A Generic System for Form Dropout

Bin Yu and Ani K. Jain

Abstract—Recent advances in intelligent character recognition are enabling us to address many challenging problems in document image analysis. One of them is intelligent form analysis. This paper describes a generic system for form dropout when the filled-in characters or symbols are either touching or crossing the form frames. We propose a method to separate these characters from form frames whose locations are unknown. Since some of the character strokes are either touching or crossing the form frames, we need to address the following three issues: 1) localization of form frames; 2) separation of characters and form frames; and 3) reconstruction of broken strokes introduced during separation. The form frame is automatically located by finding long straight lines based on the block adjacency graph. Form frame separation and character reconstruction are implemented by means of this graph. The proposed system includes form structure learning and form dropout. First, a form structure-based template is automatically generated from a blank form which includes form frames, preprinted data areas and skew angle. With this form template, our system can then extract both handwritten and machine-typed filled-in data. Experimental results on three different types of forms show the performance of our system. Further, the proposed method is robust to noise and skew that is introduced during scanning.

Index Terms—Form processing, learning form structure, document image analysis, segmentation, character reconstruction, block adjacency graph.

1 INTRODUCTION

Form processing is an essential operation in many business and governmental organizations. Forms are special documents typically used to collect or distribute data (e.g., credit card applications, income tax returns, admission forms and business reply mail (BRM) cards). A typical filled-in form consists of the following three components:

1) form frames, including black lines and blocks;
2) preprinted data such as logos, large symbols and machine preprinted characters; and
3) user filled-in data (including machine-typed and/or handwritten characters and some check marks) which are located in predefined areas, called filled-in data areas usually bounded by form frames and preprinted texts.

We regard the first two components as preprinted entities. The useful data to be extracted from the forms are contained in the filled-in text. Distinguishing the filled-in data from the preprinted entities is one of the fundamental problems in form processing. Manually keying in the filled-in data is a time consuming and expensive work because the number of forms to be processed is very large. Interactive tools which provide a graphic interface for manually extracting filled-in data [1] do not solve this problem completely. Machine understanding of form documents is an important problem in the advancement of office automation. Advances in handwritten character recognition have facilitated this problem to some extent. In spite of the existence of some forms which are printed with dropout ink, a majority of the forms are printed without dropout ink for the sake of cost and convenience. The major task in form dropout is to separate and remove preprinted entities while preserving the user filled-in data by means of image processing techniques. This is a difficult task, especially when the filled-in characters are handwritten or typed outside the data areas and touch or cross form frames or preprinted texts.

We call this situation field overlap (between preprinted domain and filled-in domain).

The literature on form processing can be categorized into two areas. Most of the effort is concentrated in form structure extraction and analysis [2] based on blank forms. The second main research area is the filled-in data extraction or form dropout. A number of previous studies can perform the extraction of filled-in text without any attention to field overlap [3]. Our goal is to extract filled-in data from a form in the presence of field overlap. Since the strokes of filled-in characters are either attached to or located across the form frames, the problem of character segmentation involves two issues:

1) separating characters from form frames and
2) reconstruction of broken strokes introduced during separation.

Because the pixels located in the areas where a character stroke crosses a form frame represent both a part of the stroke and a part of the frame, the stroke will be broken after separating the form frame. Therefore, we need to reconstruct the broken strokes. Maderlechner [4] introduced a method which separates filled-in characters from attached form frames and reconstructs broken strokes based on the skeleton. This method is time consuming and prone to noise. The system presented by Casey et al. [5] separates touching characters and reconstructs broken strokes by using a line tracking technique. The form structure is created by an interactive operation. Doerrmann and Rosenfeld [6] solved this problem by using cross section computation, which is also a time consuming algorithm. Tang et al. [7] proposed a system which can extract desired information from scanned financial documents (e.g., bank checks and drafts) by using a form description language.

We propose a system which can automatically capture form structure from a blank form and extract filled-in data from filled-in forms. Most forms are characterized by the presence of horizontal and vertical frames that delimit the usable space. A form frame can be a simple straight line or a block area. In practice, filled-in characters often touch and cross the horizontal frames. Our system focuses on the horizontal frames which are the most obvious characteristics in a form. We utilize block adjacency graph (BAG) [8] to locate the horizontal frames even when there is a modest amount of skew introduced during scanning. Fig. 1 shows a block diagram of our form dropout system. In Section 2, we introduce the BAG data structure and some of its properties. The details of form frame separation and character reconstruction are given in Section 3. We discuss form template, registration and preprinted data dropout in Section 4. Finally, the performance of the system is shown in Section 5 with some examples.

2 BLOCK ADJACENCY GRAPH

The key information that we utilize in form frame extraction is the location of horizontal frames. A horizontal form frame consists of more than one “long” horizontal run length in the digital image when the skew angle is small (<5°). Therefore, the image representation should be chosen to facilitate extraction of horizontal frames. On the other hand, a typical A4 size document will produce a digital file of up to 1 MB after scanning it in a binary mode at 300 dpi. Therefore, the scanned image should be compressed and filtered.
which is defined as set of block nodes and 

cating the connection between nodes

each node 

lengths adjacently connected and aligned on two sides with a given

given in Fig. 2. Fig. 3b illustrates the BAG representation of the 
tolerance and is characterized by its rectangular boundary coordi-
nates (x1, x2, y1, y2) where

The algorithm for creating BAG from raster image is

raster image shown in Fig. 3a. Let

be the width and height of the block node

However, BAG is a more efficient data structure for binary image 
processing since

1) it has an attractive compression rate (typically from 100:1 to
10:1 for binary images);
2) the BAG creation algorithm is a one-pass algorithm and can
be completed during scanning;
3) it preserves both shape and structure information; and
4) most traditional operations on binary images, such as con-
tour extraction, vectorization and connected component ex-
traction can be efficiently implemented based on it.

Fig. 4a shows a small part of a form image used to illustrate the 
output of various stages in our system. Fig. 4b shows its BAG 
representation.

On a BAG representation G, a connected subgraph G0(N0, E0) is a 
BAG which satisfies the following properties:

1) G0 ⊆ G and N0 ⊆ N;
2) E0 ⊆ E and ∀ni, nj ∈ N0, there is a path (ni, ni1, ..., nj, ..., njk, nj) 
such that ui, ∈ Nk for l = 1, 2, ..., p and

The rectangular boundary coordinates of G0(N0, E0) are

Each run length in the first row of the input image is regarded as a 
block.

For the successive rows in the image |

For each run length ri in the current row |

If ri is 8-connected to a run length in the preceding row |

If ri is 8-connected to only one run length rj which has not 
been connected to any other run length in the current row 
and the differences of the horizontal positions of their 
beginning and end pixels are, respectively, within a given 
tolerance, then ri is merged into the block node nj involving rj.

Else, ri is regarded as a new block node nj initialized 
with edges e(nj, ni) to those block nodes |nj which are 
8-connected to rj.

Else, ri is regarded as a new block node.

Fig. 2. Algorithm for generating BAG data structure.

Fig. 3. Horizontal form frame separation and stroke reconstruction: (a) 
raster image; (b) BAG representation; (c) horizontal form frame separa-
tion; (d) stroke reconstruction by interpolation.

where

Wk = (Xk - Xk') and Hk = (Yk - Yk') denote the width and height of 
the subgraph Gk, respectively. On a connected subgraph Gk, we define 
its surrounding edge set Ek = {e(ni, nj) | ni, nj ∈ Nk \ E0}, 
and surrounding node set Nk = {ni | ∃e(ni, nj) ∈ Ek} where \ is the 
subtraction operation between two graphs. If Ek = ∅ then Gk is 
called a complete connected subgraph representing a connected com-
ponent. We regard form frames and objects in the image as con-
ected subgraphs. Furthermore, a form image can be defined as 
\(J, C, a, \phi\), where a is its skew angle, C = {Jc, Ja, Ol}, and Jc, Ja, 
and Ol are sets of horizontal form frames, vertical form frames and 
other connected components, respectively. Our major task is to 
locate and separate these three components.
We define the horizontal and vertical distances between two nodes \( n_i \) and \( n_j \) as:

\[
\delta_i(n_i, n_j) = \begin{cases} 0, & \text{if they are horizontally overlapped,} \\ \min|y_i - y_j|, & \text{otherwise,} \end{cases}
\]

and

\[
\delta_j(n_i, n_j) = \begin{cases} 0, & \text{if they are vertically overlapped,} \\ \min|y_i - y_j|, & \text{otherwise,} \end{cases}
\]

Similarly, the horizontal and vertical distances between two connected subgraphs is defined as:

\[
\Delta_i(G_i, G_j) = \begin{cases} 0, & \text{if they are horizontally overlapped,} \\ \min|X_i - X_j|, & \text{otherwise,} \end{cases}
\]

and

\[
\Delta_j(G_i, G_j) = \begin{cases} 0, & \text{if they are vertically overlapped,} \\ \min|Y_i - Y_j|, & \text{otherwise.} \end{cases}
\]

In terms of connected components, an image can be decomposed as \( G = \bigcup_i G_i \), where \( G_i \subset G \) is a connected component and \( G_i \cap G_j = \emptyset, \forall i \neq j \), where \( \bigcup \) and \( \emptyset \) are union and intersection operations between two graphs. To reduce the computational burden, small connected components \( G_i(N_i, E_i) \) with \( |W_i|, |H_i| < T_{cc} \) could be masked in the BAG. We chose \( T_{cc} = 0.4" \).

### 3.1 Locating and Separating Horizontal Form Frames

Based on BAG, finding a long horizontal frame is equivalent to finding a connected subgraph \( G_i(N_i, E_i) \) of \( G(N, E) \) such that:

1. \( w_i > T_{hor} \) and \( h_i < T_{hor} \), \( \forall \), \( i \in N_i \);
2. the length \( W_i > T_W \); and
3. the ratio \( H_i / W_i < T_{hor} \).

We have used following threshold values: \( T_{hor} = 0.1" \), \( T_W = 0.013" \), \( T_h = 0.33" \), and \( T_{hor} = 0.2 \). Fig. 3b shows how a horizontal line (shown as shaded blocks) is located using this method. Most horizontal form frames can be located and separated by this method as shown in Fig. 3c. When separating the horizontal frame \( G_i \), we add edges \( e(n_i, n_j) \) into \( E_i \) if \( \exists \{n_i, n_j\} \in E \) or \( \exists \{n_j, n_i\} \in E \), and the distance of the ends of two stroke segments \( \delta_i(n_i, n_j) < T_{hor} \). The broken stroke segments sometimes do not overlap horizontally because of the tilt of the handwritten stroke and the skew, e.g., the rightmost character “c” in Fig. 5e. Threshold \( T_{hor} \) should be larger than 0; we have set \( T_{hor} = 0.007" \) in our experiments. This will record connectivity information about crossing strokes and vertical frames connecting the horizontal frame (see Fig. 3c). Let \( F_i = \{G_i\} \) denote the set of horizontal form frames separated from \( G \). Without the use of a specific skew detection algorithm [10], the skew angle \( \alpha \) of the image is estimated using the orientation of horizontal lines.

\[
\alpha = \frac{\sum_{G_i \in F} w_i \arctan \left( \frac{y_i^+ + y_i^-}{x_i^+ - x_i^-} \right)}{\sum_{G_i \in F} w_i^2}
\]

where \( n_i \) is chosen such that \( x_i^+ = \min _{n_i \in G_i} \{x_i\} \) and \( n_i \) is chosen such that \( x_i^- = \max _{n_i \in G_i} \{x_i\} \).

### 3 Form Frame Separation and Stroke Reconstruction

To determine the overlap between the form frames and filled-in characters, we deal with horizontal frames first. They are located and separated in the BAG. At the same time, connections between frames and character strokes are identified and the strokes broken during frame separation are reconstructed. After this process, some vertical form frame segments originally separated by those horizontal frames will become connected. It is now easier to locate these long vertical lines. This procedure leads to a decomposition of BAG from \( G \) to \( F(C, \alpha) \).

Fig. 4. A part of a form image: (a) raster image; (b) BAG representation; (c) horizontal frame separation; (d) form frame separation and character reconstruction.

Fig. 5. Examples of character reconstruction. First column shows segments of the input image, second column shows the result after the form frame separation, and the last column shows the reconstructed characters.
3.2 Stroke Reconstruction and Vertical Frame

Continuation

In the areas where the characters touch or cross form frames, they share pixels. During the form frame separation, some of the strokes of these touching or crossing characters will be broken, which is likely to increase the error rate of the character recognition module. After separating the form frames, we perform character reconstruction to restore the broken strokes. This reconstruction process is independent of the character recognition module. On the other hand, we regard a vertical form frame as a vertical segment between two horizontal form frames. To correctly locate vertical frames, we should link these segments when separating horizontal frames.

Let \( n_i \in N \) be the node connecting a horizontal frame \( G \) with a stroke segment \( S \). If \( \exists n_i \in N \) and \( e(n_i, n_j) \in E \), we say that a stroke crosses \( G \) and is broken into two segments \( S \) and \( S \), which connects \( G \) with the node \( n_i \). Otherwise, a stroke touches \( G \), such that none of its broken segments appear on both sides of the frame. If the width of the touching section between a stroke and a horizontal frame, \( W < T_w \), then we do not have to reconstruct this stroke. Otherwise, it is extended by a distance \( 0.5w \), while the width of the extended stroke smoothly converges to \( 0.5w \). A few examples of this reconstruction are given in Fig. 5. Some touching characters are difficult to restore without first applying a recognition module. For example, an upper case “U” touching a frame at its bottom area will have two vertical strokes left after separating the frame. A similar problem will be encountered for the upper case “D” in Fig. 4.

The most difficult reconstruction is encountered when the broken strokes are caused by a stroke crossing the frame. We categorize these broken strokes into two classes. The first category is called “one-to-one” where the stroke components in a pair match each other across the frame. The character “y” in Fig. 5b belongs to this class. The other type of broken strokes is called “multiple-to-one” or “one-to-multiple” where one component on one side of the crossed frame matches multiple entities on the other side. The character “W” in Fig. 5g is an instance of this type of crossing.

For the first category of crossing, we interpolate the blocks between \( n_i \) and \( n_j \) (see Fig. 3d). In the second case (see, for example, the character “W” in Fig. 5h) one segment \( S \) under the frame is related to two others \( S \) and \( S \) on the frame, i.e., \( d(n_i, n_j) \neq 0 \). We restore the strokes by interpolating extra blocks to fill in the broken area. Vertical frames are processed as the first class of broken strokes. This allows us to link several vertical segments into a single longer vertical segment.

3.3 Locating Vertical Form Frames

Similar to stroke reconstruction, most vertical form frame segments are joined into longer ones which are easier to identify. Some short vertical frames are difficult to distinguish from the long vertical strokes of characters such as numeral “1.” For a vertical frame, the angle against a vertical reference line \( \alpha \) should be close to the skew angle \( \theta \) detected during the process of locating horizontal frames. A connected subgraph \( G(N, E) \) is a vertical frame if

1) \( \forall n_i \in N, d_i < 1, d_i \leq 1, \) and \( w < T_w \), and
2) \( T_w > T_w' > T_w > T_w > 1 > v_w \)

We have used following threshold values: \( T_w \approx 0.04 \), \( T_w \approx 0.13 \), and \( \alpha = \alpha < v_w \).

We now have a form representation \( H \) and \( V \) separated the horizontal form frames \( H \) and vertical form frames \( V \) from the form image \( G \) and reconstructed the broken characters by interpolating some nodes in \( G \). We now have a form representation \( H \) and \( V \) has been updated into \( G \). Let \( G \) be a complete connected subgraph. We determine the last element in \( C \) as \( G \). In the following stages, we use subscripts 1 and 2 to denote form template and input filled-in form, respectively.

4 Filled-In Data Extraction

To extract filled-in data, the system should have the form structure knowledge in advance. This knowledge can be acquired by one of the following methods:

1) a form structure file created by a word processing software,
2) interactive learning [5], or
3) scanning and automatic recognition of a blank form.

For many applications, it is difficult to get a suitable form structure file which provides the needed geometric information in a readable format by a word processing software. On the other hand, interactive learning is time consuming and tedious. Therefore, an automatic procedure for capturing form structure is very desirable. This information can be obtained by simply processing a blank form with our system. First, the system needs to capture the structure of a specified form and record it as a template. This template is used for registering an input filled-in form and guiding the identification of preprinted data.

In the previous stages, we have determined the skew angle \( \alpha \) separated the horizontal form frames \( H \) and vertical form frames \( V \) of the form image \( G \) and reconstructed the broken characters by interpolating some nodes in \( G \). We now have a form representation \( H \) and \( V \) has been updated into \( G \). Let \( G \) be a complete connected subgraph. We determine the last element in \( C \) as \( G \). In the following stages, we use subscripts 1 and 2 to denote form template and input filled-in form, respectively.

4.1 Form Template

We have noted that even preprinted objects often touch form frames in the digitized image. In such cases, the techniques mentioned above can work by processing a blank form image in order to automatically capture the form structure. First, we recursively group or cluster a component \( G \) into a subgraph \( G \) if \( G \) is connected to \( G \) or \( G \) is isolated. Note that \( G \) will involve some connected components which are close to each other horizontally. The form template is denoted as \( J(C, \alpha) \), where \( C = \{ H, V \} \) are the sets of horizontal form frames, vertical form frames and grouped preprinted objects, respectively, and \( \alpha \) is the skew angle of the blank form image used for form template learning. Fig. 6a shows an image of a blank BRM card and the captured form template is shown in Fig. 6b in terms of the derived bounding boxes of subgraphs in \( C \).

Fig. 6. Form structure capture. (a) a blank business reply mail card; (b) form template which has been automatically captured.

4.2 Form Registration

Form registration is the process of calibrating an input form according to the form template. The commonly used correlation criterion minimizes the translational and rotational differences between the form template and the input form.

After locating and separating form frames in a filled form, the form has been decomposed as \( J(C, \alpha) \), where \( C = \{ \} \) is the form template and \( \alpha \) is the input form.
All form frames and the centers \( \left( \frac{x_0 + x_1}{2}, \frac{y_0 + y_1}{2} \right) \) of the components in the input form image are rotated in a clockwise direction by an angle \( (a_2 - a_1) \) to align the template and the input form in the same direction. We use horizontal and vertical frames to get the translation values \( (t_x, t_y) \) in horizontal and vertical directions, respectively. Let \( Y^2 = (Y_0^2 + Y_1^2) / 2 \) be the vertical position of a horizontal frame. We first increment the accumulator array \( V(Y - Y') \) by \( \min(W_0, W_1, Y_0 \in H_0, Y_1 \in H_1) \) if \( (Y - Y') \) is within the interval \([-1", 1"]\). The parameter \( t_y \) is determined such that \( V(t_y) \) is the maximum, where \( V' \) is obtained from smoothing \( V \). The algorithm for estimating \( t_y \) is analogous. The two translation parameters \( (t_x, t_y) \) are used to calibrate the input image according to the template. If no vertical form frames are available, then we obtain \( t_y \) based on the average distance between the end points of long horizontal frames which are matched to each other after vertical translation calibration.

4.3 Identifying Preprinted Data

When mapping to the template space, an object \( G_i \in O_i \) in an input form image is called preprinted, if \( \exists G_j \in C_j \), the overlap between their bounding boxes \( (X_0^j, X_1^j, Y_0^j, Y_1^j) \) and \( (X_0^i, X_1^i, Y_0^i, Y_1^i) \) is over 50% of the area \( A_i \) of \( G_i \), where \( A_i = W_iH_i \). These objects which include preprinted entities and some remainders of form frames are deleted and the remaining objects in the image are assumed to be filled-in data and/or possibly noise. The filled-in data are fed to either a postprocessing or a data compression module.

5 Experimental Results

Our system has been tested on more than 60 forms of 11 different types filled-in either by hand (cursive and printed) or machine typed. They were scanned with a small amount of skew (<5°) at 300 dpi. This resolution is required by most OCR software packages. These images were input to our form dropout system coded in C language and running on a SPARC 20 workstation. The typical image sizes for the forms shown in Figs. 7 and 8 are \( 2,550 \times 3,300 \). The same threshold values were used for all the forms. It takes approximately 10 s to extract filled-in data. The images in Fig. 9 are of size \( 1,650 \times 1,050 \), and it takes about 2 s to process them.

The forms shown in Fig. 7 were photocopied (in black and white) from a green ink-printed form and were filled in by 12 different people. Seven of the subjects filled out the form with a ball-point pen, three with an ink pen, one with a pencil, and another by a typewriter (Fig. 7c). Fig. 7 shows two of these forms. Note that in the top-left area of the form there is a shaded square bounded by a dashed rectangle which contains a line of preprinted text crossing a form line. The form dropout results for Figs. 7a and 7c are shown in Figs. 7b and 7d, respectively. Fig. 8 shows form dropout as well as OCR [11] results on the extracted filled-in data. All the BRM cards were filled in by 11 different people. Seven of them used a ball-point pen, two used a pencil, and the other two used an ink pen. Two of these forms are shown in Figs. 9a and 9c. The corresponding form dropout results are shown in Figs. 9b and 9d. In addition to the filled-in characters, some check marks are also extracted correctly. In these sample forms, the handwritten characters touch the form frames in all the four directions (top, bottom, left, and right).

6 Conclusions

We have developed a form dropout system for automatic form processing. Our system has the following properties:

1) it can process a variety of forms and business reply mail cards that are either filled-in by hand or typed;
2) it is able to accommodate a moderate amount of skew in the input images and no special skew detection and deskew processes are needed;
3) the system can locate and separate both horizontal and vertical form frames without using any particular knowledge of the form structure even when there are characters touching and crossing the frame boundaries;
4) it can correctly reconstruct most handwritten or machine-typed character strokes that are broken by the form frame separation;
5) by processing a blank form, the system can automatically pick out the form structure which is then stored as a form template;
6) based on the captured form template, the system can extract filled-in data for a class of input form images.

We have shown that the data structure, called block adjacency graph (BAG), is quite efficient for the purpose of representation and other processing stages.

In practice, besides form frames, some of the filled-in characters may also touch preprinted text or objects. This is a more difficult problem to handle. Currently, we deal with it on a case by case basis. For example, if the filled-in data item is a check mark which overlaps a preprinted box, then our system learns the positions of these boxes and saves the objects both inside and in the neighborhood of these areas. An example is given in Fig. 9d. In more complicated cases, the system will capture the filled-in data, but will also extract some preprinted text at the same time. Some high-level or knowledge-based postprocessing needs to be employed to resolve these ambiguities. On the other hand, the proposed system treats dashed lines in a form as preprinted objects. The separation of filled-in data and dashed lines is as difficult as the problem of separating filled-in data and preprinted objects.

Acknowledgments

We would like to thank Dr. Jianchang Mao and Dr. Moizdin Mohiuddin of IBM Almaden Research Center for their support of this work.
Fig. 7. Form dropout experiments: (a) a form filled in by hand; (b) dropout results for (a); (c) a form that has been typed; (d) dropout results for (c).
Fig. 8. Form processing: (a) a form filled in by hand; (b) form dropout; (c) OCR results on filled-in data.
REFERENCES


