A Hierarchical Representation of Form Documents for Identification and Retrieval

Pınar Duygulu and Volkan Atalay

Department of Computer Engineering,
Middle East Technical University,
Ankara, Turkey

ABSTRACT

In this paper, we present a logical representation for form documents to be used for identification and retrieval. A hierarchical structure is proposed to represent the logical structure of a form by using lines. The approach is top-down and no domain knowledge such as the preprinted data or filled-in data is used. Logically same forms are associated to the same hierarchical structure. This representation can handle geometrical modifications and slight variations.

Keywords: form document processing, logical layout extraction, retrieval, data processing

1. INTRODUCTION

With recent advances in multimedia technology, various types of data become available to be processed by computers as input to information systems. However, most of the time, data has to be extracted and manipulated manually. One of the challenges is to automate this information extraction process from raw data. Paper is still one the most commonly used medium for transferring information. Among paper documents, *forms* are structured documents used for information gathering, storage, retrieval, approval, and distribution. Traditional manual key entreying of data on forms is tedious, time consuming and error prone. Consequently, it is highly desirable to automatize this task as much as possible.

There are several challenging problems in form processing; layout extraction, retrieval, processing heterogeneous batches, etc. The last problem, for example, requires identifying the particular type of input form, registering the input form with a reference model form, extracting information from the input form, and storing the extracted information.

Definition 1: A *form* is a structured document which is composed of the following elements:

- horizontal and vertical layout lines: straight and continuous;
- preprinted data: machine printed characters, symbols and pictures;
- user filled-in data: machine typed, handprinted or handwritten characters.

Preprinted data and user filled-in data are located at predefined locations which are called as *fields*. A field is enclosed by a rectangular box formed by two horizontal and two vertical lines. Fields frequently have inter-relationships where the data entered in one field validates the data entered in another field. Thus, on a form the information is generally kept in tuples, preprinted being the first element and the user filled-in the second.

One of the interests in form processing systems is to extract user filled-in data and associate it with the corresponding preprinted field. However, in order to perform such a task, the structure of the form should be known in advance. The form structure can be obtained by processing a *reference model form*. The reference model form may be stored in a form database and when an *input instance form* is presented to the system, its type can be identified by matching with one of the reference model forms in the database. A reference model form can be for example, a

Further author information: (Send correspondence to P. Duygulu)
P. Duygulu: E-mail: duygulu@ceng.metu.edu.tr
V. Atalay: E-mail: volkan@ceng.metu.edu.tr

blank form on which no user filled-in data exists. The reference model form can be used to specify the fields on the form. Required data can then be automatically extracted following the specification given for the reference model form.

Another interest may be to retrieve similar form documents from a large database. Either for data processing(identification) or for retrieval purposes, in order to store a reference model form in database and also to compare an input instance form with a stored reference model form, we need a representation. The most straightforward way to represent a form document is by its direct image. But, such a physical information may not be appropriate, since the form or its image risks 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 same logical structure 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 represented by the same logical structure.

In this paper, we present a logical representation for form documents to be used for identification and retrieval. We describe a heuristic algorithm in order to transform geometric structure of a form into a logical structure by using horizontal and vertical layout lines which exist on the form. The logical structure is represented by a 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

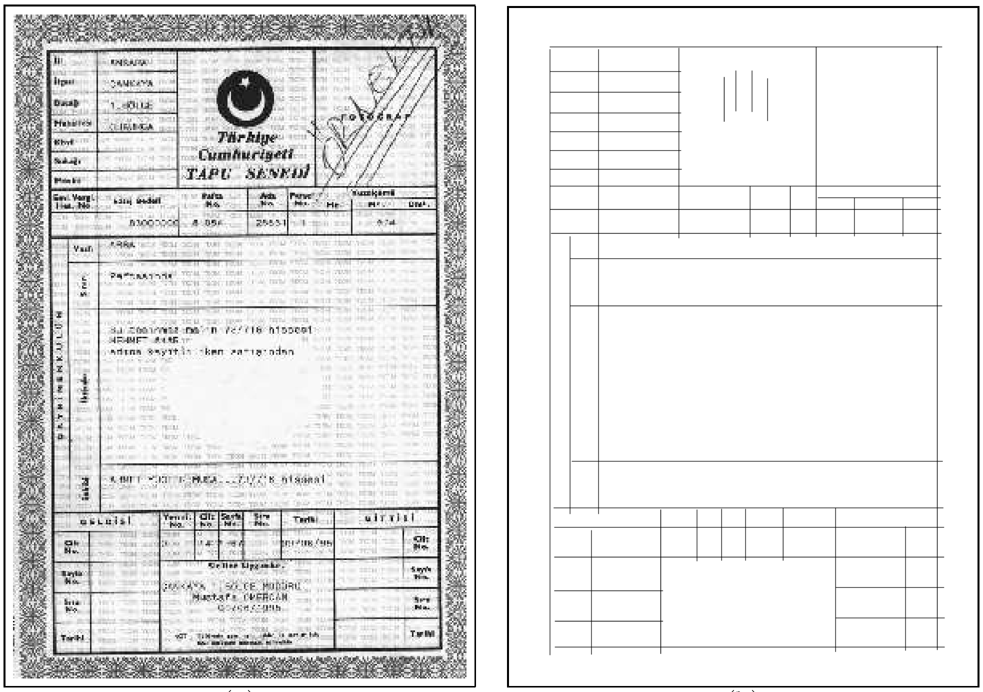
Rather than its direct image, a form document can be represented by the physical features of its components.[6,8–11] For example, length, width and position of horizontal and vertical lines can be used.[6] However, this cannot handle variations on the physical structure of logically same form. Another feature related to the horizontal-vertical lines is the line crossings. Extracted line crossings on a given form can be classified into one of various types and the form can be represented by the set of type counts.[8] However, this approach will not work even for slight variations, because the types and counts of line crossings are prone to change. On a form document, a block is defined to be a rectangular area which is surrounded by horizontal and vertical lines. In this sense, it is a higher level feature than the lines. For form representation, the position and size of blocks or their relationships can be used.[9–11] 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.[10] 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.*,[11] 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.[9] The registration is performed by seraching the classification tree.

3. LINE EXTRACTION

Line extraction process consists of the following components: binarization , Hough transform, tracing and postprocessing. Since the form documents tested in this study are not very noisy, iterative optimal thresholding is applied for binarization. Hough transform is adapted in a manner, to express the structured line segments only in horizontal and vertical directions more accurately, the θ parameter is constrained to be around 0 or 90 degrees. ρ is quantized at pixel level. The accumulator is then searched for local maxima to identify lines. However, in this case a whole line may be fragmented into many overlapping lines. To solve this problem we apply the following line tracing algorithm.

Let L_t be the horizontal line being traced at y=y_t. L_1 and L_2 be two other close horizontal lines with y_1=y_t-1 and y_2=y_t+1 respectively.

$\mathsf{L_t : (x_t^i, y_t) \quad (x_t^f, y_t)}$
$\mathsf{L_1 : (x_1^i, y_1) \quad (x_1^f, y_1)}$
$\mathsf{L_2 : (x_2^i, y_2) \quad (x_2^f, y_2)}$
$\mathsf{f(x,y) \in \{Background, Object\} \ \forall(x,y) \ \text{where} \ 0 \le x < N \ \text{and} \ 0 \le y < M.}$



(a) (b)

Figure 1. A sample form and its extracted lines.

```
x_temp = 0
WHILE (x_temp < N)
   BEGIN
      WHILE (f(x, y) = Background)
            x_temp = x_temp + 1
      x_start = x_temp
      WHILE (f(x_temp, y_1) = Object OR
            f(x_temp, y_2) = Object OR
            f(x_temp, y_t) = Object)
            x_temp = x_temp + 1
      x_end = x_temp
      IF ((x_end - x_start) > ε)
            x_t^i = x_start
            x_t^f = x_end
   END
```

Thick lines on the original document may still be represented by more than one overlapping one-pixel wide lines after the above algorithm. Thus, a postprocessing is applied. The number of times a surviving line has been processed is determined to indicate the maximum thickness on the document.

Figure1(b) presents the final extracted lines for the form document shown in Figure 1(a). During line extraction, we do not consider skew detection and correction since there exist several excellent techniques for the solution of this problem.

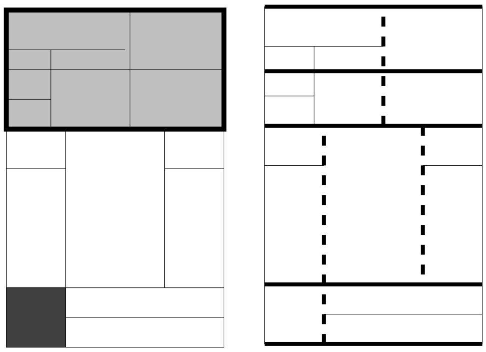
4. DEFINITIONS AND APPROACH

The physical structure of a form consists of horizontal and vertical layout 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 are given and then we resume the approach.

4.1. Definitions

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.

In Figure 2 (a) the light gray area shows an example block, the thick black lines shows the block frame for this block and the dark gray area at the bottom shows a cell. In Figure 2 (b), baselines for the initial block, which is the form itself, are shown by thick black lines and orthogonal lines are shown by dashed lines.



(a) (b)

Figure 2. An example form to demonstrate the defined terms.

4.2. Approach

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 blocks which contain similar information are grouped together and these group of blocks 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.

5. THE HEURISTIC ALGORITHM

The algorithm consists of three phases : initialization, block finding ($\mathsf{BlockFinding}$) and tree construction ($\mathsf{TreeConstruction}$). The first phase is used to initialize the current block frame ($\mathsf{block}_{\mathsf{curr}}$) to the form frame, to create a queue for the blocks to be processed and to indicate "horizontal" division initially. Partitioning of the current block frame is performed at the second phase. Extracted baselines (b_i, $\mathsf{i} = 1, \ldots, \mathsf{n}$) for the current block frame are processed and a block is always defined between the currently processed baseline ($\mathsf{b}_{\mathsf{curr}}$) and the corresponding ending baseline ($\mathsf{b}_{\mathsf{end}}$). Ending baseline is sought through orthogonal lines that intersect with the other baselines. The partitioned blocks are stored to be inserted into the tree and then in order to be further divided into smaller blocks, if possible. First, we give definitions of variables and then the main part of the algorithm.

DEFINITIONS

$\mathsf{block_{curr}}$: current block
$\mathsf{b_{curr}}$: current baseline
$\mathsf{b_{end}}$: ending baseline
BlockQ : queue of blocks
$\mathsf{B} = \{\mathsf{b_i} \mid \mathsf{i} = 1, \ldots, \mathsf{n}\}$: extracted baselines for $\mathsf{block_{curr}}$
$\mathsf{O} = \{\mathsf{o_i} \mid \mathsf{i} = 1, \ldots, \mathsf{m}\}$: basic orthogonal lines starting at $\mathsf{b_{curr}}$
$\mathsf{O}' = \{\mathsf{o_i}' \mid \mathsf{i} = 1, \ldots, \mathsf{k}\}$: other orthogonal lines starting at baselines succeeding $\mathsf{b_{curr}}$
n : number of baselines for $\mathsf{block_{curr}}$
m : number of basic orthogonal lines for $\mathsf{b_{curr}}$
k : number of other orthogonal lines for the succeeding baselines

O, set of basic orthogonal lines starting at the currently processed baseline and O', set of other orthogonal lines starting at baselines succeeding the current baseline are employed to determine the ending baseline for a block.

ALGORITHM

```
/* initialization */
block_curr ← Form
block_curr.direction ← horizontal
CreateQueue(BlockQ)

BlockFinding(BlockQ, block_curr)
TreeConstruction(BlockQ, block_curr)
```

Defining a block, particularly determining the ending baseline is not a very straightforward process. A basic algorithm for BlockFinding is given as follows.

```
BlockFinding (BlockQ, block_curr)

    Construct B = {b_1, b_2, ..., b_n}
    i ← 1
    While (i<n) Do
        b_curr ← b_i
        FindEndingBaseline(B, b_curr, b_end)
        DefineBlock(b_curr, b_end, block)
        AddQueue(BlockQ,block)
        b_curr ← b_end
        While (i < ord(b_end)) Do
            i ← i + 1
```

After finding the blocks, each of them is inserted to the tree which eventually represent the logical structure of the form. TreeConstruction procedure performs tree insertion operation in addition to the process of the whole algorithm until the cells are reached. If the processing continues, than the direction is switched.

```
TreeConstruction(BlockQ, block_curr)

    While Not EmptyQueue(BlockQ)
        DeQueue(BlockQ, block)
        InsertTree(block_curr, block)
        If Not Cell(block) Then
            block_curr.direction ← NOT(block_curr.direction)
            BlockFinding(BlockQ, block)
            TreeConstruction(BlockQ, block)
```

5.1. A Basic Algorithm for Finding Ending Baseline

If there are only borders of the block frame orthogonal to the baselines, i.e. $m = 0$, then each baseline defines a block. Otherwise, the shortest basic orthogonal line that starts at the current baseline defines the ending baseline.

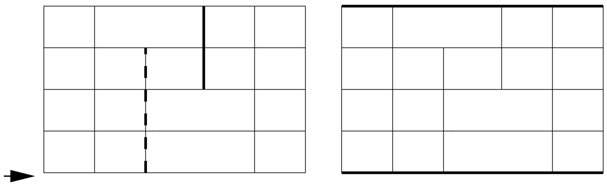
```
FindEndingBaseline(B, b_curr, b_end)
  Construct O = {o_1, o_2, ..., o_m}
  Construct O' = {o'_1, o'_2, ..., o'_k}
  If (m=0) Then
    b_end ← b_curr.next
  Else
    o_shortest ← FindShortestOrthogonal(O)
    b_end ← o_shortest.end
  return b_end
```

5.2. Improved Algorithm for Finding Ending Baseline

Blocks are defined by the shortest basic orthogonal line that starts at the current baseline. if there are baselines shared among several orthogonal lines, then the ending baseline cannot be found by the information simply coming only from a single orthogonal line, since in this case the orthogonal lines will not be independent from each other. Therefore, information from all of the orthogonal lines should be combined to solve the complex set of relations. "Other orthogonal lines", that is the orthogonal lines that are in the same current block but which start from baselines other than the current one become important. For example, if there is no grouping mechanism, such as all of the orthogonal lines do not start or do not end at the same baselines, then the block should be taken as a whole. Thus, we have to check starting and ending of all orthogonal lines.

Currrent criterion to determine block ending is the end of shortest basic orthogonal line. If at least one of the "other orthogonal lines" do not end at the same baseline as the shortest basic orthogonal line, then the ending baseline is taken as the one at which the longest basic orthogonal line ends. On the other hand, if there are "other orthogonal lines" which end with the same baseline as the shortest basic orthogonal line, then this "other orthogonal line" defines a block and its starting baseline should indicate the ending of the current block. We improve the algorithm to determine the ending baseline for a block as follows.

```
FindEndingBaseline(B, b_curr, b_end)
  Construct O = {o_1, o_2, ..., o_m}
  Construct O' = {o'_1, o'_2, ..., o'_k}
  If (m=0) Then
    lines ← 0
  Else
    o_shortest ← FindShortestOrthogonal(O)
    o_longest ← FindLongestOrthogonal(O)
    lines ← 1
    for i=1 to k do
      if o_shortest.end > o'_i.start and o_shortest.end < o'_i.end then
        lines ← 2
        break
      Else
        temp ← o'_i
        lines ← 3
  case lines
    0:   b_end ← b_curr.next
    1:   b_end ← o_shortest.end
    2:   b_end ← o_longest.end
    3:   b_end ← temp.start
  return b_end
```



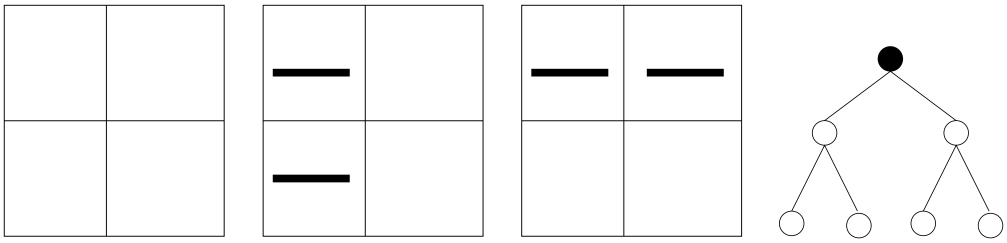
(a) (b)

Figure 3. Another form document showing the other the particular case. (case 3.B)

If the improved algorithm is applied to the form shown in Figure 3 (a), then the ending baseline is assigned to the line where the longest orthogonal line ends (as shown in Figure 3 (b)) since the other orthogonal line shown by dashed line do not end at the same line where the shortest orthogonal line ends (the baseline shown by the arrow).

5.3. Ambiguity

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 4(a), without preprinted data, it may represent both blocks shown in Figure 4(b) and shown in Figure 4(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 tree for Figure 4(a) is shown in Figure 4(d).



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

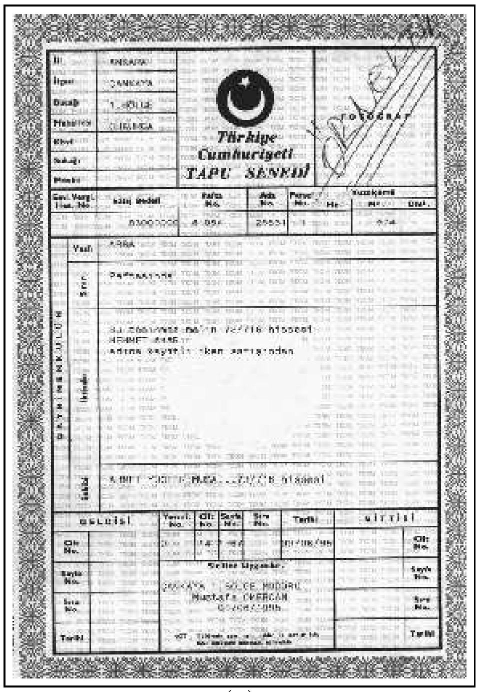
Figure 4. (a)Block without preprinted data, (b),(c) alternatives for preprinted data, (d) corresponding hierarchical tree (black node shows the ambiguity).

The ambiguity flag in the tree, brings a flexibility to the registration process. Without knowing about preprinted or user filled-in data, by using only horizontal and vertical lines, the forms can be registered even if they have slight variations.

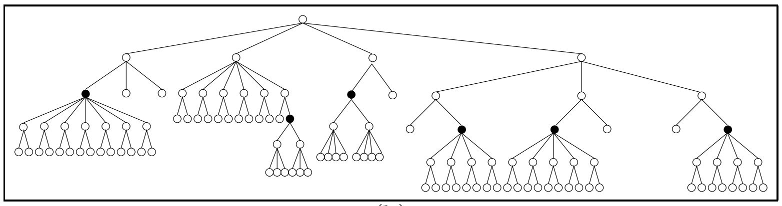
5.4. Example Form Documents and Their Hierarchical Structures

As it can be observed, two forms shown in Figure 5 are designed for the same application. However, there are some slight differences. The first, third and the fourth main blocks on these forms are same but the second block which corresponds to tax information is more detailed in the second form. With the heuristic algorithm, very similar hierarchical trees are obtained from these two forms. The trees are exactly same for the blocks which have the same information (first, third and fourth nodes at the first level) and they have small differences for the blocks which are different in the forms (the second node at the first level).

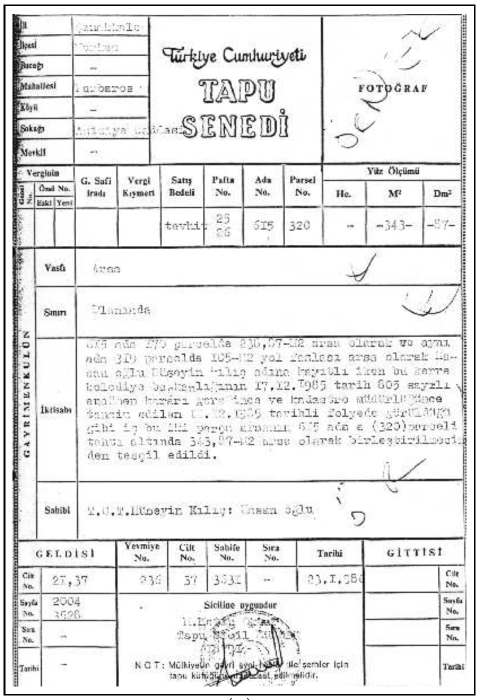
The tabular forms shown in Figure 6 are adapted from Watanabe et. al.[11] They are same application specific forms, however they have different physical structures. With the heuristic algorithm they are represented with the same hierarchical tree.



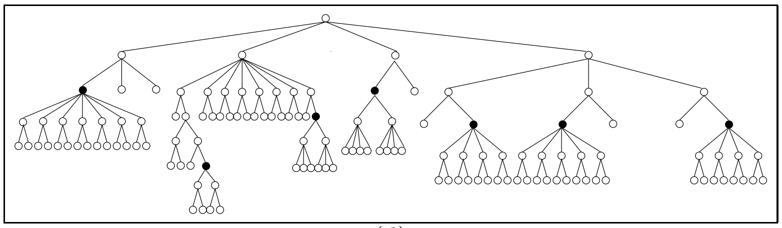
(a)



(b)



(c)



(d)

Figure 5. Example forms (a,b) and corresponding hierarchical structures (c,d)

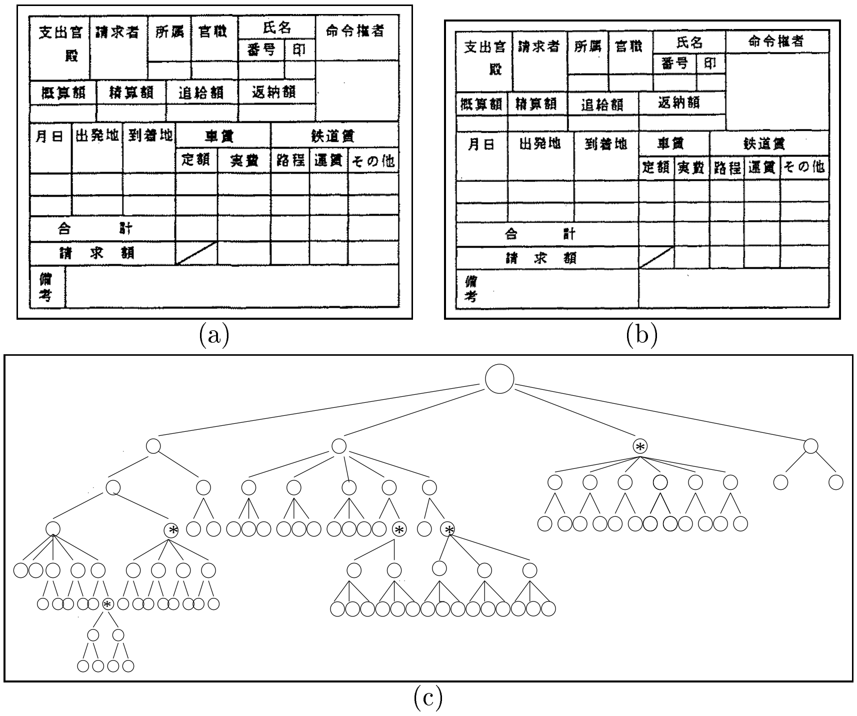


Figure 6. (a),(b) Two tabular forms adapted from Watanabe et.al. and (c) the hierarchical tree which corresponds both of the forms.

6. RETRIEVAL OF FORM DOCUMENTS VIA HIERARCHICAL REPRESENTATION

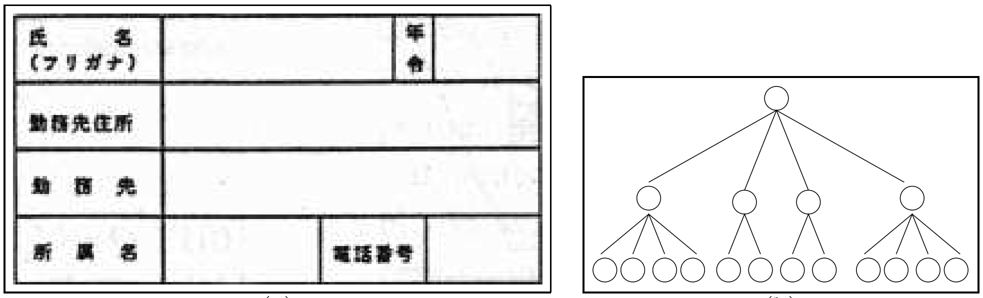
The described hierarchical structure may be used for identification, i.e. retrieval of the reference model form corresponding to an input instance form and for retrieval, i.e. retrieving similar forms. Both identification and retrieval can be realized by comparing hierarchical structures. Since the hierarchical structures in fact correspond to trees, and tree matching is known to be NP-hard problem, a coding of the trees is preferred. Such a coding then allow comparison of two forms.

6.1. Retrieval of Reference Model Form for Registration and Parsing

In order to extract the user filled-in data the input instance form should be registered with one of the reference model forms in the database. An input instance form is registered with one of the model forms in the form database if their structures are identical. In our approach, the identification of forms are simplified by identification of trees. However, since there may be ambigious branches in one of the tress, sometimes unification is performed instead of direct identification. Unifiability is possible, if one of the trees has an ambigious subtree and the other has the corresponding non-ambigious subtree. If the representations of hierarchical trees can directly be matched, then the trees are defined as identical. Otherwise, if there exists an ambiguity in any of the trees, the number of the nodes of two subtrees should be equal in order to be defined as unifiable.

Registration of trees (identification or unification) is performed in level order. Since the hierarchy of the tree corresponds to the hierarchy of the blocks in the form, lower levels carries more information than the higher levels (remark that the root corresponds to level 0). Thus, if the trees are not matched at lower levels, then the registration process stops.

After the registration of forms, the information is obtained from the input instance form by using a parser. The leaf nodes of the reference model form corresponds to the locations that carries the information, preprinted data or user filled-in data. Since the input instance form and the reference model form should have the same hierarchical tree, corresponding leaf nodes represent the same fields in the forms. The parser is used to extract the coresponding fields. The information about the field is taken from the reference model form and the user filled-in data is taken from



(a) (b)

Figure 7. A sample form and corresponding hierarchical tree.

the input instance form to process further. Both reference model form and input instance form keeps the information about the lines that surrounds cells at the leaf nodes. Since the approach is top-down, there is no need for an extra process to get the locations of the lines. The information is carried from the form frame to the cell frames. Each tree can keep its corresponding physical coordinates and this handles the problems due to translation or scaling.

After parsing the hierarchical tree, only the leaf nodes that corresponds to the locations where the user filled-in data exists are extracted. The extracted areas are processed further in order to be given to an *Optical Character Recognition (OCR)*.

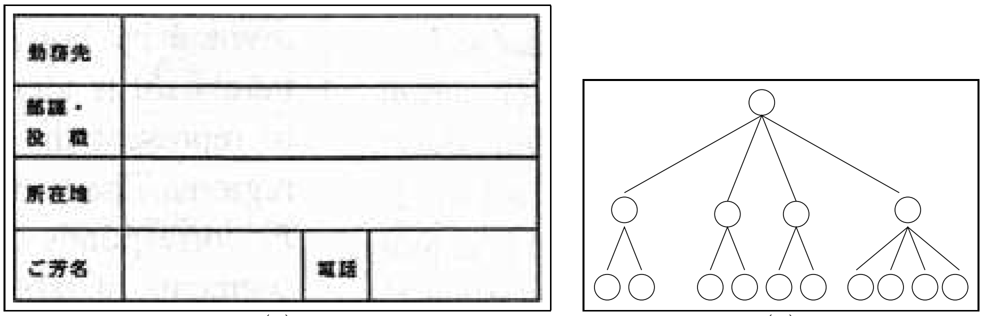
6.2. Retrieval of Similar Forms

The retrieval problem can be stated as follows[17] : "Given a form image database and a query image, how do we retrieve form images in the database with the same or similar layout structure as the query?" . There may be several motivations for retrieval: same forms at different scales (resolutions), minor physical variations which occur through the years, etc. In the case of scale change, the hierarchical structure remains the same. However, minor variations are reflected as additional subtrees.

The coding scheme for a tree is defined by associating a number to a tree node. The hierarchy is reflected by giving the highest priority (number) to the root node. Lower priorities are assigned to its child nodes and so on. The coding scheme utilizes the unique decomposition of a natural number into a product of primes. Such codes are called Gödel numberings after Kurt Gödel, who developed the technique. In this study, we assign prime numbers to the nodes. Then, we obtain a single number by multiplying these numbers similar to Gödel numbering.

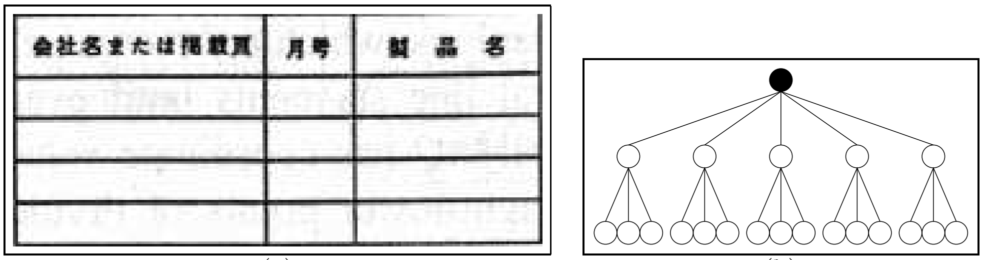
Although such an approach gives unique numbers to each tree, when the form has a detailed structure, then this coding scheme results a very big number whose manipulation is difficult. Thus, we modify the coding scheme, and instead of multiplication we use the addition of prime numbers assigned to each node.

Assuming that, the highest number of levels in a tree is 10, we assign 29 to the root of the tree and the other prime numbers to the other nodes in decreasing order. For example, the form in Figure 7 is coded as $29+4*23+12*19 = 349$, the form shown in Figure 8 is coded as $29 + 4 * 23 + 10 * 19 = 311$ and the form shown in Figure 9 is coded as $29 + 5 * 23 + 5 * 3 * 19 = 429$. The difference in the form documents, thus the difference in the extracted hierarchical structures are reflected on the codes. For retrieval purposes, it can be observed from the codes that the form in Figure 7 is relatively similar to the form in Figure 8 rather than the form in Figure 9. The similar forms shown in Figure 5(a) and (c) are coded as $29 + 4 * 23 + 3 * 19 + 7 * 17 + 7 * 2 * 13 + \mathbf{6 * 19 + 6 * 2 * 17} + 2 * 13 + 2 * 3 * 11 + 1 * 19 + 2 * 17 + 2 * 4 * 13 + 3 * 19 + 2 * 17 + 4 * 13 + 4 * 2 * 11 + 2 * 17 + 5 * 13 + 5 * 2 * 11 + 2 * 17 + 4 * 13 + 4 * 2 * 11 = 1679$ and as $29+4*23+3*19+7*17+7*2*13+ \mathbf{8 * 19 + 8 * 2 * 17 + 2 * 13 + 4 * 11 + 2 * 7 + 2 * 2 * 5} + 2*13+2*3*11+2*19+2*17+2*4*13+3*19+2*17+ 4*13+4*2*11+2*17+5*13+5*2*11+2*17+4*13+4*2*11 = 1889$. The differences between the coding of these similar forms is shown as bold characters. The difference is relatively slight with respect to the coding values. (210/1679 and 210/1889) This is because the difference is in the lower level subtrees and this contributes to the code by small coefficients.



(a) (b)

Figure 8. A sample form and corresponding hierarchical tree.



(a) (b)

Figure 9. A sample form and corresponding hierarchical tree.

7. CONCLUSION

Identification and retrieval is an essential process even for a single type of form, since there may be modifications on the physical structure of the form or there may be more than one design for the same application specific form. Most of the current studies use physical features and can handle only some translation or scaling problems. However, there is a need for more logical features if the forms have several different physical structures and if the number of forms to be compared is large.

In this study, the logical structure of the forms are represented by a tree which corresponds to the hierarchy of the blocks on the form. Since the forms are designed in a hierarchical manner, that is the blocks which contain similar information are grouped together and seperated from the other groups, this representation is also close to the human point of view.

The described hierarchical structure is used both for identification and retrieval. A coding scheme is described for simple comparison of hierarchical structures. Experimental results are promising. However, it is necessary to build a database containing a large number of form documents and do extensive experimental work to test the ideas described in this study.

REFERENCES

1. Y. Y. Tang, C. D.Yan, M.Cheriet, C. Y.Suen. Automatic Analysis and Understanding of Documents. In *Handbook of Pattern Recognition and Computer Vision*, pp. 625–654, 1993.
2. D.Niyogi, S.Srihari. Using domain knowledge to derive the logical structure of documents. *SPIE*, pp. 114–125, 1996.
3. B.Yu, A. K.Jain. A Form Dropout System. *Proc. 13th Int. Conf. on Pattern Recognition*, ICPR'96, Vienna, Austria, August, 1996, pp. 701–705.
4. D. Wang, S. N.Srihari. Analysis of Form Images. *Proc. First Int. Conf. on Document Analysis and Recognition*, ICDAR'91, Saint-Malo, France, Sept, 1991, pp. 181–191.

5. R. Casey, D.Ferguson, K. Mohiuddin, E.Walach. Intelligent Forms Processing System. *Machine Vision and Applications*, vol.5, no.3, pp. 143–155, 1992.
6. J.Mao, M.Abayan, K.Mohiuddin. A Model-Based Form Processing Sub-System. *Proc. 13th Int. Conf. on Pattern Recognition*, ICPR'96, Vienna, Austria, August, 1996, pp. 691–695.
7. J.Lin, C.Lee, Z.Chen. Identification of Business Forms Using Relationships Between Adjacent Frames. *Machine Vision and Applications*, 9:56–64, 1996.
8. S.Taylor, R.Fritzson, J.Pastor. Extraction of Data From Preprinted Forms. *Machine Vision and Applications*, vol.5, pp. 211–222, 1992.
9. S.Shimotsuji, M.Asano. Form Identification based on Cell Structure. *Proc. 13th Int. Conf. on Pattern Recognition*, ICPR'96, Vienna, Austria, August, 1996, pp. 793–797.
10. Y.Hirayama. Analyzing Form Images by Using Line-Shared-Adjacent Cell Relations. *Proc. 13th Int. Conf. on Pattern Recognition*, ICPR'96, Vienna, Austria, August, 1996, pp. 768–772.
11. 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.
12. J.Liu, X.Ding, Y.Wu. Description and Recognition of Form and Automated Form Data Entry. *Proc. Third Int. Conf. on Document Analysis and Recognition*, ICDAR'95, 1995, pp. 579–582.
13. O.Hori, D. S.Doermann. Robust Table-form Structure Analysis Based on Box-Driven Reasoning. *Proc. Third Int. Conf. on Document Analysis and Recognition*, ICDAR'95, 1995, pp. 218–221.
14. S. Srihari, Y. Shin, V. Ramanaprasad, D. Lee, Name and Address Block Reader System for Tax Form Processing, *Proc.* 3^{rd} *Int. Conf. on Document Analysis and Recognition(ICDAR)*, vol. 1, pp. 5-10, 1995.
15. P.Heroux, S.Diana, A.Ribert, E.Trupin. Classification Method Study for Automatic Form Class Identification. *Proc. 14th Int. Conf. on Pattern Recognition*, ICPR'98, Brisbane, Australia, August, 1998, pp. 926–928.
16. C.Cracknell, A.C.Downton. Document Image Understanding of Handwritten Forms using Rule-trees. *Proc. 14th Int. Conf. on Pattern Recognition*, ICPR'98, Brisbane, Australia, August, 1998, pp. 936–938.
17. J.Liu, A.K.Jain. Image-based Form Document Retrieval. *Proc. 14th Int. Conf. on Pattern Recognition*, ICPR'98, Brisbane, Australia, August, 1998, pp. 626–928.
18. E.Dincel. A Form Document Image Parser. M.S. Thesis, Middle East Technical University, Ankara, Turkey, June, 1998.
19. D. Trier, A. Jain, A Goal-Directed Evaluation of Binarization Methods, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 1191-1201, 1995.
20. D. Trier, T. Taxt, Evaluation of Binarization Methods for Document Images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, pp. 312-315, 1995.
21. Gauch J., http://www.eecs.ukansas.edu/$\tilde{j}$gauch.
22. R. Gonzalez, J. Woods, *Digital Image Processing*, Addison Wesley Publishing Company, 1993.