the dimension line as follows.
   (a) Search for two oblique lines l_k, l_j, $k \neq j$, in OL, where l_i intersects both l_k and l_j. Let the intersection points be (x_1, y_1) and (x_2, y_2), respectively.
   (b) Search for two candidate extension lines in (L-OL), where l_m intersects with l_i at (x_1, y_1), l_n intersects with l_i at (x_2, y_2), and $l_i \perp l_m$, $l_i \perp l_n$ (i.e. l_m and l_n are two). If successful, create a candidate dimension shape $d_i\langle l_i, l_m, l_n, l_k, l_j\rangle$;
   (c) If d_i is the first candidate dimension shape of l_i found, create a new dimension group dg containing only d_i,
   else let the current group be $dg = \langle d_1\langle l_i, l_{1e1}, l_{1e2}, l_{1o1}, l_{1o2}\rangle, \ldots, d_s\langle l_i, l_{se1}, l_{se2}, l_{so1}, l_{so2}\rangle\rangle$, and add d_i into dg at the correct position as follows:
      (i) If $l_m = l_{re2}$, $l_n = l_{(r+1)e1}$, for some $1 < r < s$, then insert d_i into dg between d_r and d_{r+1}
      (ii) If $l_n = l_{1e1}$, then insert d_i into dg as the first dimension shape.
      (iii) If $l_m = l_{se2}$, then add d_i into dg as the last dimension shape.
3. Find all the candidate dimension texts for each dimension shape and all the candidate dimension shapes for each dimension text as follows. Consider each candidate dimension shape found in step 2, $d_i\langle l_{id}, l_{ie1}, l_{ie2}, l_{io1}, l_{io2}\rangle$, for each text t_j in T, make t_j a candidate dimension text for d_i and make d_i a candidate dimension shape for t_j if distance$(t_j, l_{id}) < \xi$, where $\xi = 4 \times TextHeight$, and t_j appears between the extension lines l_{ie1} and l_{ie2}.
4. Perform bidirectional matching. Let ct_{di} denotes the set of candidate dimension texts of dimension shape d_i and let cs_{tj} denotes the set of candidate dimension shapes of text t_j.
   (a) Consider every ct_{di} that is a singleton set: $ct_{di} = \{t_k\}$. Output $\langle d_i, t_k\rangle$ as a recognized dimension. Remove ct_{di} and delete t_k from the other $ct_{dj} (j \neq i)$.
   (b) Consider every cs_{ti} that is a singleton set: $cs_{ti} = \{d_k\}$. Output $\langle d_k, t\rangle$ as a recognized dimension. Remove cs_{ti} and delete d_k from the other $cs_{tj} (j \neq i)$.
   (c) Repeat this step until no new dimension is recognized;
5. If some ct_{di} has more than one items and ($\xi \geq TextHeight$), then set $\xi = \xi/2$ and goto 4; else match the remaining candidate dimension shapes and texts according to distance, or warn the user.

Fig. 7 shows two dimension groups to be recognized and Table 2 shows the incremental recognition results. After step 3, the set of candidate dimension texts of d_i is {800, 300, 1200, 500}. After the first iteration of step 4, the set is reduced to {800, 1200, 500} because 300 is matched by d_{j+1} and thus is removed from d_i and d_j. After another iteration of step 4, the set is further reduced to {800, 1200}. When only one text is left for a dimension shape, the dimension is considered successfully recognized. After step 5, 1200 is matched to d_i according to distance. The text 1100 is not used because it is not between two extension lines, thus not a dimension text.

3.1.2. Recognition of axes nets and axes nets-based geometry reconstruction

There are usually hundreds of dimensions distributed in all the CSDs in a project. To reconstruct the global logical coordinate system, the first step is to normalize all the recognized dimensions.

Table 2
Results of running the dimension recognition algorithm on Fig. 7(a)

	d_i	d_{i+1}	d_{i+2}	d_j	d_{j+1}	d_{j+2}	d_{j+3}
Step 3	{800, 300, 1200, 500}	{700, 900}	{900}	{800, 300, 1200}	{300}	{500, 700, 900}	{900, 1400}
Step 4, iteration 1	{800, 1200, 500}	{700}	–	{800, 1200}	–	{500, 700}	{1400}
Step 4, iteration 2	{800, 1200}	–	–	{800, 1200}	–	{500}	–
Step 5	{800}	–	–	{1200}	–	–	–

The two end points of a dimension are the intersection of its dimension line with the two extension lines. (If dimensions are represented with short oblique lines, then the end points are usually also intersection of oblique lines and extension lines.) For brevity, we define $a_1 \approx a_2$ if and only if $|a_1 - a_2| < \xi$, where ξ is the following threshold (calculated from only information in the CSD, before any high-level components are recognized):

$$\xi = TextHeight/20 \tag{1}$$

Let the two end points of a dimension d_i be (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}). Without loss of generality, we assume $x_{i1} \leq x_{i2}$, and if $x_{i1} \approx x_{i2}$, then $y_{i1} \leq y_{i2}$; that is, the dimension vector $\boldsymbol{v}_i = \langle (x_{i1}, y_{i1}), (x_{i2}, y_{i2}) \rangle$ is directed to the right and upward.

Two dimensions d_i and d_j are connectable if their corresponding dimension vectors $\boldsymbol{v}_i = \langle (x_{i1}, y_{i1}), (x_{i2}, y_{i2}) \rangle$ and $\boldsymbol{v}_j = \langle (x_{j1}, y_{j1}), (x_{j2}, y_{j2}) \rangle$ satisfy the following condition:

$$[|\theta_i - \theta_j| < \theta_\xi] \wedge [[x_{i2} \approx x_{j1} \wedge y_{i2} \approx y_{j1}]] \vee [x_{i1} \approx x_{j2} \wedge y_{i1} \approx y_{j2}]]$$

where θ_i and θ_j are the angles of $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$, with respect to the horizontal line, and $\theta_\xi(\boldsymbol{v}_i, \boldsymbol{v}_j)$ is the angle threshold defined as

$$\theta_\xi(\boldsymbol{v}_i, \boldsymbol{v}_j) = \frac{TextHeight}{\max(\min(length(\boldsymbol{v}_i, \boldsymbol{v}_j)), TextHeight)} \times MAX\_ANGLE\_THRESHOLD \tag{3}$$

and it decreases when the lengths of $\boldsymbol{v}_i$ and $\boldsymbol{v}_j$ increase. In our experiments, we set *MAX_ANGLE_THRESHOLD* to 3. If the dimensions are represented as arcs (angular dimensions), then d_i and d_j are connectable only if

$$[[(x_{i1} - C_{jx})^2 + (y_{i1} - C_{jy})^2]^{1/2} \approx R_j] \wedge [[x_{i2} \approx x_{j1} \wedge y_{i2} \approx y_{j1}]] \vee [x_{i1} \approx x_{j2} \wedge y_{i1} \approx y_{j2}]] \tag{4}$$

where (C_{jx}, C_{jy}) is the center of the arc of the dimension shape d_j, and R_j is the radius.

After the dimension normalization, we select those dimensions whose extension lines are connected to or near some grid annotations (such as 'A', 'B', 'C' or '1', '2', '3'), and organize them into *grid groups* if they are connectable according to (2). An extension line of a dimension with a grid annotation is also called a *grid line*. A merged grid group can be represented as

$$GG = \langle \boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3, \ldots, \boldsymbol{v}_m \rangle$$

where $\boldsymbol{v}_i$ is a dimension vector. To remove redundant information, we also merge those recognized grid groups that share the same extension lines but are not connectable according to (2), e.g. parallel grid groups as shown in Fig. 8. Finally, we add all the nearby grid annotations (numbers or alphabets in a circle) to the corresponding grid groups; they will be used for determining the internal grid points in Section 3.2.1.

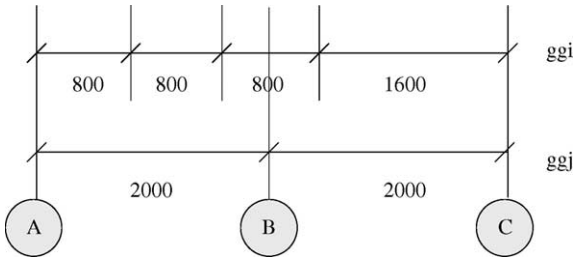

Fig. 8. Parallel grid groups sharing the same extension lines.

Given two grid groups $gg_i = (\boldsymbol{v}_{i1}, \boldsymbol{v}_{i2}, \boldsymbol{v}_{i3} \ldots \boldsymbol{v}_{im})$ and $gg_j = (\boldsymbol{v}_{j1}, \boldsymbol{v}_{j2}, \boldsymbol{v}_{j3} \ldots \boldsymbol{v}_{jn})$, $m \geq 1$, $n \geq 1$, where $\boldsymbol{v}_{ik} = \langle (x_{ik1}, y_{ik1}), (x_{ik2}, y_{ik2}) \rangle$. An axes net $\langle gg_i, gg_j \rangle$ is considered recognized if the vectors in gg_i and gg_j are orthogonal and near each other. Without loss of generality, we shall assume that gg_i is directed somewhat to the right and gg_j is directed somewhat upward. If the groups gg_i and gg_j are composed of angular dimensions, then we build an angular axes net if $x_{i11} \approx x_{j11}$ and $y_{i11} \approx y_{j11}$.

3.1.3. Reconstruction of global logical coordinate system

Next we construct coordinate systems from axes nets. A CSD project has only one global logical coordinate system, but always has several local logical coordinate systems, each associated with one CSD. In each local logical coordinate system, there may also exist several axes net logical coordinate systems. In this case, SINEHIR needs to select one of them to be the benchmark axes net and recursively transforms all the axes nets nested in this axes net. We store these parent–child relationships among the axes nets in a CSD as a tree, with the benchmark axes net at the root. For example, for the CSD in Fig. 1, suppose A_1 is selected as the benchmark axes net (because it has horizontal and vertical grid groups, alternatively A_3 may be selected because it has the longest grid lines), then A_3 is nested in A_1 at O_3, hence SINEHIR defines A_3 as a child of A_1. Similarly, A_2 is nested in A_3 at O_2, hence A_2 is identified as a child of A_3.

The detailed algorithm for constructing the global logical coordinate system is as follows.

1. Transform all the axes nets in the same CSD into a local logical coordinate system.
   (a) Consider each axes net ax_k with grid groups $\langle gg_i, gg_j \rangle$. Build a right-handed logical coordinate system $F(ax_k)$ as follows: let the intersection point of the vectors of gg_i and gg_j ($\boldsymbol{v}_{ggi}$ and $\boldsymbol{v}_{ggj}$) be the origin of $F(ax_k)$, the direction of $\boldsymbol{v}_{ggi}$ and $\boldsymbol{v}_{ggj}$ be the x and y axis of $F(ax_k)$, respectively. Mark the coordinates on the x axis and y axis according to the dimension texts along the sorted dimensions of gg_i and gg_j.
   (b) Identify the parent–child relationships of all the $F(ax_k)$'s as follows. If the origin of a coordinate system $F(ax_k)$ is inside another coordinate system $F(ax_r)$, then make $F(ax_k)$ a child of $F(ax_r)$.

(c) Traverse the parent–child tree structure from the leaves, transform a coordinate system $F(ax_k)$ to its parent coordinate system $F(ax_r)$ using the following transformation matrix:

$$M = \begin{pmatrix} \cos\theta & -\sin\theta & -x_0\cos\theta + y_0\sin\theta \\ \sin\theta & \cos\theta & -x_0\sin\theta - y_0\sin\theta \\ 0 & 0 & 1 \end{pmatrix} \quad (5)$$

where (x_0, y_0) are the coordinates of the origin of $F(ax_k)$ with respect to $F(ax_r)$, and θ is the rotation angle needed to align the axes of $F(ax_k)$ with those of $F(ax_r)$. At the root, we obtain the logical coordinate system of the CSD.

2. Among all the local logical coordinate systems corresponding to the CSDs, select the one with the longest x or y axis as the benchmark coordinate system. Integrate all the local logical coordinate systems by translating or rotating each of them into the global logical coordinate system.

The integration of local coordinate systems into a global logical coordinate system eliminates several problems: a multitude of dimension in different views, the complexity of isomerous coordinate systems, and intersection of different coordinate systems. Inconsistency of dimension sets can also be easily checked.

3.2. Structural-components recognition algorithms

Most of the current recognition methods for architectural or engineering drawings are bottom-up methods. Bottom-up approaches begin with examining neighboring pixels, and are mostly used to solve lower level interpretation tasks, such as vectorization. After that, prior to recognition of high-level structural objects, parameterization is needed to correct all the distortions or errors [3–5,21]. Since our objective is to automatically recognize structural objects directly from electronic CSD drawings, bottom-up methods are not applicable. SINEHIR adopts a top-down strategy, starting from global relationships and constraints of structural objects based on architectural knowledge. Requiring additional parameterization or editing operations during the recognition process is unrealistic and infeasible for the following reasons: (1) multitude of manual interactions will greatly decrease the efficiency of recognition system; (2) we cannot ensure that all the errors are corrected by the users before the recognition starts; (3) most importantly, the users of this kind of systems are always real-life constructors, designers or quantity surveyors who cannot understand why editing is necessary or do not know how to edit all the distortions or incompleteness in the drawings (which can be easily understood by humans).

The SINEHIR model is not based only on geometrical constraints; semantic and domain knowledge are used as far as possible to circumvent problems due to imperfections in drawings, such as lines are disjoint from the primitives when they should be connected. Since the outlines of structural objects are usually not closed due to imperfections, geometry-based methods are not suitable for the recognition of real-life CSDs. These methods include shape grammars, rule-based, network-based and template-matching recognition methods [1,4,22–24]. In contrast, SINEHIR first transforms all the primitives that could represent structural objects (such as polylines, solids, blocks) into free lines, and merge all the connected and collinear lines into long lines. After that, SINEHIR tracks the candidate free lines around each internal grid point (intersection of previously recognized grid lines) to find the columns. Because at most one internal grid point falls within a column's outline and at most one column lies near an internal grid point, we build a bounding box for each internal grid point enclosing its four neighboring grid points. Then we use this box to constrain the recognition of columns distributed around this internal grid point. After the columns are recognized, those free long lines crossing multiple columns are split into shorter ones to facilitate the recognition of beam segments between columns using a threshold calculated based on the recognized columns. These recognized columns and segments form seed components. Finally, the less regular structural components are recognized by attempting to extend from the seed components. This is a dynamic process within the CSD environment and based only on the domain knowledge, so it is feasible to recognize variants of structural objects despite drawing imperfections.

3.2.1. Column recognition based on section tracking

Columns are usually distributed around the internal grid points of axes nets. They support beams, slabs, rooms or other structural objects, and are the first to be built in real-life construction. Columns in real-life drawings may be represented in many shapes, so it is important to design an algorithm that is general and can recognize variants of known structural objects. Fig. 9 shows some examples of columns. The external outlines represent the column shapes, and the internal red lines indicate the column steel bars. The estimation of steel bars in columns will strongly depend on the column shape. Shape grammars have been proven to be efficient in recognition of engineering designs [22,23] and certain types of architectural drawings. Given a shape, they rely on matching variants of shapes generated from shape rules. Using shape grammars to recognize objects in CSDs would be difficult. Very often, part of the column outline is shared or overlapped with grid lines, neighboring symbols or components, thus the shape of a candidate column may not be closed. Without any hints that a column might exist, it is difficult to judge whether a shared or overlapped line should be used. Moreover, columns that appear to have the same shape may be composed of completely different kinds of graphic primitives in different CSDs; for example, a rectangle shape may be composed of lines, polylines, triangle solids, or rectangle solid. In fact, it is unreliable to

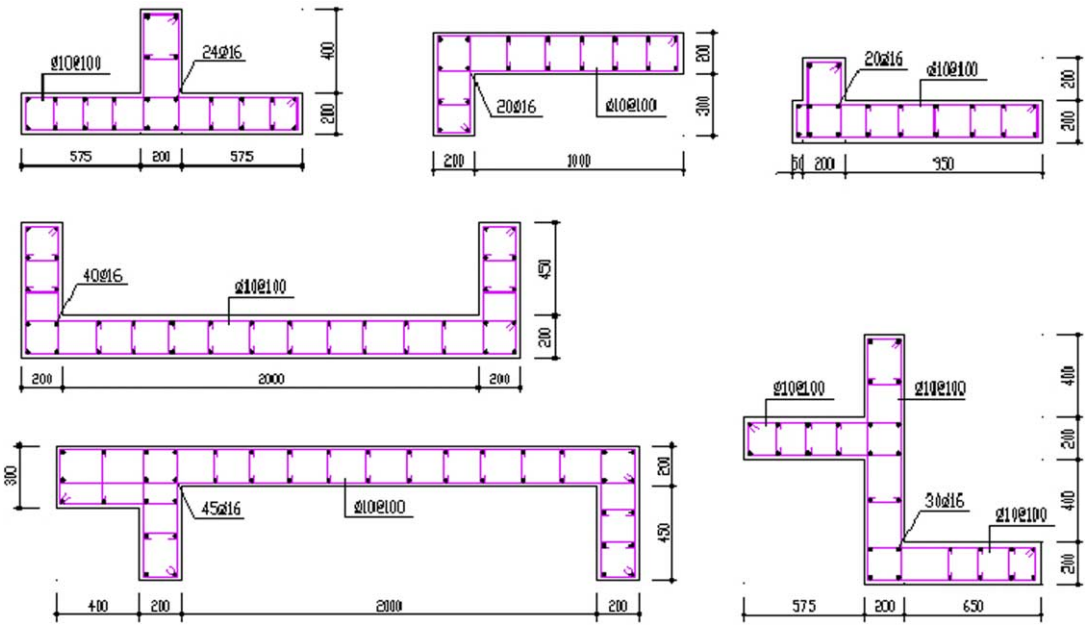

Fig. 9. Source column sections.

recognize objects from thousands of overlapping or connected lines in a real-life CSD.

SINEHIR uses a section-tracking method to dynamically analyze columns of various shapes. In the real-life drawings, columns are usually located around internal grid points. So we first calculate all the internal grid points of an axes net, and for each of the grid points, we compute a bounding box and explode all the neighboring lines (which may be part of polylines, solids or blocks) which intersect this bounding box into free lines. Next we try to trace closed paths from these free lines to find all the loops that potentially represent columns. Finally, we omit those loops that are too small or too big (i.e. do not meet the size and proportion constraints of the CSD). The detailed algorithm is as follows.

1. Suppose an axes net is denoted as $an_k = \langle gg_i(\boldsymbol{v}_{i1}, \boldsymbol{v}_{i2}, \boldsymbol{v}_{i3} \ldots \boldsymbol{v}_{im}),\ gg_j(\boldsymbol{v}_{j1}, \boldsymbol{v}_{j2}, \boldsymbol{v}_{j3} \ldots \boldsymbol{v}_{jn}) \rangle$. Calculate the intersection of all the grid lines in gg_i and gg_j. Let (x_{ikjl}, y_{ikjl}) be the intersection point of the grid lines $\boldsymbol{v}_{ik}$ and $\boldsymbol{v}_{jl}$, $1 \le k \le m$, $1 \le l \le n$.
2. For each internal grid point (x_{ikjl}, y_{ikjl}), check if there is a column around it:
   (a) Define a bounding box for (x_{ikjl}, y_{ikjl}) as $\{(x,y) | x_{i(k-1)jl} < x < x_{i(k+1)jl},\ y_{ikj(l-1)} < y < y_{ikj(l+1)},\ 1 \le k \le m,\ 1 \le l \le n\}$. Explode all the neighboring polylines, solids, and blocks that intersect with this bounding box into free lines (including those lines that are part of other recognized objects, like grid lines).
   (b) Omit those free lines that cannot candidate column lines, e.g. free lines nearest to tiny circles in the bounding box because they are lines of steel bars if tiny circles are present. Let the remaining set be *SL*.
   (c) Trace the closed outline of each candidate column in *SL*:
      i. Compute all the intersection points of the lines in *SL* and store it in a set *V*.
      ii. If two points in *V* are on the same line in *SL*, create an edge connecting them and store it in an edge set *E*.
      iii. Trace a closed path by depth-first search in *E*. Suppose the current edge is E_i and the candidate next edge under consideration is E_j. The edge E_j is added to the path if E_i and E_j share a common point in *V* and E_i and E_j are not parallel. If a loop is found, then we have found a candidate column. Repeat this step until all the edges in *E* have been considered.
   (d) Omit those candidate columns that are too small or too large compared with the size of the bounding box.

Fig. 10 shows an example of tracking the outline of a column. We do not start with a set of known shapes and attempt to do shape matching. Instead we track closed shapes using the constraints derived from architectural knowledge (e.g. columns exist around internal grid points, size and proportion constraints of columns in CSDs). In this way, variants or new shapes can be tracked and recognized. When all the columns in a CSD have been recognized, the details (e.g. internal steel bars, dimensions) of the source columns are copied to their corresponding columns without attributes in other CSDs. SINEHIR first checks the consistency of the two columns by matching their outlines: starting from the longest side of each column, traverse anticlockwise and match the convexity/concavity of

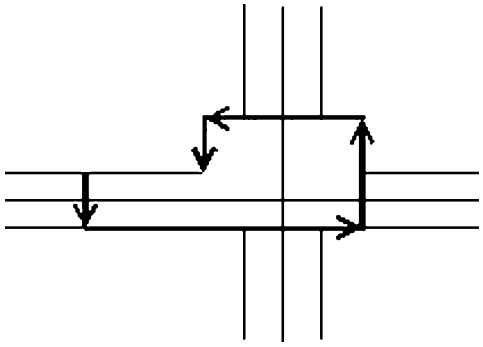

Fig. 10. Recognizing a column by section tracking.

the vertices. If all the boundary lines and vertices match, SINEHIR calculates the angle between the two corresponding longest lines and copy the details.

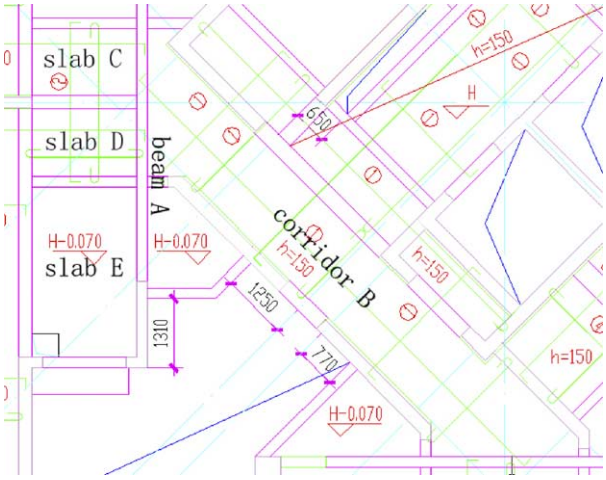

Fig. 11. Beam A should not be recognized as a whole.

3.2.2. Segment recognition based on semantic relation

Beams and walls are usually represented as pairs of parallel outlines whose projections onto each other overlap partly or completely. Fig. 4 shows some beams colored in pink. The shape of beams appears simple, but because many lines overlap and are parallel to each other, it is very difficult to decide what kind of parallel pairs may represent beams. The lines of a beam may appear in different colors or even in the drawings of other building layers. Furthermore, a beam is often not represented as a closed shape, hence we neither use the section-tracking method, nor any other methods based on geometrical constraints for symbol recognition [4,5,24]. We must design a new top-down method under the guidance of semantic constraints to correctly recognize beams from thousands of lines in real-life CSDs.

To simplify and abstract the relationships of various kinds of structural objects for the later 3D reconstruction, we introduce *nodes* and *segments*. They form the two basic kinds of elements in the SINEHIR model. The recognized columns are the nodes. A segment is a minimal portion of a parallel line pair. A parallel line pair that spans multiple columns or crosses parallel line pairs are divided into segments. We called a segment that is between two columns a *seed segment*, and a portion that is between a column and another parallel line pair, or between two parallel line pairs, an *extended segment*. We introduce segments because: (1) the lengths of the two lines in a parallel pair could be different in real-life CSDs and long lines are often shared by other parallel pairs to improve drawing efficiency; (2) if long parallel pairs are recognized in their entirety, then it is difficult to recognize other objects that are composed of shorter lines; for example, in Fig. 11, if two vertical parallel lines of beam A are recognized as a whole, then it is difficult to recognize the surrounding components which are bounded by only part of beam A (e.g. corridor B, slab C, D or E). The recognition of slabs will be introduced in Section 3.2.3. Each junction area formed by two pairs of perpendicular parallel lines will be recognized as an *extended node*. We will represent the relationships of nodes and segments by a *component relation graph*, with the nodes as the vertices and the segments as the edges (Fig. 16).

3.2.2.1. Recognition of seed segments. Our semantic-relations-based segment recognition algorithm starts from recognizing seed segments. To recognize seed segments, we first scan all the recognized columns and find free candidate lines between every two columns (Fig. 12). We split long lines that cross two columns, and create new candidate free lines of shorter lengths. Next, we identify all possible candidate parallel pairs from these free lines, and remove those that do not satisfy the semantic constraints (e.g. the draught lines of annotation 300×800 in Fig. 13(a)). Let C be the set of the recognized columns, and F be the set of free lines (lines not used by columns, dimensions or other recognized objects). The recognition algorithm for seed segments considers each pair of two columns c_i and c_j in C, with (x_i, y_i) and (x_j, y_j) as their physical coordinates of their

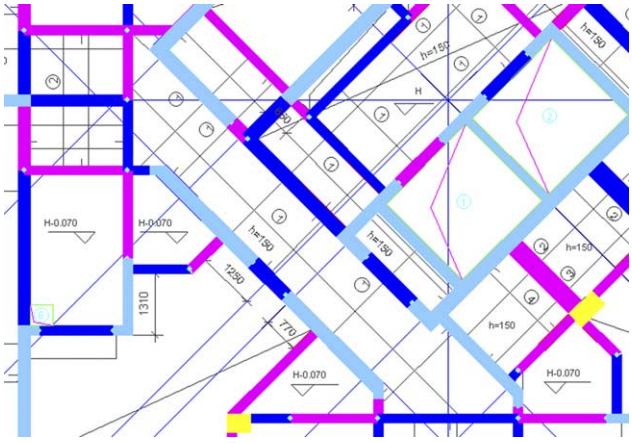

Fig. 12. Recognized segments from Fig. 9.

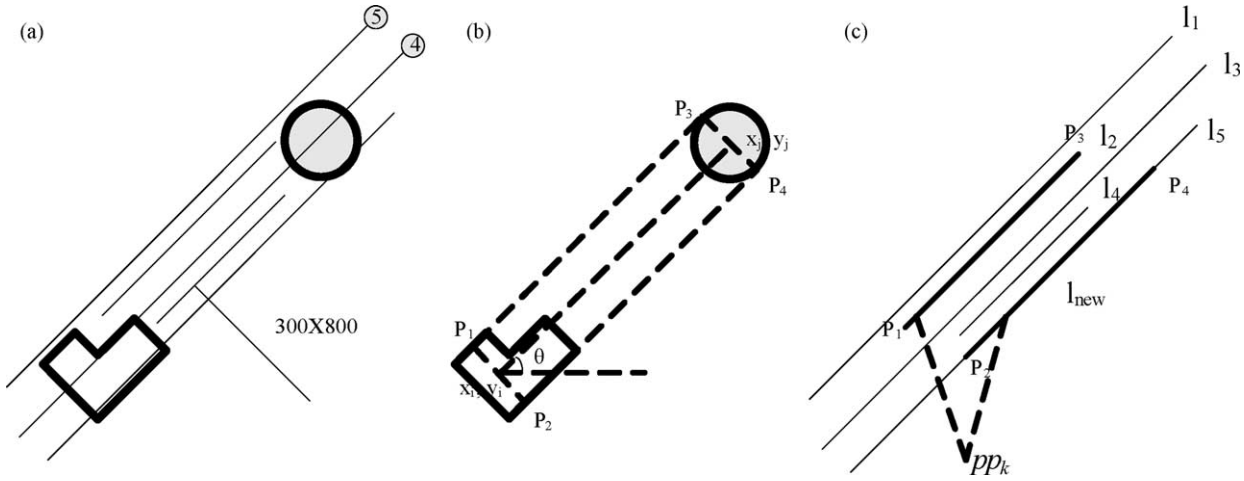

Fig. 13. Recognizing seed segments. (a) Parallel lines between two columns (b) a bounding box of two columns (c) a candidate parallel-line pair.

centroids, respectively, and tries to find a seed segment between the columns.

1. Build a bounding box for the two columns c_i and c_j. Let the four corners of the box be $p_1 = (x_i - \Delta x, y_i + \Delta y)$, $p_2 = (x_i + \Delta x, y_i - \Delta y)$, $p_3 = (x_j - \Delta x, y_j + \Delta y)$, $p_4 = (x_j + \Delta x, y_j - \Delta y)$, where $\Delta x = d\sin\theta$, $\Delta y = d\cos\theta$, with $d = \max(width(c_i), width(c_j))/2$, and $\theta = \text{atan}((y_j - y_i)/(x_j - x_i))$ assuming $(x_j \neq x_i)$. (Fig. 13(b)). Let the two sides of the box at column c_i and c_j be $l_i = p_1p_2$ and $l_j = p_3p_4$, respectively.
2. Consider each free line f connecting (x_{f1}, y_{f1}) and (x_{f2}, y_{f2}) and check if it is a candidate segment line. Let S be the target set of candidate segment lines and initialize $S = \{\}$.
   i. If (x_{f1}, y_{f1}) and (x_{f2}, y_{f2}) are both inside the box, add f to S, where ξ is calculated as follows:
   $$\xi = \frac{1}{2n}\sum_{i=1}^{n} width(node_i), \text{ where } n \text{ is } sizeof(C) \quad (6)$$
   ii. If (x_{f1}, y_{f1}) is inside the box, and the free line f intersects the side l_i of the box at (x_a, y_a), where $x_{f1} \le x_a \le x_{f2}$, $y_{f1} \le y_a \le y_{f2}$, $x_j - \Delta x \le x_a \le x_j + \Delta x$, $y_j - \Delta y \le y_a \le y_j + \Delta y$, add a new line l_{new} connecting (x_{f1}, y_{f1}) and (x_a, y_a) to S. Perform a similar test on (x_{f2}, y_{f2}) and the other side l_j of the box (e.g. Fig. 13(c)).
   iii. If the free line f intersects with the side l_i of the box at (x_a, y_a), and with the other side l_j of the box at (x_b, y_b), where $x_{f1} \le x_a, x_b \le x_{f2}$, $y_{f1} \le y_a, y_b \le y_{f2}$, $x_i - \Delta x \le x_a \le x_i + \Delta x$, $x_j - \Delta x \le x_b \le x_j + \Delta x$, $y_i - \Delta y \le y_a \le y_i + \Delta y$, $y_j - \Delta y \le y_b \le y_j + \Delta y$, then add a new line l_{new} connecting (x_a, y_a) and (x_b, y_b) to S;
3. Find all the candidate parallel line pairs from the set S. Let PP be the target set of candidate parallel line pair and set $PP = \{\}$. Consider every two free lines l_m and l_n in S and add $pp_k\langle l_m, l_n\rangle$ to the set PP if the following conditions hold:
$$[|\theta_m - \theta_n| < \theta_\xi] \wedge [0.4 \le overlaplength(l_m, l_n)/\min(length(l_m), length(l_n)) \le 1] \quad (7)$$
where θ_m and θ_n are the angles of l_m and l_n, with respect to the horizontal line, $\theta_\xi(l_m, l_n)$ is the angle threshold calculated from (3), $overlaplength(l_m, l_n)$ is the overlapping length of l_m and l_n, and 0.4 is an empirical minimal scale for overlapping.
4. Consider every candidate parallel line pair pp_k in PP and delete it if one of the following conditions is met:
   i. $\text{width}(pp_k) < \text{width}(pp_{\max})$, where $pp_{\max}$ is the parallel line pair with maximum width in PP.
   ii. starting from column c_i (respectively c_j), one can visit along pp_k and other parallel pairs in PP and back to column c_i (respectively c_j) (because no beams loop back to the same column).

The remaining parallel line pair is a seed segment if PP is not empty, and at most only one beam that connects the two columns in the given direction will be found. If PP is empty, no beams will be recognized.

Fig. 13 shows an example. There are five free lines, l_1 to l_5 in Fig. 13(a). The line l_1 is excluded because it is outside the bounding box. At step 3, the line l_5 is truncated to get a new line l_{new}, eliminating the problem of recognizing long lines in their entirety. At step 4, $PP = \{\langle l_2, l_3\rangle, \langle l_2, l_4\rangle, \langle l_2, l_{new}\rangle, \langle l_3, l_4\rangle, \langle l_3, l_{new}\rangle, \langle l_4, l_{new}\rangle\}$. After step 4, only one parallel pair is left, $PP = \{\langle l_2, l_{new}\rangle\}$, which is the correct seed segment. Semantic constraints could be helpful in accelerating this process; for example, l_4 may be excluded earlier on if it is previously recognized as a draught line.

Real-life electronic drawings often do not strictly follow drafting standard. For example, l_2 does not intersect the outlines of the two columns at all; l_5 is too long, overshooting both columns (may also cross other objects). The segment between those two columns cannot be recognized from its

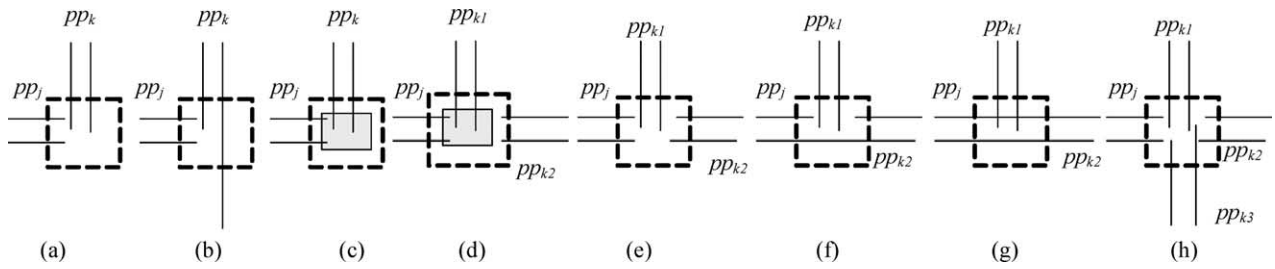

Fig. 14. Examples of extended segments.

boundary outline because it is not closed. Only under the guidance of recognized components can such candidate segments be recognized as seed segments.

3.2.2.2. Recognition of extended segments. An extended segment is a minimal portion of a parallel pair delimited by a column and another parallel pair, or by two parallel pairs. Compared with the recognition of seed segments, there are fewer clues for the recognition of extended segments from thousands of lines in CSDs. A feasible way is to start from recognized columns and seed segments. This approach has proven to be efficient in our experiments with nearly two hundred real-life CSDs.

Let C denote the set of nodes (initially consisting of only the recognized columns), F denote the set of free lines, and SS denote the set of recognized seed segments. Consider each node c_i in C with one or more recognized segments connected to it. Let one of the connected segments be $pp_j\langle l_{j1}, l_{j2}\rangle$. We try to recognize other segments connected to node c_i.

1. recalculate a threshold ξ from SS:

$$\xi = \frac{2}{n}\sum_{i=1}^{m} width(segment_i), \quad m = sizeof(SS) \tag{8}$$

2. create a square bounding box for c_i with side of length 2ξ;
3. search for a candidate parallel-line pair $pp_k\langle l_{k1}, l_{k2}\rangle$ from the set of free lines F such that
   (a) one end of l_{k1} or l_{k2} is inside the box, and
   (b) the other end of l_{k1} or l_{k2} is connected to another node $c_k(k \neq i)$; output pp_k as an extended segment.

Fig. 14 shows some recognizable extended segments. Imperfections in drafting may exist; for example, a column may not be drawn explicitly; instead two pairs of parallel lines perpendicular to each other may be drawn to intersect at an area that represents a column semantically. Our proposed semantic-relation-based method successfully resolves all these cases.

Calculating thresholds is always a big problem in recognition of drawings. We use a dynamic adjustment method to compute threshold as more components are recognized. At the beginning, we calculate the distance threshold based on the physical coordinates of the texts using formula (1). After the columns are recognized, we recalculate the threshold based on the size of the recognized columns using formula (6) to recognize the seed segments, which are connected to columns. After the seed segments are recognized, we have more information on the average physical width of seed segments, which will help us decide which parallel lines may constitute the extended segments in the complex environment, so we recalculate the threshold by formula (8).

After the extended segments are recognized, we find their intersections with other seed segments or other extended segments and split long segments into minimal portions. During this process, each connection area formed by two segments is recognized as an extended node and is added to the set C, which initially contains only the recognized columns. Then these newly added extended nodes bring about another round of recognition of extended segments. The process is ended when no extended node or extended segment is recognized. In that sense the node and segment recognition is a self-growing process.

Fig. 15 shows the final result of the node and segment recognition of the CSD in Fig. 2. Fig. 16 shows the same recognition result in the form of a component relation graph.

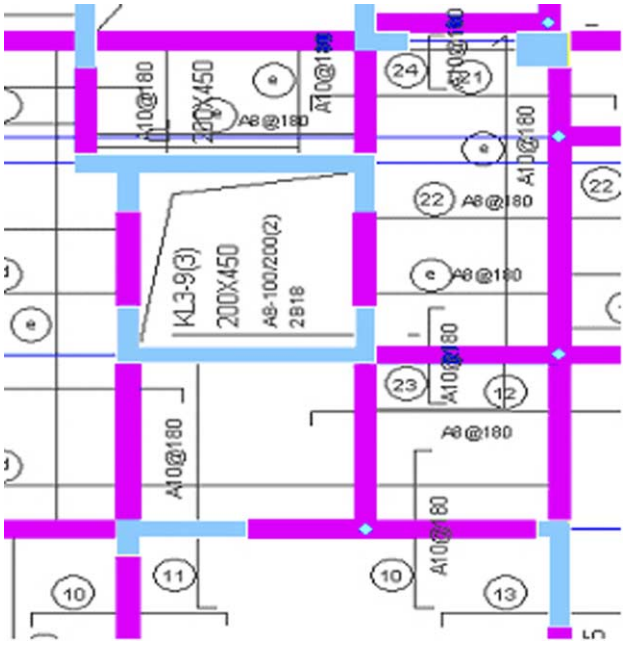

Fig. 15. Recognized nodes and segments.

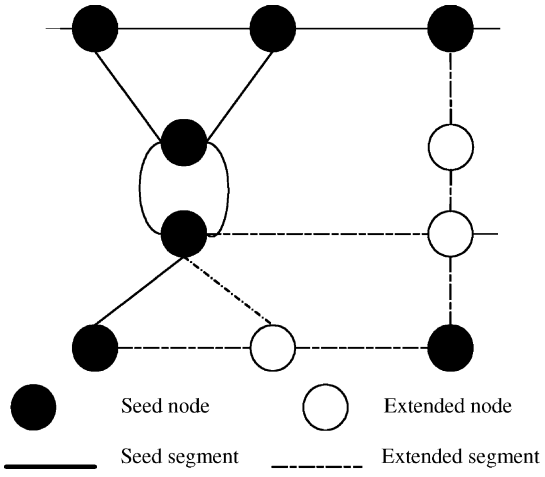

Fig. 16. Component relation graph of Fig. 15.

3.2.3. Recognition of composite components based on component relation graph

Composite components are high-level components that are made up of a group of related nodes and segments. In the slab drawing, a slab is surrounded by columns and beams, which can be extracted as a minimum enclosure surrounded by connected nodes and segments in a component relation graph; that is, there is no other node or segment within a slab's enclosure (Fig. 17). Suppose G_s is the component relation graph of a slab drawing, S is the set of segments and N is the set of nodes in G_s, then for all $s_i, s_j \in S$, $(i \neq j)$, s_i does not intersect s_j (except possibly at their endpoints); that is, $G_s = (S,N)$ is a planar graph and each bounded plane in G_s represents a slab in the corresponding slab drawing. SINEHIR traces simple cycles in the graph to recognize the slabs (Fig. 18). Fig. 19 shows the recognized slabs from a real-life slab CSD, with adjacent slabs shaded in different colors. The slab recognition process no longer depends on the complex graphics details of the primitives.

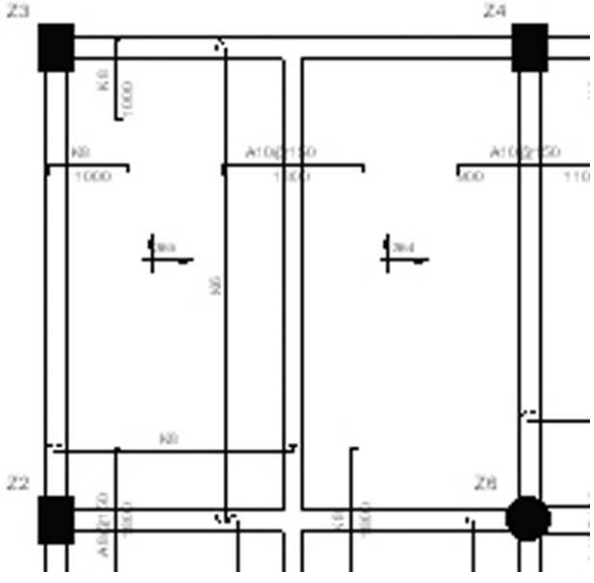

Fig. 17. Two slabs in part of a CSD.

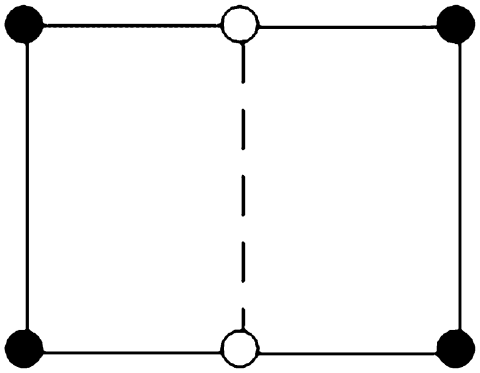

Fig. 18. Component relation graph of Fig. 17.

Walls or beams are composed of consecutive nodes and segments. In CSDs, a wall or a beam usually has a text name (e.g. Q1 and LL1, respectively, in Fig. 20), which is connected to a segment by a line. So after the nodes and segments are recognized from a CSD, part of a wall or a beam can be recognized from their text name and segment. Then, based on the relationships of the segments and nodes in the component relation graph, all the beams and walls can be quickly recognized. Fig. 21 shows the beams and walls recognized from Fig. 20. The bottom left wall named 'Q1' is composed of two connected segments (and three nodes) because the third segment is annotated by another wall text 'Q11'.

3.2.4. 3D reconstruction

After the columns, beams, slabs and walls are recognized, we reconstruct the whole 3D building. We first calculate, for each recognized component, its closed shape in terms of 2D physical coordinates, then recognize and interpret the nearby attribute annotations. Next, from its physical shape and annotated attributes, we create a 3D shape for each component using the global logical coordinate system. Lastly we reconstruct the whole building based on the 3D components. The process is described as follows:

1. Calculate the physical coordinates of the shapes of recognized nodes and segments. Build the rectangular shape of a segment $ss_i\langle l_{i1}, l_{i2}\rangle$ as follows:
   (a) Find the two nodes c_{i1} and c_{i2} connected to ss_i, with centroids (x_{i1}, y_{i1}) and (x_{i2}, y_{i2}), respectively.
   (b) Project (x_{i1}, y_{i1}) to both l_{i1} and l_{i2}, and let their projection points be (x_{p11}, y_{p11}), and (x_{p12}, y_{p12}), respectively. Similarly, project (x_{i2}, y_{i2}) to both l_{i1} and l_{i2} and let their projection points be (x_{p21}, y_{p21}) and (x_{p22}, y_{p22}), respectively. Build a new rectangular shape from the four projection points obtained;
2. Transform the shapes of the nodes and segments into the global logical coordinate system built in Section 3.1;

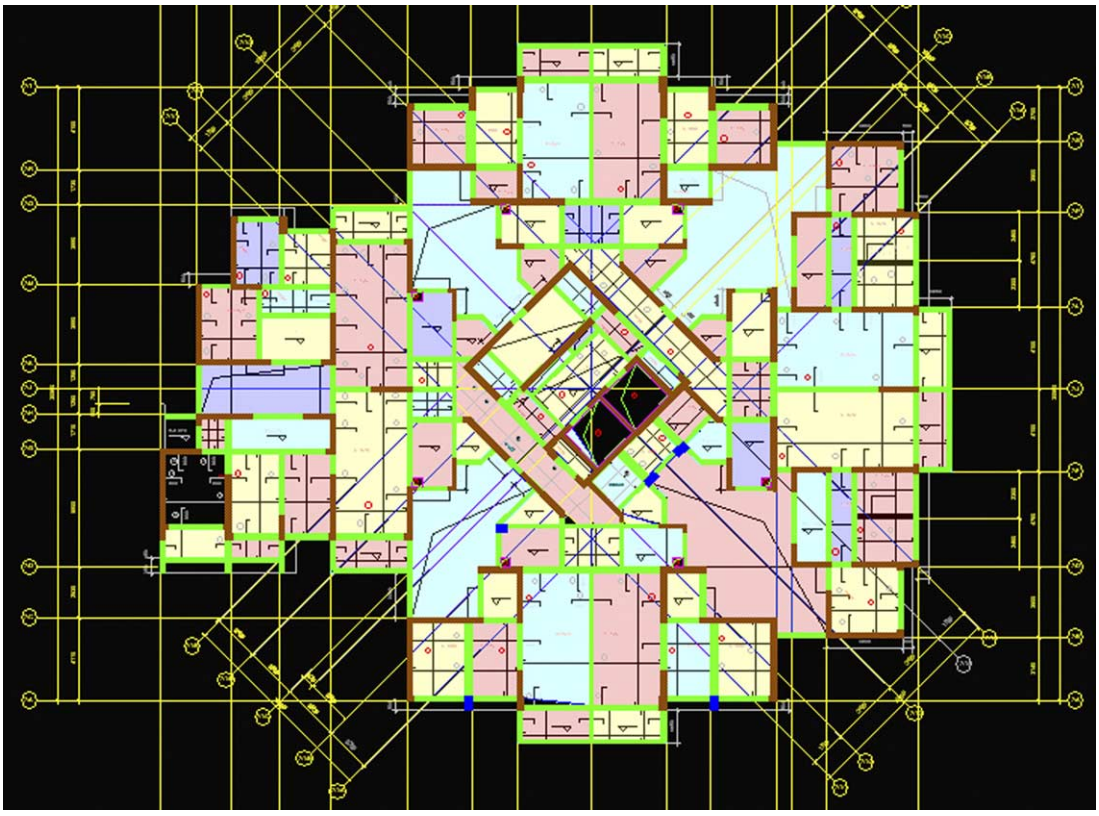

Fig. 19. Recognized slabs from a real-life slab CSD.

3. Recognize and interpret the annotations of columns, beams or walls, which are near to the shapes of the recognized components. These annotations are in fixed formats. For example, the text '300×500' in Fig. 13(a) will be interpreted as a beam of width 300 mm and height 500 mm;
4. Scan all recognized nodes and segments, reconstruct a 3D solid for each component according to the global coordinates of the shape and the 3D attributes of the component;
5. Reconstruct the building based on the component solids.

3.2.5. Calculating internal steel

Our last goal is to recognize the steel bars distributed on or in the recognized components (for example, drawn

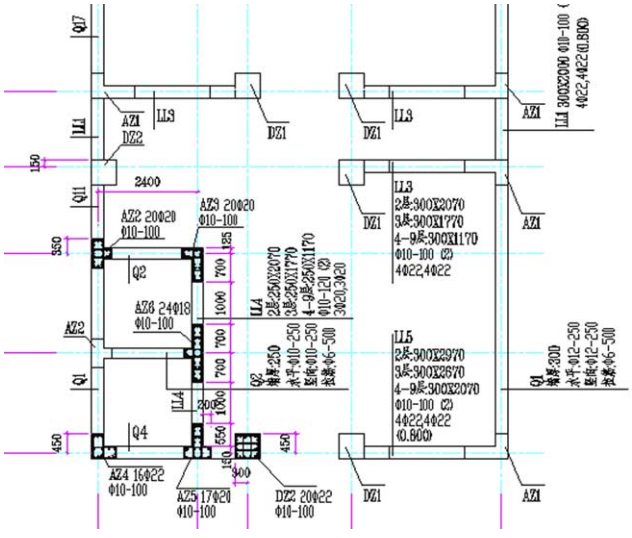

Fig. 20. Part of a CSD.

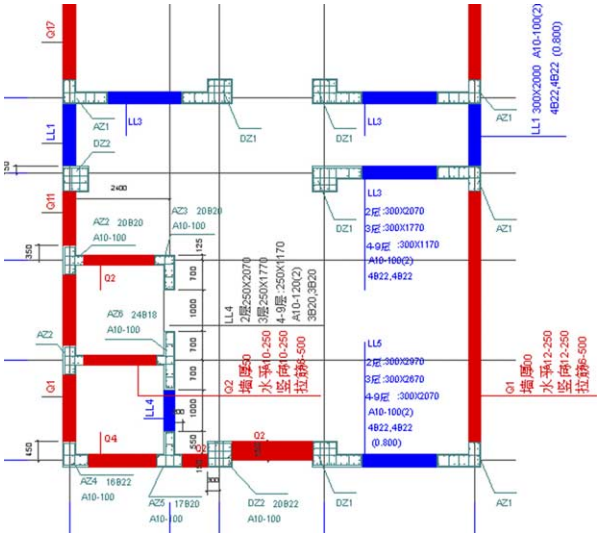

Fig. 21. Beams and walls recognized from Fig. 20.

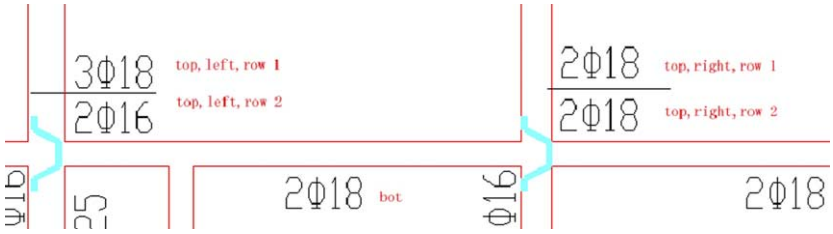

Fig. 22. Different positions of text mean different steel types.

as lines or polylines with hooks at the end on slabs in Fig. 5; the red lines and tiny circles in the sections of columns in Fig. 9 (web version only); or presented as texts likes 2∅18 in Fig. 20). Since by now the number of unused graphic primitives is sharply reduced, the steel bars can be quickly analyzed. The texts of different types of steel bars (e.g. top-left-row1, top-left-row2, top-right-row1, top-right-row2 in Fig. 22) usually appear at different position relative to a beam or wall. Since different types of steel have different formulae for shape calculation, they generate different engineering data and 3D reconstruction. Fig. 23 shows some calculation methods and resulting shapes based on the steel types and the text positions, which are stored in SINEHIR as calculation rules. For example, the length formulae for the top-left-row-1 steel of the beam in red includes: the bending length 15*d*, the length extended into the connected column 0.4Lae, and the length inside the beam Ln/3. So the final length and shape of this kind of steel is shown in Fig. 24(a). Similarly, SINEHIR can interpret the length of the top-left-row-2 steel and the bottom steel as shown in Fig. 24(b) and (c), respectively.

4. Experimental results

We have implemented an architectural drawing recognition and interpretation system using Visual C++6.0. Table 3 gives the experimental results of recognizing CSDs of three different real-life projects, without any user intervention during the recognition process. For a typical CSD project, marked A in Table 3, made up of about 72,000 graphic primitives, the average recognition rate is 93.90% and about 17,000 structural objects are recognized. For project A, our system automatically calculates the engineering data in less than 1 h, and the average accuracy of the calculated engineering data is 98.11%, which completely satisfies practical requirement.

To visualize the reconstructed structure, our system also provides 3D display functionality. Fig. 25 shows the various structural objects reconstructed by SINEHIR. The internal steel of the components is first recognized, and then analyzed from the steel texts or positions, and lastly calculated according to the calculation rules of SINEHIR. Fig. 25(b) shows a column steel that is automatic recognized, analyzed and calculated from a column CSD.

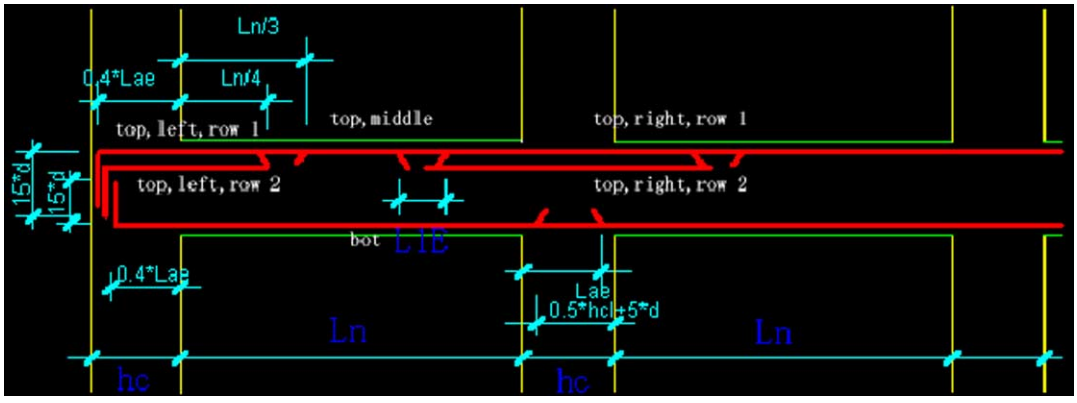

Fig. 23. Steel of different types has different calculation formulae and shapes.

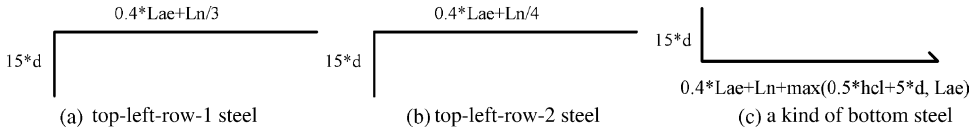

Fig. 24. Final results of lengths and shapes of steel bars.

Table 3
Recognition rate of three real-life projects

Recognition parameters	A[a]	B[b]	C[c]
Graphic primitives	72,1997	35,361	42,327
Dimensions/rate (%)	2268/96.22	1989/97.18	4905/94.62
Seed nodes/rate (%)	765/100	333/99.10	468/97.88
Segments/rate (%)	7897/95.38	2264/94.44	3034/89.49
Composite components/rate (%)	4450/92.40	1396/88.09	1157/82.05
Average recognition rate (%)	96.00	94.70	91.01

[a] A dormitory project in Nanjing.
[b] An office building project in Guangdong.
[c] A government project in Shanghai.

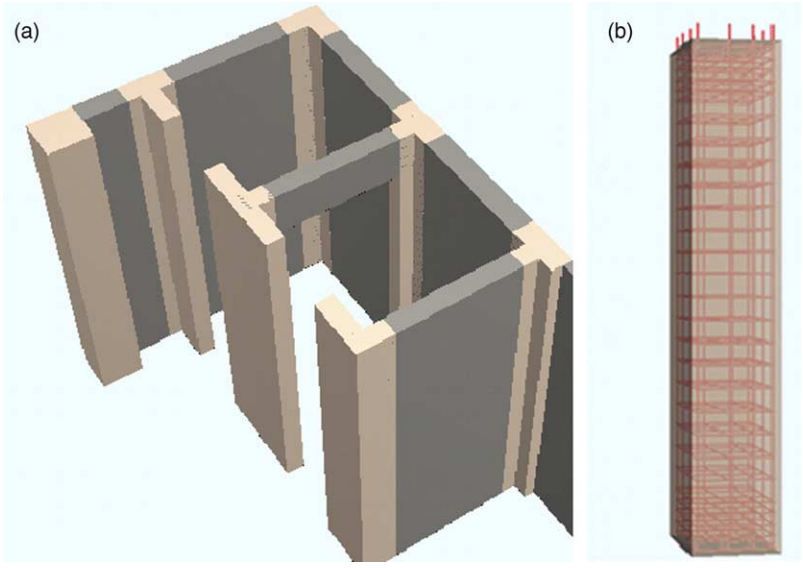

Fig. 25. The reconstructed components of a 3D building. (a) Reconstructed walls and column. (b) A column and its internal steel.

5. Conclusions

In this paper, we discuss the problems encountered in the recognition of electronic construction drawings, and present the Self-Incremental Axis-Net-based Hierarchical Recognition model. Recognition algorithms for the various classes of high-level component objects are presented. Experimental results show that our system can recognize real-life drawings at the average rate of more than 93.90%.

SINEHIR has a few limitations. (1) Drawings that contain graphical sections of structural objects in the form of tables cannot be well recognized, because SINEHIR is a top-down model, which is under the direction of relationships among structural objects. (2) The representation methods of objects in the library are always not enough for real-life drawings; for example, we find that there are at least 11 types of presenting the steel symbol 'Φ': it may be composed of a circle, two half-circles, a letter 'O', or number 'o' with a straight line, or a graphical polylines, etc. We expect to see even more new ways of presenting this symbol in new drawings. (3) Some distortions, incompleteness or minor errors can be automatically resolved, but others, such as the missing of thickness annotation of a slab or the missing of one of the parallel lines of a beam, cannot be automatically resolved. SINEHIR has to warn the users or indicate it in the 3D building.

Further research includes more general symbol recognition, recognition of other kinds of components (such as stairs and groundwork), analysis of other types of architectural drawings (such as decoration drawings), bottom-up methods for structural objects in tables, and 4D modeling.

Acknowledgements

This work was partially supported by a grant from the Research Grant Council of the Hong Kong Special Administrative Region, China (Project No. HKUST6210/02E).

References

[1] Tombre K. Structural and syntactic methods in line drawing analysis: to which extend do they work? In: Advances in structural and syntactical pattern recognition, Leipzig, Germany; 1996, p. 310–21.

[2] Tombre K. Ten years of research in the analysis of graphics documents: achievements and open problems. In: Proceedings of 10th Portuguese conference on pattern recognition, Lisbon, Portugal; 1998, p. 11–7.
[3] Ah-Soon C, Tombre K. Variations on the analysis of architectural drawings. In: Proceedings of fourth international conference on document analysis and recognition, Ulm (Germany); 1997, p. 347–51.
[4] Ah-Soon C. A constraint network for symbol detection in architectural drawings. In: Proceedings of the graphic recognition: algorithms and systems: second international workshop. Berlin: Springer; 1997, p. 80–90.
[5] Koutamanis A, Mitossi V. Automated recognition of architectural drawings. In: Proceedings of 11th international conference on pattern recognition, Den Haag (Netherlands); 1992, p. 660–3.
[6] Nagasamy V, Langrana N. Engineering drawing processing and vectorization system. Comput Vision Graph Image Process 1990;49: 379–97.
[7] Dori D. Syntax enhanced parameter learning for recognition of dimensions in engineering machine drawings. Int J Robot Automat 1990;5(2):59–67.
[8] Dori D, Tombre K. From engineering drawings to 3D CAD models: are we ready now? Comput. Aided Des. 1995;24(4):243–54.
[9] Aelion V, Cagan J, Powers G. Input variable expansion—an algorithmic design generation technique. Res Eng Des 1992;4: 101–13.
[10] Tombre K, Analysis of engineering drawings: state of the art and challenges. In: Proceedings of second international workshop on graphics recognition, Nancy (France); 1997, p. 54–61.
[11] Devaux P, Lysak D, Kasturi R. A complete system for the intelligent interpretation of engineering drawings. J Doc Anal Recogn 1999;2: 120–31.
[12] Dori D, Liu W. Automated CAD conversion with the machine drawing understanding system: concepts, algorithm, and performance. IEEE Trans Syst, Man, Cybern—Part A: Syst Hum 1999;29(4):411–6.
[13] Jensen C, Helsel JD. Engineering drawings and design. New York: McGraw-Hill; 1990.
[14] Song J, Su F, Tai C-L, Cai S. An object-oriented progressive-simplification-based vectorization system for engineering drawing: model, algorithm, and performance. IEEE Trans Pattern Anal Mach Intell 2002;24(8):1048–60.
[15] Xi X, Dou W, Lu T, Cai S. Research on automated recognizing and interpreting architectural drawings. Proceedings of the first international conference on machine learning and cybernetics (IEEE), Beijing, China 2002;1000–4.
[16] Su F, Song J, Tai C-L, Cai S. Dimension recognition and geometry reconstruction in vectorization of engineering drawings. IEEE conference on computer vision and pattern recognition, Hawaii; Dec 11–13, 2001, p. 710–6.
[17] Das AK, Langrana NA. Recognition and integration of dimension sets in vectorized engineering drawings. Comput Vision Image Understand 1997;68(1):90–108.
[18] Lin SC, Ting CK. A new approach for detecting of dimensions set in mechanical drawings. Pattern Recogn Lett 1997;18(4):367–73.
[19] Collin S, Vaxiviere P. Recognition and use of dimensioning in digitized industrial drawings. In: First international conference on document analysis and recognition, St Malo, France; 1991, p. 161–9.
[20] Lai CP, Kasturi R. Detection of dimension sets in engineering drawings. IEEE Trans, Pattern Anal Mach Intell 1994;16(8):848–55.
[21] Ablameyko S, Bereishik V, Gorelik A, Medvedev S. Reconstruction of 3D object models from vectorised engineering drawings. Pattern Anal Appl 2002;5:2–14.
[22] Agarwal M, Cagan J. On the use of shape grammars as expert systems for geometry based engineering design. Artif Intell Eng Des, Anal Manuf 2000;14:431–9.
[23] McCormack J, Cagan J. Enabling the use of shape grammars: shape grammar interpretation through general shape recognition. In: Proceedings of the 2000 ASME design engineering technical conferences: design theory and methodology conference, ASME, New York; 2000 [DET2000/DTM-14555].
[24] Chhabra AK. Graphic symbol recognition: an overview. In: Proceedings of second international workshop on graphics recognition, Nancy (France); 1997, p. 244–52.

Tong Lu received the BS degree in computer science from Nanjing University in 2002. Currently, he is a PhD candidate in the Department of Computer Science and Technology at Nanjing University. His research interests include graphics recognition, automatic interpretation of engineering drawings, and image processing.

Chiew-Lan Tai is an Associate Professor at the Department of Computer Science, Hong Kong University of Science and Technology. She received her BSc in Mathematics from the University of Malaya, her MSc in Computer and Information Sciences from the National University of Singapore, and her DSc in Information Science from the University of Tokyo. Her research interests include geometric modeling, computer graphics, and graphics recognition.

Feng Su received the BS degree and the PhD degree in computer science from Nanjing University in 1997 and 2002, respectively. His research interests include computer graphics, image processing, and pattern recognition. He is a member of the IEEE and the IEEE Computer Society.

Shijie Cai graduated from the Department of Mathematics at Nanjing University in 1967. He is a Professor in the Department of Computer Science and Technology at Nanjing University. His research interests include computer graphics, graphics recognition, and document analysis and recognition.