Logical Structure Representation Of Form Documents Based on Line Information [1]

Pınar Duygulu *, Volkan Atalay * , Ebru Dincel *+
* Dep.of Computer Eng., Middle East Technical University, Ankara, TR-06531 Turkey
+ TÜBİTAK-BİLTEN, Middle East Technical University, Ankara, TR-06531 Turkey
e-mail : {duygulu,volkan,dincel}@ceng.metu.edu.tr

1 Introduction

A form processing system aims to extract meaningful data from a form document for office automation [1,2,3]. A form is a structured document which is generally composed of horizontal and vertical lines, preprinted data (machine printed characters, symbols and pictures) and user filled-in data which are located at predefined locations. In order to extract the user filled-in data, the form structure should be known in advance. The form structure which consists of horizontal and vertical lines and preprinted data, can be obtained by processing a blank model form on which no user filled-in data exists. A model form can then be used to specify predefined locations of individual data items which are considered to be meaningful. This model form may be stored in a database in order to register eventually a filled-in instance form. When an instance form is presented to the system, its type can be identified by matching with one of the model forms in the database and meaningful data can then be automatically extracted.

The most straightforward way to store a form document is the direct representation by its image. But, such a physical information may not be appropriate, since the forms or their images risk to be modified geometrically (enlarged/shrinked, translated/rotated, etc.) or to be distorted due to printing or digitization. On the other hand, logical structure represents the semantics of the form, and although we are mostly concerned in the reverse process, same logical structure

[1] This work is partially supported by TÜBİTAK-BİLTEN (Scientific and Technical Research Council of Turkey - Institute of Information Technologies and Electronics) under the project 96-14-030.

can be formatted in a variety of physical layouts. The geometrical structure should be mapped to a logical structure by considering the logical relations. Even the multi-kinds of the same application specific forms (logically same but physically different forms) can then be identified by considering the logical structure.

In this study, our aim is to develop a logical representation for form documents. A heuristic algorithm is presented to transform geometric structure of a form into a logical structure by using horizontal and vertical lines which exist on the form. The logical structure is represented by a hierarchical tree. The hierarchy of the tree corresponds to the hierarchy of the blocks in the form document. The proposed representation is close to human point of view for the form structure. Logically same forms have the same hierarchical tree structure. Also, geometrical modifications and slight variations on a form are handled by the proposed representation.

2 Related Work

Several approaches are proposed for form recognition and identification [4,5,6,7,8]. Physical features such as length, width and position of horizontal and vertical lines can be used to identify the forms [4]. Although such an approach may solve the problems related to skew, scaling and translation, variations on the physical structure of the logically same form cannot be handled. For identification, another physical feature is the line crossings that appear on a form. Taylor et. al. [5] use the count of various types of line crossings on a form in order to classify them. However, this approach will not work even for slight variations, because the types and counts of line crossings are prone to change. Another approach at a more logical level is to use blocks and their relationships [6,7,8]. In such an approach, use of physical information such as the location and size of blocks is still possible [6]. But, multi-kinds of forms cannot be handled by such a scheme. In a similar way, relations of the blocks which share lines can be employed [7]. However, in such a scheme, hierarchical relationships of block groups are not represented. A more general method is to use a bottom-up approach to form blocks. In the work of Watanabe et. al.[8], three binary trees are constructed. Two of them are for global and local structure. The last one is for classification whose construction is performed through a block division process which is not given in detail in the paper. In addition to the horizontal and vertical lines, some of the preprinted data is also used. In the literature, the method is found to be hard to apply to the analysis of filled-in forms, because it is considered to be limited to empty fields [6].

3 Definitions and Approach

The physical structure of a form consists of horizontal and vertical lines in addition to the preprinted and filled-in data. We propose a hierarchical structure to represent the logical layout of a form by using lines. First, some definitions will be given and then we will resume the approach.

A *block* is a rectangular area on the form which is surrounded by the longest horizontal and vertical lines at any given instant. For example, the biggest block of a form is the form itself. A *cell* is defined as the smallest block which only consists of a block frame. A *block frame* is the horizontal and vertical lines that surround a block and the lines which have the same length as the borders of the block frame are defined as the *baselines. Orthogonal lines* are the lines which are orthogonal to the baselines, that start at any baseline and end at another baseline. For the overlapping orthogonal lines, the one with the maximum length is taken as the orthogonal line. A block frame is defined to be *ambiguous*, if it includes only the lines of length equal to the borders of the block frame.

The main aim is to partition the form into blocks which can further be divided until cells are reached. The partitioning results in a tree where the root is the form itself and the leaf nodes correspond to the cells. The heuristic behind the approach is that the block which contains similar information are separated by lines which are relatively longer than the others. Such lines by definition are called as baselines. Thus, the information about how to partition a block is given by the baselines. However, not all baselines provide this information, but the orthogonal lines give clues about which baselines are separators.

In order to achieve a sequence of block partitioning, a switching of horizontal and vertical divisions is proposed. This exploits all of the information inherited by the logical structure of the form via horizontal and vertical lines. At a step, since only the information supplied by a certain type of lines is exploited, for further partitioning at the next step, the information provided by the other types of lines is used.

Using the given definitions, the proposed approach is implemented by an algorithm to extract a hierarchical representation of a given form document. The approach is top-down and no domain knowledge such as the preprinted data or filled-in data is used. At the end, two forms which have different physical structures but the same logical structure will be associated to the same representation.

4 Heuristic Algorithm

The heuristic algoritm to implement the above approach is given as follows.

- Phase I : Initialization.

  1. Initialize the current block frame to the form frame.
  2. Goto Phase II.

- Phase II : Block finding.

  1. Find the baselines on the current block frame.
  2. Initialize the current baseline to the first line of the current block.
  3. Define the block by searching the orthogonal lines for the current baseline.
  4. Continue with 3 until there are no other remaining baselines in the current block.
  5. Goto Phase III.

- Phase III : Hierarchical tree construction.
  For each defined block

  1. Insert a node corresponding to the defined block into the hierarchical tree.
  2. If the block is a cell, then stop,
  3. Else assign the block to the current block and Goto Phase II.

The algorithm consists of three phases : initialization, block finding and hierarchical tree construction. In fact, the whole algorithm is recursive in the sense that the last phase may refer to the second phase and itself during the process. An initial partitioning of the block is done at the second phase. The partitioned blocks are stored to be inserted into the tree and then to be further divided into smaller blocks, if possible. The partitioning is always performed at the second phase whose third step is used to define a block. Defining a block by using baselines and orthogonal lines is not a very straightforward process. We will start with a basic algorithm and then refine it by considering different cases. A block is always defined between the current baseline and the ending baseline. A basic algorithm for defining a block is given as follows.

A. If the orthogonal lines are only the borders of the frame, then

   1. Define the next baseline as the ending baseline.
   2. Define a block as between the current baseline and the ending baseline
   3. Assign the ending baseline to the current baseline.

B. Else

   1. Find the maximum orthogonal line that starts at the current baseline.
   2. Define the baseline where the maximum orthogonal line ends as the ending baseline.
   3. Search for the orthogonal lines that also start at current baseline and which are shorter than the maximum orthogonal line.

4. If there are orthogonal lines that satisfy the above condition, then
    - Find the shortest one among them.
    - Change the ending baseline as the baseline where this orthogonal line ends.
5. Define a block as between the current baseline and the ending baseline.
6. Skip the other baselines between the current baseline and the ending baseline. Assign the ending baseline to the current baseline.

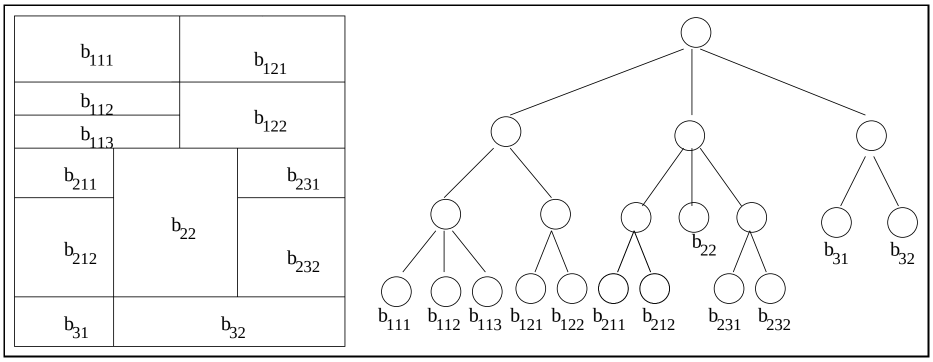


Figure 1: (a)A sample form (b)and its hierarchical tree.

In order to visualize the execution of the algorithm, we will proceed with the form shown in Figure 1. At the first level where the current frame is the form frame, five baselines are found. Initially, the current baseline is the top border of the form. Since there is only one orthogonal line starting from the current baseline, the first block is defined till the third baseline. In defining the next block, there are two orthogonal lines and the block is then defined by the shorter one. The second block is thus between the third and fourth baselines while the last block is between the fourth and the fifth baselines. Whenever execution comes to the third phase, all of the blocks will further be partitioned since none of them is a cell. But this time, division by vertical lines will be performed. Finally, when the algorithm terminates, the tree presented in Figure 1(b) will be constructed.

The above algorithm works fine for simple forms. However, when we consider more complex forms such as shown in Figure 2, some additional conditions should be checked. With the current algorithm, although initially the maximum orthogonal line is searched, in fact blocks are defined

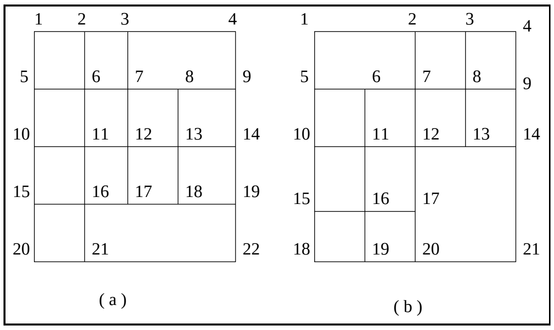


Figure 2: Form documents showing two particular cases (numbers show the vertices).

by the shortest orthogonal line that starts at the current baseline. However, the maximum orthogonal line becomes significant when there are other lines starting from the baselines which succeeds the current one and shares at least two baselines with the shortest orthogonal line(the case where there is one shared baseline is handled by the previous algorithm). Then, a block will additionally be defined by the following rules.

- If both type of orthogonal lines end at the same baseline (e.g.Figure 2(a)), then ending baseline is changed to the next baseline (which is the baseline between the vertex pair (5,9) in Figure 2(a)).
- Otherwise, (e.g.Figure 2(b)) if the orthogonal line that starts at the baseline succeeding the current one ends at the same baseline as the maximum orthogonal, then the ending baseline remains unchanged, i.e. the baseline where the maximum orthogonal line ends (which is the baseline between the vertex pair (18,21) in Figure 2(b)).

As a special case, the current block may consist of only the orthogonal lines with the same length equal to the length of the borders of the block frame and there may be no other line parallel to the baselines. In such a case, there is an ambiguity in defining blocks in the sense that only the vertical and horizontal lines do not give enough information about the type of division. If we consider the example in Figure 3(a), without preprinted data, it may represent both blocks shown in Figure 3(b) and shown in Figure 3(c). This ambiguity can be handled by defining the current block as ambigious and forming new blocks between each baseline. In the tree construction part, a flag is used to define the ambiguity for the node which represents the current block. The rest of the tree is constructed in a usual manner. The hierarchical tree for Figure 3(a) is shown in Figure 3(d).

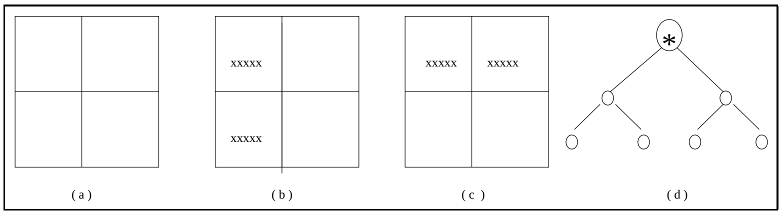


Figure 3: (a)Block without preprinted data, (b),(c) alternatives for preprinted data, (d) corresponding hierarchical tree (* shows the ambiguity).

After defining the ambiguity, we put one more constraint into the algorithm to handle an exceptional case. In a block frame, all of the orthogonal lines starting from the top border of the block frame may have same length as the borders. However, there may be other orthogonal lines which are shorter than the maximum orthogonal line but start at the succeeding baselines. If the basic algorithm were applied, the whole block would be taken as a single one. To avoid this, a block is defined between the current baseline and the next baseline. Then, the next baseline is assigned to the current baseline, as usual. For the remaining block, the ambiguity check is performed, since the remaining block should be taken as a single block if it is ambigious. Otherwise, each two baselines define a block. This process is reasonable, because we have other orthogonal lines that give us the information about division. This exceptional case may also occur if the previous blocks are cells or they are defined between two succeeding baselines and if the maximum orthogonal line for the current baseline ends at the border of the current block.

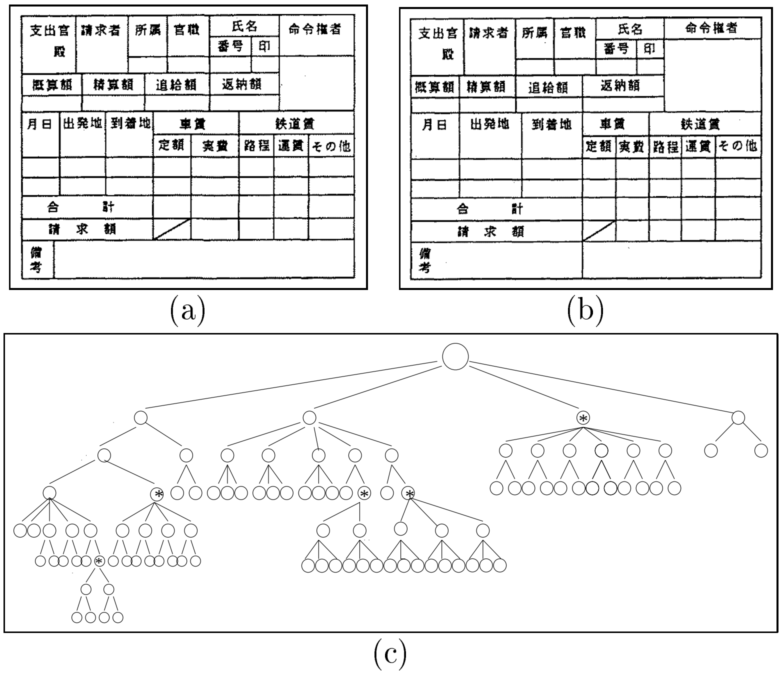


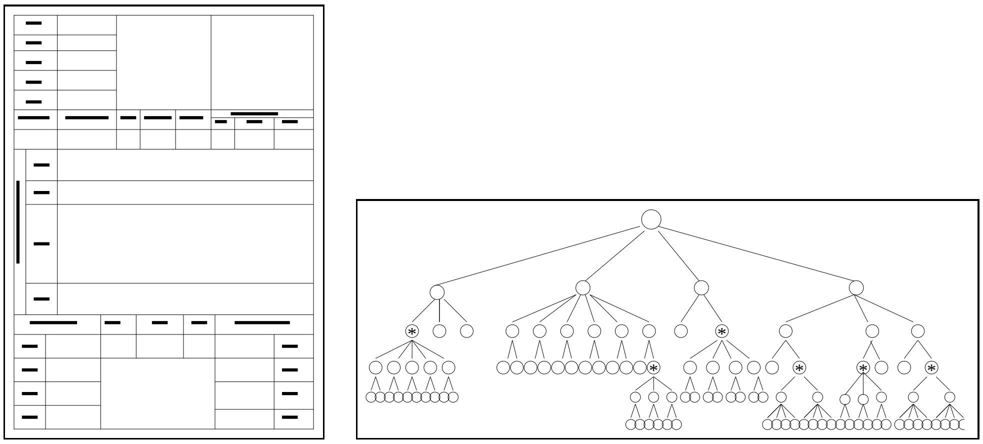
Figure 4: Multi-kinds of the same table form document (adapted from Watanabe et. al. [8]) which are represented by the same hierarchical tree.

5 Examples

In this section, we give two example form documents and their representations. The first example is adapted from Watanabe et. al.[8]. In fact, there are two form documents which are logically same, but physically different. When the proposed heuristic algorithm is applied to these form documents, the same tree is obtained (c.f. Figure 4). The second example form document is an official one used for registration of buildings. It is shown in Figure 5(a) and the black thin lines indicate the preprinted data. The algorithm easily extracts the representation of such a non-table form document as shown in Figure 5(b).

6 Conclusion

In this study, a heuristic algorithm is proposed in order to represent the logical structure of a form document. The algorithm uses a top-down approach and depends on the use of horizontal



(a) (b)

Figure 5: A non-table form document and its hierarchical tree.

and vertical lines existing on a form document. A hierarchical tree is reconstructed to represent the logical structure. The hierarchy of blocks are represented by the hierarchy of the tree. Logically same but physically different forms are associated to the same hierarchical tree. This representation can handle geometrical modifications and slight variations.

References

[1] Y.Tang, C.Yan, M.Cheriet, C.Suen, "Automatic Analysis and Understanding of Documents," in *Handbook of Pattern Recognition and Computer Vision*, pp. 625-654, 1993.

[2] D.Wang, S.Srihari, "Analysis of Form Images," *First Int'l Conf. Document Analysis and Recognition*, Saint-Malo, France, pp. 181-191, Sept. 1991.

[3] R.Casey, D.Ferguson, K.Mohiuddin, E.Walach, "Intelligent Form Processing System," *Machine Vision and Applications*, vol. 5, pp. 143-155, 1992.

[4] J.Mao, M.Abayan, K.Mohiuddin, "A Model-Based Form Processing Sub-System," *Proc. 13th Int'l Conf. Pattern Recognition*, Vienna, Austria, pp. 691-695, 1996.

[5] S.Taylor, R.Fritzson, J.Pastor, "Extraction of Data From Preprinted Forms," *Machine Vision and Applications*, vol. 5, pp. 211-222, 1992.

[6] S.Shimotsuji, M.Asano, "Form Identification based on Cell Structure", *Proc. 13th Int'l Conf. Pattern Recognition*, Vienna, Austria, pp. 793-797, 1996.

[7] Y.Hirayama, "Analyzing Form Images by Using Line-Shared-Adjacent Cell Relations," *Proc. 13th Int'l Conf. Pattern Recognition*, Vienna, Austria, pp. 768-772, 1996.

[8] T.Watanabe, Q.Luo, N.Sugie, "Layout Recognition of Multi-Kinds of Table-Form Documents," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol.17, no.4, pp.432-445, April 1995.