Automated Table Understanding using Stub Patterns Analysis

Roya Rastan, Hye-Young Paik, John Shepherd
{rrastan,hpaik,jas}@cse.unsw.edu.au

School of Computer Science and Engineering,
University of New South Wales, Sydney, Australia

Abstract. Tables are a widely-used structure for data presentation and summarisation in documents from many different domains. However, most of tables are designed for human readers and their layout and logical structure are not well defined for machine processing. As a result, the tasks of extracting data from tables are not yet fully automated, and existing solutions tend to focus on a domain or application, relying on external knowledge sources. Considering the diversities of tables in their topological structure, finding an automated solution that works for all different tables is difficult. In this paper, we identify some of the common features in the logical structures of tables by analysing the header arrangement patterns as a step towards building an automatic processing technique. This allows us to describe a set of table understanding rules and heuristics in an unambiguous way. Based on the patterns, we have developed our own table understanding system. We report on its performance on well-known datasets.

Keywords: Table Understanding, Table Logical Structure, Table Stub Analysis, Table Categories, Category Hierarchy

1 Introduction

Tables are a widely-used structure for data presentation and summarisation in documents. In tables, layout (positioning of cells and lines) is used to encode the data into different arrangements to present and communicate complex relationships. Human readers, in turn, decode the different layout features and use them as clues for understanding/interpreting the meaning of the information in tables.

Since tables are a rich and easily accessible source of inter-related data, there has been significant effort dedicated to automatically extracting and manipulating the data by computers [21, 4, 2, 8].

The computing task of *Automatic Table Understanding and Interpreting* goes beyond detecting table structure or layouts. By exploiting the same features that humans use, it tries to extract the relationships amongst table cells (e.g., header cell, data cell) and ultimately obtain a representation of the table data which is

independent of its original layout. Different levels of "knowledge" may be considered for the task and result in different depths of table understanding. For example, one could purely rely on the table structure, or include a document context into the analysis, or consider external knowledge sources such as a domain ontology or knowledge base.

As pointed out in [11], less reliance on external knowledge sources makes table understanding solutions more generic and reusable. Some of the recent work in this area [18, 10] takes this approach and focuses on the various features within a table (e.g., layout, cell content) as the only source of knowledge for designing table understanding systems. The aim of such work is to automatically transform the table data into a generic data structure and create a basis for further data analysis (e.g. semantic relation detection, query answering).

Considering the diversity in the structure of tables, it is difficult to formulate an one-solution-fits-all technique for understanding tables this way. In most existing work, the arrangement of table headers (both row and column headers) is viewed as the main cause of diversity. It is generally accepted that tables can be *understood* if one can detect the hierarchical structure of table header cells properly and determine how each table data cell can be uniquely accessed through them.

A range of different approaches have been taken to deal with this complexity. These include, for instance, only processing a specific type of table such as *Well-Formed Tables* [6] and *Multi-Dimensional Tables* [1], or involving a human in the header detection process [15], or removing all layout features in a table and just analysing the tables based on the cell arrangement of their headers [18].

Although all of these have contributed to advancing table understanding techniques, most of the work until now treated row and column headers in the same way with regards to extracting the header hierarchy. However, we argue that there are certain groups of features which are unique in row headers (*table stub*) that can convey more accurate structural information and excluding such features from a table understanding process will lose information in processing some tables.

In this paper, we identify patterns of different cell arrangements and content formatting features that commonly occur in the stub region of tables. This classification enables us to detect the hierarchical structure in table headers in cases which would be missed by existing methods. The classification also gives us a concrete basis for building automatic table understanding algorithms that can be effectively applied to a wider range of tables. Specifically, we do not assume that tables are *well-formed* (in the sense of [9]), and we do not require human intervention during the process. Based on the patterns we have identified, we have developed a set of algorithms to accurately identify the hierarchical relationship between table headers. In our implementation, we receive a segmented table as input and provide a table header structure as output, as well as all *access paths* for each data cell. We evaluate our methods on two well-known public datasets.

Concretely, we make the following contributions: (i) we provide a classification of different table header features in table stubs and an interpretation of

the logical structures they imply; (ii) we present algorithms for automatic table understanding that make fewer assumptions about the source tables than existing algorithms, (iii) we demonstrate how our table understanding solution leads to straight-forward transformation of source tables to the well-known Wang Abstract Table [19] model.

2 Preliminaries

In this section, we explain the concepts and terminologies used throughout the paper to make the paper self-contained. It is noted that more detailed descriptions and discussions about these can be found in [16].

2.1 Regions in Tables

In order to discuss table structures, we follow Wang's table terminology [19] which is a well-known reference in the area.

As shown in Figure 1, Wang divides a table into four main regions: *stub*, *stubhead*, *boxhead* and *body*. The regions are delineated by a *stub separator* and a *boxhead separator* which are frequently, but not always, shown as physical lines. An *entry* in a table is the basic data. An *entry cell* contains an entry. A *label* is auxiliary data used to locate the basic data. A *label cell* contains a label.

The body (lower-right) contains entry cells. The boxhead (upper-right) consists of label cells whose values are used to identify access paths to individual columns. The cells in the stubhead (upper-left) and stub (lower-left) contain labels whose content values are used to identify access paths to individual rows.

A combination of access paths, originating from the boxhead and stub, ultimately leads to *entries*.

2.2 Categories, Labels and Access Paths

The above discussion focuses on the physical features of tables. Now we introduce the logical features of tables that are relevant to *Table Understanding*.

Categories and Labels: A table is a two-dimensional representation of a multi-dimensional space. The labels are arranged into hierarchies that map the multi-dimensional space onto the two-dimensional data grid. We consider that each dimension represents a logical relationship between a label and an entry. In Wang's work, a dimension is referred to as a *category*. A category may contain sub-categories and/or labels. The categories that form the root of a hierarchy are called *top-level categories*.

Take the table shown in Figure 1 as an example. The *entries* are average marks of the assignments and examination of a course. `Year` is a label denoting a category $Year$ that consists of the labels `1991` and `1992` $Term$ is a category that consists of the labels `Winter`, `Spring`, and `Fall`. $Mark$ is a category that consists of two sub-categories $Assignment$ and $Examination$and a label `Grade`.

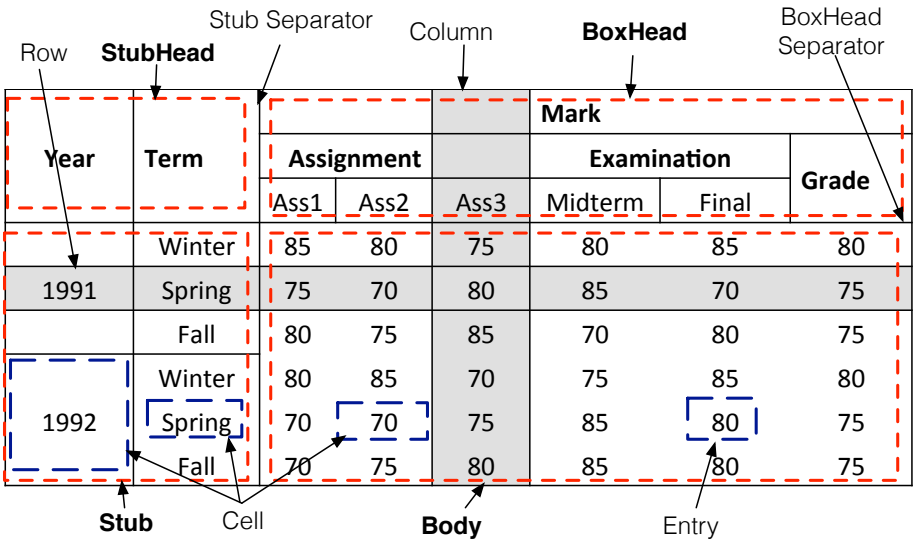

Fig. 1. Wang Table Terminology

Access Paths: Each entry is associated with each of the top-level categories. An *access path* is a sequence of labels that leads from a top-level category to a row or a column. An entry can be *uniquely accessed* via a combination of the access paths from *all* top-level categories. One or more of these access paths starts from the boxhead region; one or more of these starts from the stub region.

In the table in Figure 1, there are three top-level categories: $Year$, $Term$ and $Mark$. Take the entry 85 at the top-left corner of the body as an example. We can define three access paths: `Year.1991`, `Term.Winter` and `Mark.Assignment.Ass1`.

In our work, *Table Understanding* focuses on detecting the categories and the access paths to the entries through them. The output of *Table Understanding* is modelled in the generic table model proposed by Wang called *Abstract Tables* [19] components of which we have described above. Since an Abstract Table presents the logical relationships in a table, it is considered to be a 'presentation/layout independent' representation of the table. Such a representation has been found to be useful by many applications in table data extraction [11, **?**,**?**].

2.3 Annotated Table Metadata

TO-DO: Add a short description about the table metadata and CSV-based annotated table standard. We will make it short, but clear. The page limit is 15 - and we still need some room for improving evaluation part.

2.4 Tasks in End-to-End Table Processing

An end-to-end table processing is divided into *Table Extraction*, *Table Understanding* and *Table interpretation*. The table extraction deals with locating tables in a document and segmenting them into individual cells, rows and columns. We outlined our own design and implementation of an end-to-end table processing in a system named TEXUS [16], in which we provide a concrete set of pre-defined tasks that are put together to form a processing pipeline (Figure 2). The pipeline takes a document as input and produces an *abstract table* for each table located in the input document. We define the following tasks: (1) Document Converting: converts the input document to our own document model, (2) Locating: finds the tables and their outer boundaries in the input, (3) Segmenting: recognises the inner boundaries in a table (cells, rows and columns), (4) Functional Analysis: identifies the role of each cell in from a segmented table (entries or labels), and (5) Structural Analysis: detects the categories and access paths. The result is an Abstract Table.

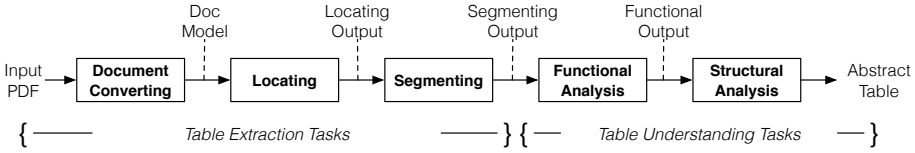

Fig. 2. An End-to-end Table Processing Pipeline in TEXUS

Concretely speaking, we consider the term *Table Understanding* as a sequence of two tasks: functional analysis followed by structural analysis. Later in the paper, we present the implementation of our *automatic table understanding* algorithms in terms of these two tasks.

It is also noted that for the rest of the paper, we assume that our tables are processed up to segmenting which means we can refer to individual cells, rows and columns.

3 Related Work

In this section, we discuss previous work on table understanding techniques and systems, and summarise how our work extends existing approaches.

As noted above, end-to-end table processing involves *extraction*, *understanding* and *interpretation*, following the approach first outlined in [3]. *Table Extraction* aims to locate tables in the document and then segment them into cells, rows and columns. *Table Understanding* aims to discover the logical structure of tables, via functional analysis and structural analysis. *Table Interpretation* aims to map table data to a specific set of domain entities and relationships; it is thus dependent on an application domain.

The pipelined approach used in TEXUS [16] can be characterised via a sequence of models: physical, logical and semantic. The physical model describes the output of the *Table Extraction* process and defines the outer boundaries of the table and the boundaries of the cells within the table. The logical model describes the output of the *Table Understanding* process and defines the categories and paths which provide access to the data entries. The semantic model describes the output of the *Table Interpretation* process and gives a mapping from the labels and entries in the table to a specified set of domain entities and relationships.

There has been considerable research on the *Table Extraction* task [13, 20, ?]. The work in this paper assumes that table and cell boundaries have already been identified, and does not consider *Table Extraction* further. The *Table Interpretation* process has been less well-studied, but, since it is inherently domain-dependent and since our goal is to provide automated domain-independent processing of tables, we do not consider *Table Interpretation* further.

Our focus in this paper is on work that deals with the functional and structural analyses that form the core of the *Table Understanding* process. Most previous table extraction work has represented a table's logical structure in terms of the Abstract Table model of Wang [19]. Our view of *Table Understanding* is thus that it takes a segmented table, determines table categories, identifies the category hierarchy which determines the access paths to the data entries, and describes this in terms of an Abstract Table model.

A variety of approaches and techniques have been developed to transform a segmented table to an Abstract Table. The most cited work in automated *Table Understanding* comes from research groups led by George Nagy and David Embley. They started with a semi-automated approach where assistance from a human user is assumed during the category detection phase. This approach is used in many tools like WNT [7], TAT [14], and VeriClick [12]. To move one step further towards automation of this process, they developed a learning-based classification algorithm to detect the four *critical cells* in a table that help to distinguish the header/column cells (labels) from data cells (entries) [9]. In their recent work, they proposed an algorithmic solution to index table columns and rows aso that detection of the critical cells could be automated [18].

These systems and approaches they have three basic assumptions: (i) tables are well-formed, (ii) layout formatting is not relevant, (iii) column and row headers contain one label. *Well-formed* tables are those where the row headers must be to the left and aligned with their data entries, and column headers must be above and aligned with their corresponding data entries. Nagy et al. [12] deal with layout formatting by first transforming the table into 'CSV' format, which removes both fine-grained layout and formatting information (such as bold fonts). By assuming that each cell contains one label, they can use the same methods to determine categories and category hierarchies in both row and column headers.

In our proposed system we also consider well-formed multi-dimensional tables. However, we believe that the category hierarchy can be more accurately

identified if certain commonly-used conventions in the stub are treated specially (i.e. we do not treat the row header and column headers in exactly the same way). For example, formatting features within a single cell can be used by a table designer to indicate the category hierarchy.

Somewhat related work appears in Fang et. al. [5], where they aim to classify tables by considering their header types and layout complexity. They identified several types of table (e.g. 1,2,multi-dimensional, complex, long, folded), and employed machine learning techniques to classify tables by using the various header features. However, they did not attempt to extract categories or a category hierarchy. Seth et. al. [17] describe a taxonomy of table column headers and propose a model based on XY cutting to convert the geometric structure of the header to a representation equivalent to Wang notation. Their work is applicable only to column headers and they assume that the input tables are canonical which narrows down the accepted tables for processing.

While previous work has tended to focus on discovering the category hierarchy in the boxhead, this paper considers the problem of discovering the category hierarchy in the stub. While some existing approaches, adapted from boxhead processing, are applicable, properties specific to the stub cells allow us to develop new approaches to determining the stub's logical structure. This extends the range of tables which can be automatically mapped to the Wang Abstract Table model.

4 Stub Analysis

The stub and boxhead regions contain multiple categories. We see that the choices people make to arrange the labels in those regions fall into two arrangement types: structural organisation of cells and layout styling.

In structural organisation, typically a combination of spanning cells and empty cells are used to encode the categories. Layout styling involves formatting of labels in individual cells such as use of bullet points, indentation, or font styling.

The boxhead region tends to use the structural organisation whereas in the stub, one can observe both structural organisation and layout styling. As we pointed out earlier, much of the existing work focuses on the use of structural organisation to determine categories.

We argue that consideration of both structural organisation and layout styling in the stub region can lead to more accurate analysis of the categories.

In this section, we present *Stub Analysis*. The aim of the analysis is to first determine top-level categories and all sub-categories and labels contained within each top-level category. As part of the analysis, we introduce the idea of *arrangement patterns* for the arrangement types that appear in the stub region. Through each pattern, we identify how the characteristics in the pattern can be used to detect the categories and labels in the stub.

We define the patterns, we draw on from the work of Fang [5] and Wang in [19] and our own observations of public datasets available. For example, Wang

introduced some layout styling rules for the stub, and Fang investigated different structural organisations that are commonly applied to arranging labels.

In the following, we present three main patterns: (i) *Layout Styling Patterns*, (ii) *Structural Organisation Patterns* and (iii) *Content Patterns*. By referring to these patterns, we can define a set of algorithms to detect the top-level categories and their sub-categories and labels.

4.1 Layout Styling Patterns

In layout styling patterns, we look for formatting styles in the cell content in the stub region. Such formatting styles could be indentation, listing or font formatting. The formatting is used to encode categories. There are two sub patterns.

Indented Stub: In this pattern, we observe the indentation style such as whitespaces, tab or bullets/numbering in labels. We refer to the labels with indentation or bullets/numbering as *indented labels*, and the labels without indentation are *natural labels* for this pattern.

Figure 3 shows examples of whitespaces and bullets. Taking the first table in Figure 3 as an example, `Effectiveness` is a natural label and `Employment` and `Further Studies` are indented labels.

Indicators	Weight of indicator in 2006
Effectiveness	
Employment	40
Further Studies	15
Processes	
Dropping out	15
Ratio of Qualification	13

(Indented Stub: Whitespaces)

Indicators	January 2015
Imports (RM million)	
• Capital goods	4,027
• Intermediate goods	13,646
Money Supply (%p.a.)	
• M1	-3.4
• M3	16.1

(Indented Stub: Bullet)

Fig. 3. Layout Styling Pattern: Indented Stub

when we find this pattern, we consider each natural label as a category. The indented labels of each natural label becomes the labels of the category. That is, in the first table of Figure 3, `Effectiveness` is a category and `Employment` and `Further Studies` are its labels.

Formatted Font Stub: In this pattern, we look for formatting features in fonts (e.g., changes and modifications of font appearances - bold, italic, larger size, underlines, colour) in labels. We refer to the labels as *formatted labels* and *unformatted labels* for this pattern. Figure 4 shows an example of this case.

The mapping of the detected labels to categories works the same way as for the indented stub pattern.

	2005	2004
Amount in Accordance with AGAAP		
Interest Income	15,113	12,939
Interest Expense	9,868	8,184
Profit From Ordinary Activities		
Income tax Expense	4,150	3,492
Net Profit		
Managed Investment Schema	2,579	2,928

Fig. 4. Layout Styling Pattern: Formatted Font Stub

4.2 Structural Organisation Patterns

In structural organisation patterns, we observe the merging or grouping of cells as an indication of potential categories. They can be summarised in two main groups:

Forward Expanding Stub: In this pattern, we look for the existence of spanning cells in the stub region. We name this pattern 'forward expanding' because the labels in one column span the labels in the adjacent column to the right. We refer to the identified labels as *spanning labels* and *covered labels*. Taking the first table in Figure 5, for example, `California` is a spanning label for `San Diego` and `Los Angeles`. `Los Angeles` is a spanning label for `Malibu` and `Compton`. `Malibu` and `Compton` are covered labels.

When we find this pattern, starting from the left most column, we map the spanning labels from the first column to a category. Labels in the next column become the children of their spanning labels. This process continues until we reach a column where the labels are not spanning.

Forward Reduction Stub: In this pattern, we use the presence of empty cells as an indication of a spanning cell. We name it 'forward reduction' because we effectively reduce the cells containing a label followed by empty cells into a single spanning cell. Once they are reduced to a single spanning cell, this case becomes isomorphic to the forward expanding stub case. An example of this is shown in the second table of Figure 5.

4.3 Content Patterns

Besides the patterns originated from layout styling and structural organisation types, we also examine patterns that consider the content (i.e., the text values) of cells. In these *content patterns*, we observe the existence of repeated labels or a special character in a cell in the stub region. There are two sub patterns:

Cross-product Stub: In this pattern, the set of labels for a category is repeated for every label of another category. This repetition can occur either in adjacent columns (see Figure 6 (Cross-product Stub - Over Columns)) or in the cells of a single column (see Figure 6 (Cross-product Stub - in One Column)).

State	City	Town	POP
New York	Rensselaer	Troy	1
		Brunswick	2
	St. Lawrence	Potsdam	3
		Canton	4
California	San Diego	Coronado	5
		Del Mar	6
	Los Angeles	Malibu	7
		Compton	8

(Forward Expanding Stub)

Scale	Size (IN.)	Cross Sectional Area	Free Point Constant
1.000 ×	0.080	0.221	552
	0.087	0.239	598
	0.095	0.275	643
1.250 ×	0.102	0.351	820
	0.109	0.374	936

(Forward Reduction Stub)

Fig. 5. Structural Organisation Pattern

Treatment/Therapy		Suffered From	Followed Treatment
Allergy problems	Count	93	77
	Percent	18.8%	15.6%
Anxiety disorder	Count	81	29
	Percent	16.4%	5.9%
Asthma	Count	31	22
	Percent	6.3%	4.4%

(Cross-Product Stub - Over Columns)

Treatment/ Therapy	Suffered From	Followed Treatment
Allergy problems		
Count	93	77
Percent	18.8%	15.6%
Anxiety disorder		
Count	81	29
Percent	16.4%	5.9%
Asthma		
Count	31	22
Percent	6.3%	4.4%

(Cross-Product Stub - in One Column)

Fig. 6. Content Pattern: Cross-product Stub

If we find this pattern in a stub, we map the column containing the repeating labels as a new category. For example, Figure 6 (Cross-product Stub - Over Columns), we have a top-level category *Treatment/Therapy* and its labels `Allergy Problems`, `Anxiety disorder`, `Asthma`. The repeating labels `Count` and `Percent` become the labels of the new category. Since we do not have a label to denote the new category, we add a virtual top-level category *VirtualCat*.

Leading Label Stub: In this pattern, we observe the existence of special characters like ':', '=', ':-' at the end of a label. We call this label, a leading label for the pattern. Figure 7 shows an example where `Costs, Expenses and Other` and `Basic Earnings` are terminated by ':'.

When we find this pattern, we consider each leading label as a category. The following labels of each leading label becomes the labels of the category. That is, in Figure 7, we have two categories *Costs, Expenses and Other* and *Basic Earnings*. The category *Costs, Expenses and Other* has labels `Materials and`

	2004	2003
Costs, Expenses and Other:		
Materials and production	4,959.8	4,436.9
Marketing and administrative	7,346.3	6,394.9
Research and development	4,010.2	3,279.9
Basic Earnings:		
Continuing Operations	7,974.5	9,051.6
Discontinued Operations	2,161.1	2,462.0

Fig. 7. Content Pattern: Leading label Stub

`Production`, `Marketing and administrative` and `Research and development`.

5 Implementation

In this section, we present the overall design of TEXUS and the implementation of the system, including the functional and structural analysis.

5.1 Overall Design

In order to facilitate a systematic development and reuse of the concepts and their implementation defined in our table processing system (TEXUS), we defined the table processing tasks as a set of components.

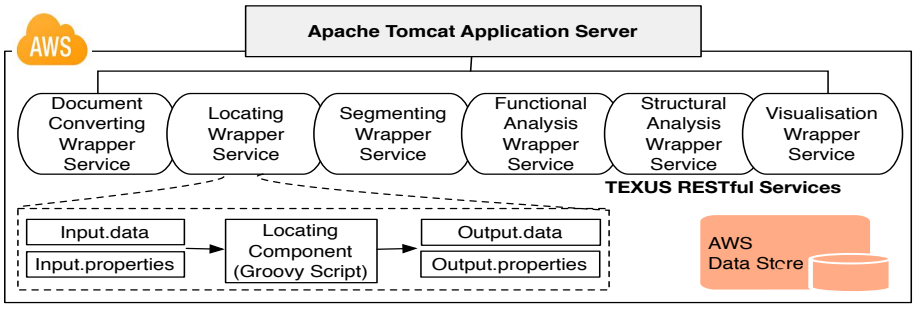

Fig. 8. Implementing TEXUS as Web-based Service Component

Figure 8 shows our system implementation. We have implemented each task as a Web-enabled service component which can be executed individually or any combination defined by user. For example, providing a segmented table as the

input the sequence of functional and structural analysis can be executed as a separate table understanding system.

The figure also illustrates that the 'body' part of each component (i.e., implementation of the task) takes two sources of data: an input data model instance and a set of configuration parameters. The outputs are an output data model instance and a set of properties (e.g., header type or error codes if any). To store data model instances, we have chosen to use XML, because (i) it is suitable for describing structured textual information, (ii) it is a platform-independent open standard, and (iii) it is easily transformable into different formats when necessary. For example, we can visualise the output of any components using a simple XSLT[1] script.

5.2 Table Extraction Sub-System

Receiving a PDF document as the input, the Document Converting component converts it to an XML document using a well-known conversion utility named '`pdftohtml`[2]'. This utility partitions the document into a sequence of `<page>`s, where each page is a sequence of `<textChunk>`s, and each `<textChunk>` is an XML `TEXT` element with the following attributes: `top` (vertical distance from top of page); `left` (horizontal distance from left edge of page); `width` (width of text chunk); `height` (height of text chunk); `font` (size, family, and color of text chunk).

We then perform further optimisation on the XML before producing output. First, we tag possible table cells containing multi-lines. Second, recognising multiple page columns (to be distinguished from table columns later). Third, tag text lines, properly recognising lines in each page column. As the system processes the document and locates and segments tables in the document, the same object model will be defined for detected tables.

Taking an instance of TEXUS document model as input, Locating component first attempts to separate *table lines* from *normal text lines*. It looks for lines with more than one text chunk as a potential table line. It then uses the transitions from text lines to (potential) table lines, and vice versa, to determine table boundaries. For potential table lines, our implementation also looks for type patterns (numeric, alphabetic, date, etc.). When a sequence of text chunk categories forms a pattern for that line. A sequence of lines that follows the same, or a similar, type pattern, is a strong candidate for containing the data cells of a table.

We also look for spatial patterns. The aim is to determine the left and right boundaries of each table column. Since the text chunks in a column are unlikely to have identical `left` and `right` attributes, we form column extents by considering the left boundaries of chunks in column i and the right boundaries of chunks in column $i + 1$. The sequence of chunk extents determined in this way

[1] XSLT, `http://www.w3.org/TR/xslt`
[2] pdftohtml, `sourceforge.net/projects/pdftohtml/`

forms a *spatial pattern* for the data in the table. The Locating component outputs XML file which encloses table lines in `<table>` elements, and adds `pattern` attributes to existing `<line>` elements.

After that segmenting component detects the inner boundaries of the table as cells, rows and columns. First, we look for dominant table line pattern to determine table rows, and then recognise lines that deviate from the pattern. These lines could be considered potential header lines or uncertain table lines (e.g., summary lines like 'Total'). Using the spatial pattern from Locating, starting from the table's dominant spatial pattern, we build a list of column horizontal boundaries, then scans the table and checks cell boundaries against these, allowing it to both detect spanned cells and, by consider each cell's `top` and `bottom`, to determine the vertical extent of the column. Finally, we determine table cells. In a table row, each text chunk has boundaries and content data type. Most table cells are clearly delimited from surrounding cells, but we handle two special cases: (i) *span cells*: if the extent of a single text chunk extends across multiple columns, it is labelled as a span cell, (ii) *blank cells*: once rows and columns are determined, the boundaries of cells are known. we detect whether and expected cell location has no corresponding text chunk and labels it as a blank cell.

Segmenting produces an XML file which encloses detected cells in `<td>...</td>` and places the text chunks from each table row in `<tr>..</tr>`. This becomes the input to the next sub-system *Table Understanding*.

5.3 Table Understanding Sub-System

Regardless of the source of the input (i.e., by our own Table Extraction subsystem or an external system), we assume that the following information is accessible from the input:

- coordinates of the cells, rows and columns,
- content of the individual cells in the table,
- spanned and merged cells,
- formatting styles of cells (e.g., alignment, font face, font color)

We represent a table T as a DOM[3] object and considers it to be a special case of a directed acyclic graph $D =< N, \varepsilon, r >$ where $N = N_t \cup N_e$ with N_t representing the set of text nodes (table cells), N_e the set of element nodes, $\varepsilon \subset N_t \times N_e$ the directed edges between the nodes, and $r \in N_e$ the designated root node. Each text node $n_t \in N_t$ has a string attribute s and each element node $n_e \in N_e$ a set of attributes A, where each $a \in A$ is a name-value pair. Figure9 shows a portion of the output of segmenting for the table in Figure 6 (Cross-product stub - Over Columns). The attribute *column* and $RowID$ are used to detect the direction and edges in the graph.

In Functional Analysis component, the goal is to detect each cell function as *entry* or *label*. Content types and spatial patterns of the cells are considered

[3] Document Object Model

```
<table id="0" >
    <fontspec id="0" size="13"  bold="yes" family="Times" color="#000000" />
    <fontspec id="1" size="13"  family="Times" color="#000000" />
    <tr left="135" top="115" bottom="130" RowID="0">
      <td top="115" left="135" width="141" height="15" font="0" column="01" colspan="2">Treatment/Therapy</td>
      <td top="115" left="345" width="108" height="15" font="0" column="2">Suffered From</td>
      <td top="115" left="483" width="148" height="15" font="0" column="3">Followed Treatment</td>
    </tr>
    <tr left="135" top="147" bottom="162" RowID="1">
      <td top="147" left="135" width="118" height="15" font="1" column="0" rowspan="2">Allergy problems</td>
      <td top="147" left="271" width="42" height="15" font="1"  column="1">Count</td>
      <td top="147" left="345" width="18" height="15" font="1"  column="2">93</td>
      <td top="147" left="483" width="18" height="15" font="1"  column="3">77</td>
    </tr>
```

Fig. 9. Representation of a Segmented Table (First Table in Fig. 6)

for detecting the boundary of the table body containing entries and the table boxhead enclosing the labels.

Based on our table model, $\forall n_t \in N_t, n_e \in N_e,\ A = A \cup (ContentType = "n_t.Type")$ in which $Type$ is one of the following content types: numeric, alphabetic, alphanumeric, percent, date, currency, plain text and blank cell. Then a *type pattern*, which is the concatenation of all patterns found in the cells of a row, is assigned to each row. First, we assume the bottom right corner cell in the table body boundary is an entry. Then its neighbour cells are compared against the spatial and content type patterns to determine if the cell is an entry cell or label cell. At the same time, the top most row is assumed to be a *potential* label line and the structural organisation of cells (e.g., spanning cells) along with the spatial and type patterns help detect the similarity of the neighbouring cells. The final meeting point of these two 'scanning' processes is considered the boxhead separator.

For the stub, we start from the left most column which is always assumed to contain *labels*. If we encounter an empty cell in the left most column or any vertical spanned cell in that column, we consider the following column for the label functionality class as well. We continue to add columns to the stub until we reach to a column where the number of non-empty cells is equal to the maximum number of non-empty cells in the table columns. The functional analysis component adds `function` attribute to each `<td>` element of a table row `<tr>`.

In Structural Analysis, we detect any stub pattern that may exist and identify the top-level categories, labels and access paths. Then an XML presentation of Abstract Table will be created for each table. We present the processing logic of the Structural Analysis component as follows:

Pattern Identification: In this step two types of cells in the stub are recognised (if any). *Enclosing* cells which are the potential categories and *Enclosed* cells which are the related labels in that category. Since there can a combination of patterns in one stub, there should be prioritisation between the patterns in identifying the enclosing and enclosed cells. We give more weight to structural organisation patterns since they are more common and an obvious way of con-

veying some hierarchical information. Content patterns are in the second level. Layout styling Patterns have the least priority since this kind of formatting may be used just for emphasizing on certain part of the table or cell information.

Table Normalisation:The enclosed cells get affixed by the content of their related enclosing cell for all recognised patterns. The equivalent normalised table, for the first table presented in Figure 3, is shown in Figure 10. `Employment` and `Further Studies` are two enclosed cells which are affixed by `Effectiveness` as their related enclosing cell and the same logic applies for `Dropping out` and `Ratio of Qualification` which are affixed by `Processes`.

Indicators	Weight of indicator in 2006
Effectiveness||	
Effectiveness||Employment	40
Effectiveness||Further Studies	15
Processes||	
Processes||Dropping out	15
Processes||Ratio of Qualification	13

Fig. 10. Normalised Table for Table in Fig. 3

Hierarchy Tree Formation: Starting from the right most column in the stub, we build a tree by considering all enclosed cells as the children and add the affixed part as the parent in the tree. We continue to reach to the left most column in the stub . The potential root for the tree is the stubhead content (if there is any) otherwise we add a virtual root for the tree. Figure 11 shows the hierarchy tree for the table in Figure 3.

Category Detection: After building the hierarchy tree, we analyse it to detect the top-level categories. Potentially the root of the tree is a top-level category and the paths to the leaves (labels) are its related sub-categories, unless there is a repeated sub-tree which can be an indication of existence of another top-level category. Therefore, we analyse the tree level-by-level to see if there is a repetition. Figure 12 shows the category detection for the tables presented in Figure 6. If there is a label in the stubhead for the related category it will be considered as the label for the top-level category, otherwise we use a virtual header for the category, which is the case for the tree in Figure 12.

The labels arrangement in the boxhead mostly follow the structural organisation patterns or cross-product over several rows. The same logic for detecting enclosing and enclosed cells, category tree formation and analysis is applied. Figure 13 shows the output of our structural analysis for the first row of the table in the Figure 6. This table has three main top-level categories, one with

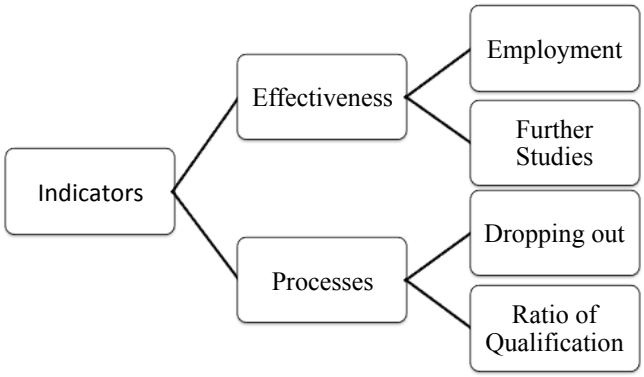

Fig. 11. Hierarchy Tree for Stub (Table in Fig. 3)

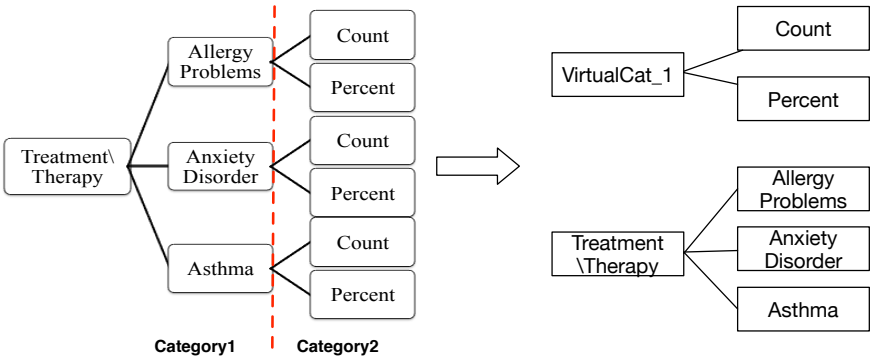

Fig. 12. Stub Category Detection (Tables in Fig. 6)

explicit label for that `Treatment/Therapy` and we add two virtual labels for the two other top-level categories.

Abstract Table Formation: The access paths generated in structural analysis and the recognised top-level categories along with their sub-categories and labels forms the abstract table representation for the table. The metadata containing some descriptive information (e.g, document type, page numbers, table caption), and structural information (i.e., categories and labels) are added to the Abstract Table representation. If any information that is not included in the final output of the analysis as a result of transforming tables to Abstract Table, we keep it as part of metada. For example, in the second table of Figure 5, the labels `City` and `Town` do not appear in the access path in the final abstract table as they are considered sub-categories of *State*. Therefore, we report these two labels in the abstract table metadata part. The final output is presented

```
<table id="0" pattern="7711">
    <tr left="135" top="115" bottom="130"  pattern="777" RowID="0">
      <td top="115" left="135" width="141" height="15" font="0" textType="7" column="01" colspan="2"
                    function="LABEL">Treatment/Therapy</td>
      <td top="115" left="345" width="108" height="15" font="0" textType="7" column="2"
                    function="LABEL">Suffered From</td>
      <td top="115" left="483" width="148" height="15" font="0" textType="7" column="3"
                    function="LABEL">Followed Treatment</td>
    </tr>
    <tr left="135" top="147" bottom="162" status="tableLine" pattern="7711" RowID="1">
      <td top="147" left="135" width="118" height="15" font="1" textType="7" column="0" rowspan="2"
                    function="LABEL">Allergy problems</td>
      <td top="147" left="271" width="42" height="15" font="1" textType="7" column="1"
                    function="LABEL">Count</td>
      <td top="147" left="345" width="18" height="15" font="1" textType="1" column="2" function="ENTRY"
          AccessPaths="Treatment/Therapy||Allergyproblems;;VC1||Count" ;; "VC2||Suffered From">93</td>
      <td top="147" left="483" width="18" height="15" font="1" textType="1" column="3" function="ENTRY"
          AccessPaths="Treatment/Therapy||Allergy problems;;VC1||Count" ;; "VC2||Followed Treatment">77</td>
    </tr>
```

Fig. 13. Structural Analysis Output (First Table in Fig. 6)

to user both in XML and HTML format through our visualisation component. The body, boxhead and stub are color-coded for better presentation and a tree structure representation of the categories are provided.

6 Evaluation

In this section, we report on our system performance of table understanding implementation. First, we explain the characteristics of the dataset chosen, then we present the results. We evaluate our system in three ways:

- compare our system result with Nagy's group latest result [18] to compare on the DocLab dataset which they used for evaluation.
- We ground-truthed ICDAR and PDF-Trex and provide our system result for each stub pattern.
- we transform our abstract tables with detailed category hierarchy to a data cubes and shows how having more accurate and detailed category hierarchy result in more accurate querying by providing more detailed aggregations and hierarchical level of dimensions.

6.1 Datasets

To evaluate our system, we selected three publicly available datasets known to table processing research community. The first one is the ICDAR competition dataset [4] which contains 67 PDF documents with 156 tables. The second one is PDF-TREX [5] which is a collection consisting of 100 PDF documents with

[4] http://www.tamirhassan.com/dataset/
[5] http://staff.icar.cnr.it/ruffolo/files/PDF-TREX-Dataset.zip

164 tables, written in Italian and English languages. The third dataset in DocLab Table Dataset [6] contains 165 documents and 172 tables. Since none of the datasets is ground-truthed for table understanding purpose[7], we have manually created ground-truthed datasets based on three human judges. The human judges were asked to nominate the categories, and the total number of distinct top-level categories in each table.

Table 1. Proportion of Table Types in the Dataset

Table Type	PDF-Trex	ICDAR	DocLab
2D	143	124	122
MD	21	32	50
One-column Stub	136	139	
Multi-Column Stub	28	17	

Table 1 shows the distribution of the tables in each dataset based on the number of columns in the stub and their number of categories. As can be seen, most of the tables have only one column in the stub which justifies our argument that a detailed analysis on the stub (e.g., consideration of the layout styling or content patterns in the column) could lead to more accurate table understanding. The two-dimensional tables have two top-level categories (one each from the boxhead and stub), whereas the multi-dimensional tables can have more than two. The indented stub (using whitespace, bullet and numbering) is the most common pattern in our dataset, followed by the leading label stub pattern. In multi-column stubs, forward expanding pattern is the most common. The cross-product stub patterns mostly occur in one-column stubs. Figure 14 shows the distribution of the patterns. Table may have more than one pattern in the stub.

6.2 Results

We report on the system performance in two ways: (i) table understanding system (the sequence of functional and structural analysis) (ii) structural analysis service to present how correct we detect the patterns on the tables with known stub and boxhead separators.

The performance of our functional analysis can be summarised as follow:

- 96% correct boxhead and stub separator (309 tables/320 tables)
- The unsuccessful cases were:
  - 2 folded tables (i.e., when the stub itself is repeated in the table),
  - 7 tables with repeated header rows in the middle of the table,

[6] http://www.iapr-tc11.org/dataset/DocLab_ Table_ Dataset_ Tables.zip

[7] The ICDAR dataset is ground-truthed for table extraction (locating and segmenting) only.

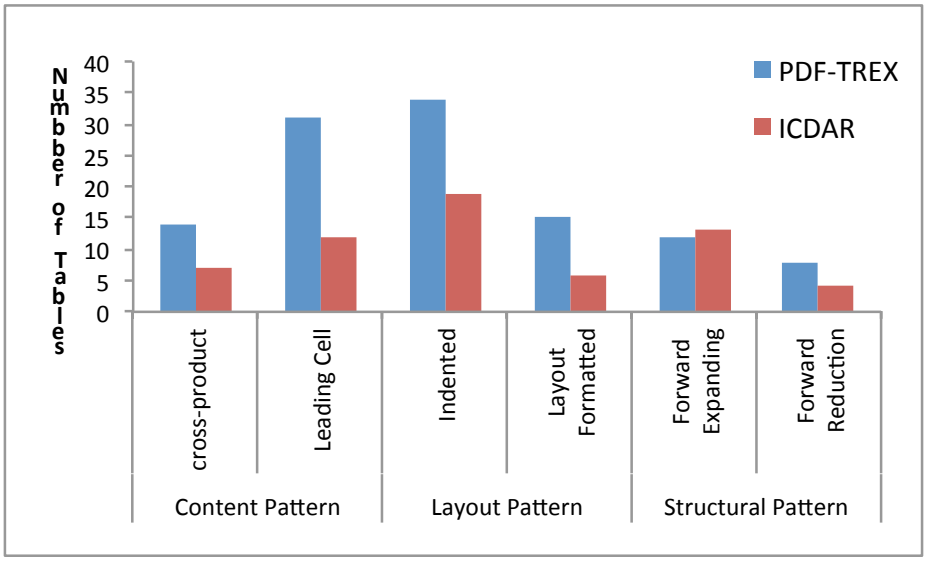

Fig. 14. Distribution of Stub Patterns in the Dataset

- one long table extending in two pages,
- one table with vertical text direction in the header rows (we only process horizontal text direction)

The result of structural analysis in detecting the stub patterns is shown in Table 2. There were tables which have a summary row (e.g. total value, sum) at the last line of the table which aligned with the cells matched in the cross-product stub. This resulted in our algorithm not recognising the cross-product stub patterns. We can improve our algorithm by considering the probability of the existence of summary rows at the first or last row of table. There were also cases in which the cells in the stub had different layout formatting just for styling purpose and did not carry any meaning of the hierarchy in a category.

Table 2. Result of the Stub Pattern Detection

Pattern	**Recall**	**Precision**	**F-Meas.**
Cross-Product	0.853	0.934	0.891
Leading Cell	0.936	0.976	0.955
Indented	0.962	0.891	0.925
Layout Formatted	0.952	0.904	0.927
Forward Expanding	0.96	1	0.979
Forward Reduction	1	0.923	0.959

Table 3 shows the result of detecting the categories and the hierarchies within them. We have created an XML representation of the *category trees* for each table in the ground-truthed dataset. We compared the category trees obtained from our algorithms against the ground-truth category trees. The detection is recognised as successful, if the category trees (in XML files) are totally matched.

Table 3. Result of Top-level Categories and Category Hierarchy Detection

	Top-level Category	**Category Hierarchy**
2D	0.974	0.913
MD	0.927	0.892

7 Conclusion and Future Work

We presented an analysis on the stub region of tables and defined three main patterns based on the arrangement types of cells. The aim of this analysis is to provide a basis for improving the table understanding tasks without relying on any external knowledge source. As far as we know, most of the work so far focused on table headers in the boxhead region and we are the first one reporting on the stub analysis. We implemented a system for a table understanding purpose utilising the identified patterns. We reported on the system performance that it has overall more than 90% accuracy in detecting the pattern and category hierarchy detection. The output of the system can be used for further table analysis (such as in semantic interpretation applications) and better similarity detection.

References

1. Norah Alrayes and Wo-Shun Luk. *Automatic transformation of multi-dimensional web tables into data cubes*. Springer, 2012.
2. Bertrand Coüasnon and Aurélie Lemaitre. Recognition of tables and forms. *Handbook of Document Image Processing and Recognition*, pages 647–677, 2014.
3. Ana Costa e Silva, Alípio Jorge, and Luís Torgo. Design of an end-to-end method to extract information from tables. *International Journal of Document Analysis and Recognition*, 8(2-3):144–171, 2006.
4. David W Embley, Matthew Hurst, Daniel Lopresti, and George Nagy. Table-processing paradigms: a research survey. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):66–86, 2006.
5. Jing Fang, Prasenjit Mitra, Zhi Tang, and C Lee Giles. Table header detection and classification. In *AAAI*, 2012.
6. Ramana C Jandhyala, Mukkai Krishnamoorthy, George Nagy, Raghav Padmanabhan, Sharad Seth, and William Silversmith. From tessellations to table interpretation. In *Intelligent Computer Mathematics*, pages 422–437. Springer, 2009.

7. Piyushee Jha and George Nagy. Wang notation tool: Layout independent representation of tables. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.
8. Shah Khusro, Asima Latif, and Irfan Ullah. On methods and tools of table detection, extraction and annotation in pdf documents. *Journal of Information Science*, page 0165551514551903, 2014.
9. George Nagy. Learning the characteristics of critical cells from web tables. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 1554–1557. IEEE, 2012.
10. George Nagy, David W Embley, Mukkai Krishnamoorthy, and Sharad Seth. Clustering header categories extracted from web tables. In *IS&T/SPIE Electronic Imaging*, pages 94020M–94020M. International Society for Optics and Photonics, 2015.
11. George Nagy, Sharad Seth, and David W Embley. End-to-end conversion of html tables for populating a relational database. In *Document Analysis Systems (DAS), 2014 11th IAPR International Workshop on*, pages 222–226. IEEE, 2014.
12. George Nagy and Mangesh Tamhankar. Vericlick: an efficient tool for table format verification. In *IS&T/SPIE Electronic Imaging*, pages 1–9. International Society for Optics and Photonics, 2012.
13. Ermelinda Oro and Massimo Ruffolo. PDF-TREX: An approach for recognizing and extracting tables from pdf documents. In *10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 906–910. IEEE, 2009.
14. Raghav Krishna Padmanabhan. *Table abstraction tool*. PhD thesis, Citeseer, 2009.
15. Raghav Krishna Padmanabhan, Ramana Chakradhar Jandhyala, Mukkai Krishnamoorthy, George Nagy, Sharad Seth, and William Silversmith. Interactive conversion of web tables. In *Graphics Recognition. Achievements, Challenges, and Evolution*, pages 25–36. Springer, 2010.
16. Details removed for double blind reviews. Details removed for double blind reviews. Technical report.
17. Sharad Seth, Ramana Jandhyala, Mukkai Krishnamoorthy, and George Nagy. Analysis and taxonomy of column header categories for web tables. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 81–88. ACM, 2010.
18. Sharad Seth and George Nagy. Segmenting tables via indexing of value cells by table headers. In *Document Analysis and Recognition (ICDAR), 2013 12th International Conference on*, pages 887–891. IEEE, 2013.
19. Xinxin Wang. *Tabular abstraction, editing, and formatting*. PhD thesis, University of Waterloo, 1996.
20. Xing Wei, Bruce Croft, and Andrew McCallum. Table extraction for answer retrieval. *Information retrieval*, 9(5):589–611, 2006.
21. Richard Zanibbi, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.