Automatic Tabular Data Extraction and Understanding

Roya Rastan

A thesis in fulfillment of the requirements for the degree of

Doctor of Philosophy

UNSW
AUSTRALIA

School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

January 2017

THE UNIVERSITY OF NEW SOUTH WALES
Thesis/Dissertation Sheet

Surname or Family name: **Rastan**

First name: **Roya** Other name/s:

Abreviation for degree as given in the University calendar: **PhD**

School: **School of Computer Science and Engineering** Faculty: **Faculty of Engineering**

Title: Automatic Tabular Data Extraction and Understanding

Abstract 350 words maximum

Tables in documents are a widely-available and rich source of information, but not yet well-utilised computationally because of the difficulty in automatically extracting their structure and data. This thesis focuses on the automatic processing of tables in unstructured documents with the aim to make their contents more widely usable.

Most existing work on table processing tackles only specific types of tables, or focus on sub-tasks of the complete table processing problem (e.g. locating tables). Importantly, most systems are designed as monolithic black-boxes, which makes it difficult to investigate their structure and performance, or to re-use/replace components to advance the state of the art.

The thesis offers a coherent and systematic view of table processing to provide flexibility and re-usability, and by widening the range of domains and formats that can be effectively processed. The thesis makes the following contributions: (i) introduces a task-based end-to-end approach for table processing, by providing a set of well-defined tasks, data models and interfaces between tasks. Ultimately, this provides a 'Workbench' for developing tools and techniques to further the state of the art in table processing, (ii) proposes a table-oriented document model blocks as a way to deal with a wide variety of document types and formats. The model encompasses the particular needs of the table processing systems to make their components more generic in design and implementation, (iii) describes an approach to automatic domain-independent table understanding by identifying patterns of layout, formatting and structural features which enable us to better extract information from tables than existing methods.

We evaluate the effectiveness and efficiency of our approach on well-known community datasets and compare it to existing table processing systems, based on an implementation of the approach developed through the thesis. Finally, we discuss a range of applications made possible by our approach.

Declaration relating to disposition of project thesis/dissertation

I hereby grant to the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or in part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all property rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstracts International (this is applicable to doctoral theses only).

5

The University recognises that there may be exceptional circumstances requiring restrictions on copying or conditions on use. Requests for restriction for a period of up to 2 years must be made in writing. Requests for a longer period of restriction may be considered in exceptional circumstances and require the approval of the Dean of Graduate Research.

FOR OFFICE USE ONLY	Date of completion of requirements for Award

Originality Statement

'I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, or substantial proportions of material which have been accepted for the award of any other degree or diploma at UNSW or any other educational institution, except where due acknowledgement is made in the thesis. Any contribution made to the research by others, with whom I have worked at UNSW or elsewhere, is explicitly acknowledged in the thesis. I also declare that the intellectual content of this thesis is the product of my own work, except to the extent that assistance from others in the project's design and conception or in style, presentation and linguistic expression is acknowledged.'

Roya Rastan
August 30, 2016

Copyright Statement

'I hereby grant the University of New South Wales or its agents the right to archive and to make available my thesis or dissertation in whole or part in the University libraries in all forms of media, now or here after known, subject to the provisions of the Copyright Act 1968. I retain all proprietary rights, such as patent rights. I also retain the right to use in future works (such as articles or books) all or part of this thesis or dissertation.

I also authorise University Microfilms to use the 350 word abstract of my thesis in Dissertation Abstract International (this is applicable to doctoral theses only).

I have either used no substantial portions of copyright material in my thesis or I have obtained permission to use copyright material; where permission has not been granted I have applied/will apply for a partial restriction of the digital copy of my thesis or dissertation.'

Roya Rastan
August 30, 2016

Authenticity Statement

'I certify that the Library deposit digital copy is a direct equivalent of the final officially approved version of my thesis. No emendation of content has occurred and if there are any minor variations in formatting, they are the result of the conversion to digital format.'

Roya Rastan
August 30, 2016

Abstract

Tables in documents are a widely-available and rich source of information, but not yet well-utilised computationally because of the difficulty in automatically extracting their structure and data. This thesis focuses on the automatic processing of tables in unstructured documents with the aim to make their contents more widely usable.

Most existing work on table processing tackles only specific types of tables, or focus on sub-tasks of the complete table processing problem (e.g. locating tables). Importantly, most systems are designed as monolithic black-boxes, which makes it difficult to investigate their structure and performance, or to re-use/replace components to advance the state of the art.

The thesis offers a coherent and systematic view of table processing to provide flexibility and re-usability, and by widening the range of domains and formats that can be effectively processed. The thesis makes the following contributions: (i) introduces a task-based end-to-end approach for table processing, by providing a set of well-defined tasks, data models and interfaces between tasks. Ultimately, this provides a 'Workbench' for developing tools and techniques to further the state of the art in table processing, (ii) proposes a table-oriented document model blocks as a way to deal with a wide variety of document types and formats. The model encompasses the particular needs of the table processing systems to make their components more generic in design and implementation, (iii) describes an approach to automatic domain-independent table understanding by identifying patterns of layout, formatting and structural features which enable us to better extract information from tables than existing methods.

We evaluate the effectiveness and efficiency of our approach on well-known community datasets and compare it to existing table processing systems, based on an implementation of the approach developed through the thesis. Finally, we discuss a range of applications made possible by our approach.

Publications

Parts of the thesis have been published in the following peer-reviewed publications:

- *Towards Generic Framework for Tabular Data Extraction and Management in Documents*, Roya Rastan, In Proc. of the Sixth Workshop on Ph.D. Students in Information and Knowledge Management (PIKM at CIKM), 2013, ACM, pp. 3–10
- *TEXUS: A Task-based Approach for Table Extraction and Understanding*, Roya Rastan, Hye-young Paik and John Shepherd, In Proc. the ACM Symposium on Document Engineering (DocEng), 2015, ACM, pp. 25–34
- *Automated Table Understanding Using Stub Patterns*, Roya Rastan, Hye-young Paik, John Shepherd and Armin Haller, In Proc. of 21st International Conference on Database Systems for Advanced Applications (DASFAA), 2016, Springer, pp. 533–548
- *A PDF Wrapper for Table Processing*, Roya Rastan, Hye-young Paik and John Shepherd, In Proc. the ACM Symposium on Document Engineering (DocEng), 2016, ACM, pp. 115–118

Also the main concepts and contributions of our table processing framework is presented in the following tutorial:

- *Table Modelling, Extraction and Processing*, Max Göbel, Tamir Hassan, Ermelinda Oro, Giorgio Orsi and Roya Rastan, In Proc. the ACM Symposium on Document Engineering (DocEng), Vienna, Austria, 2016, ACM, pp. 1–2

Acknowledgements

First and foremost I would like to express my special appreciations and thanks to my advisors Hye-Young (Helen) Paik and John Shepherd for their continuous support and encouragement in this project. I appreciate all their contributions of time and ideas to make my PhD experience productive and stimulating.

I want to specially thank Helen for the joy and enthusiasm she has for her research which was contagious and motivational for me, even during tough times in the PhD pursuit. It was Helen's patience, tolerance and guidance that kept me on track. I am thankful for the excellent example she has provided as a successful and caring research supervisor.

This research project cannot be completed without the vision of John. He guided me through all steps along the way from the very first research proposal to proof-reading my writings that led to the final draft of this dissertation. His openness in research and pursuing critical thinking will continue to have positive influence on my professional career in the years to come.

I would like to express my appreciation to the school of computer science and engineering of the University of New South Wales for providing financial support and office facilities during my PhD candidature.

Finally I want to thank my supportive mother for her endless love and encouragement. She raised me with a love of science and supported me in all my pursuits.

Abbreviations

AT: Abstract Table

CSV: Comma-Separated Values

DB: DataBase

GMTPS: General Model for Table Processing System

IE: Information Extraction

IR: Information Retrieval

NER: Named Entity Recognition

NLP: Natural Language Processing

OCR: Optical Character Recognition

OLAP: Online Analytical Processing

OWL: W3C Web Ontology Language

PDF: Portable Document Format

QA: Question Answering

RDF: Resource Description Framework

TDM: Table-oriented Document Model

TEXUS: Table EXtraction and Understanding System

XML: eXtensible Markup Language

Contents

List of Figures

List of Tables

Chapter 1

An Introduction To Table Processing

1.1 Introduction

Among the various structures that can be used for data summarisation, encapsulation and presentation in documents, tables are a common conceptual presentation structure. Tables contain a variety of data and information (e.g. words, digits, formulas, or images) and are embedded in a variety of document types (e.g. plain text, image, handwritten, or web pages). Because the information contained in tables is rich and potentially very useful, it would be convenient to have automated processes to identify and extract the information contained in tables in documents, which could then be stored in structured form for use in a variety of applications [Lon10].

Solving this problem is challenging, because of the complexity and diversity of layouts and encodings used in table presentation [Liu09]. Tables make use of layout

features and spatial arrangement to convey meaning, and can thus present and communicate complex information to human readers. Human readers are capable of using these layout features as cues for interpreting the logical meaning of the information contained in tables, but this rich combination of formatting and content is not so easily understood by a computer. In addition to layout diversity, tables occur in a variety of encodings in unstructured and semi-structured documents, (e.g. HTML, PDF, plain text). Because of its potential utility, table processing has gained attention from a number of research communities, such as document engineering, artificial intelligence, information extraction and retrieval, database and knowledge management and so on. This thesis aims to further the state of the art in this important area [eS10].

1.2 Research Questions and Objectives

Over the last two decades, various solutions and techniques have been proposed to meet the increasing demand for table extraction and understanding, and a number of "table processing" systems have been produced [EHLN06, SSKD10, eSJT06, GHOO12]. They typically adopt one of two general strategies to dealing with the wide variety of table structures: require manual intervention in recognising table structures and identifying data components [JN08, NT12], or develop specialised domain-dependent solutions that cannot be applied to other contexts [WMC09].

Table processing is a multi-faceted problem, but existing systems have generally approached the problem by considering just some aspects, or by partitioning the problem in an ad-hoc manner. This has led to a plethora of black-box systems, with low re-usability and little scope for a coherent approach to the table processing problem [GHOO13]. In addition, much work has focused on the sub-problem of table recognition and extraction, and ignored important problems such as distinguishing

between header cells and data cells or interpreting the content of the table [KLU14]. All of this has combined to somewhat stymie progress in the field, which would be better served by having a well-defined structure for the overall problem, into which new contributions and tools could be embedded [eSJT06].

Considering the drawbacks of existing systems and approaches, we believe that significant progress will only be made if any components that have been developed can be refined and integrated in a systematic way. This in turn requires defining a workflow around a set of essential table extraction and understanding components and setting up a framework for interoperability of these components. Therefore, in this thesis we look at the problem of automatic table processing from the process point of view and redefine the notion of "end-to-end table processing" to introduce more reusability, repeatability and extensibility to the automatic table processing systems.

This thesis intended to answer the following questions:

- What are the potential characteristics and application of an automatic table processing system?
- What are the atomic tasks involved in a generic, domain independent table processing system?
- How a table processing system can be designed to improve reusability and extensibility?
- how tables can be understood and interpreted in more domain-independent way?

1.3 Scope of this Research

Table processing is a large research area which can be looked at from a range of different perspectives: handling diverse table layout and design, dealing with the variety of source documents, or providing generic solutions for table understanding, to name a few.

The core of this dissertation is to provide a generic end-to-end table processing system which is automatic, domain-independent and which facilitates reusability and repeatability. Given the variety of document formats containing tables, we focus on PDF documents since PDF is a very common document format used in a wide variety of domains. This is not overly restrictive, since most semi-structured document formats (e.g. Word, HTML, LaTeX) can be converted to PDF, while preserving table layout structures.

We implement an end-to-end table processing pipeline which takes a PDF document as input and provides all contained tables in a generic representation as output. The pipeline consist of a number of modules, and each module has formally-defined inputs and outputs. While most existing work has focused on the modules we call locating and segmenting, this thesis expands the pipeline at both ends. First, we design a PDF Wrapper module to extract the useful elements from a PDF document to produce a document model that is suitable for table processing purposes. Second, to increase the generality of the system, we improve the table understanding module to perform more accurately without relying on specific domain information. And, in between, we provide our own modules for locating and segmenting.

1.4 Contributions

The goal of this research is to improve the design and development of automatic table processing systems by proposing an end-to-end table processing pipeline. We have the following specific contributions:

1. Introducing a Task-based End-to-End approach for building table processing systems by providing a set of well-defined tasks and concrete data models and interfaces. These provide a "workbench" for developing tools and techniques to further the state of the art in table processing.

2. Proposing a table-oriented document model and its building blocks as a way to handle the difficulties originating from the variety of document types and formats. The model encompasses the particular needs of the table processing systems to make their components more generic in design and implementation.

3. Proposing automatic domain-independent table understanding by identifying specific patterns of layout, formatting and structural features in tables which enable us to better understand the content of tables. This improves on previous work by recognising more kinds of layout structures and by not relying on domain-specific ontologies.

1.5 Organisation of the Thesis

Chapter 2 provides and extensive literature review which summarises the different approaches to table processing from the point of view of various research communities. The review helps us to understand the set of characteristics a generic table processing system should have.

Chapter 3 then provides a fine-grained, formal and precise definition of the primary tasks and elements which should be covered in an end-to-end table processing system. Based on that, we consider the problem from a software engineering perspective and propose a general architecture for table extraction and understanding systems. Our model includes a set of generic table extraction components and precisely defines the interfaces between them. One useful aspect of such an design is the flexibility it provides in developing components for specific kinds of extraction and to make the choice of an appropriate set of components during the table processing task.

Chapter 4 introduces a Table-oriented Document Model (TDM) which provides an effective input model for our pipeline and describes a PDF Wrapper component which converts PDF documents into the TDM. Chapter 5 describes our modules for locating and segmenting tables, which takes TDM as input and produces a highly accurate model of the table structures within a document. Chapter 6 shows how we can build the table understanding modules (functional analysis and structural analysis) in a domain-independent way by exploiting patterns in table headers and stubs.

Chapter 7 shows how our final table representation can be converted to other formats for data integration purpose, and also discusses how our standard definitions can help in refining of the some of current table processing challenges. Finally, Chapter 8 concludes the thesis and discusses prospects for future work in this area.

Chapter 2

Overview of Table Processing Research

Table processing has been studied for at least two decades and examined from diverse perspectives by different research communities. This chapter provides a survey of research related to table processing using the 4W+1H model explained in [JCYT15]. Using this approach, we create an extensive review of the field by considering different applications around table processing, the techniques used for table extraction and understanding, and the variety of input media exploited. We investigate different levels at which table research appears to operate and the different table models discussed in every level. We show how table processing research has evolved over time and underline the importance of end-to-end table processing system design. The goal is to enhance table processing research by defining a common set of reusable table processing components. In summarising the prior work, we discuss what elements a General Model for Table Processing System (GMTPS) should contain to facilitate end-to-end table processing and thus feed into applications based on the data derived from table processing.

2. Overview of Table Processing Research

We first review the definition of tables and describe how they are different from forms, lists, and other multicolumn text layouts. Then we describe different research approaches around tables from various communities to categorise the applications and needs of table processing systems.

Table processing is at the intersection of work from several research communities. This leads to some confusion when comparing research efforts from different points of view. To resolve this, we use a 4W+1H model to capture the essential details of table processing literature by focussing on six aspects:

1. Who: Involved research communities and their research groups
2. Why: Motivations and objectives of work on table processing
3. What: Research ideas and issues considered
4. Where: The context of the research problems, and venues of publication
5. How: Methods and techniques that have been applied

We have collated information about active table-processing research communities to determine the breadth of the field. We have used the motivation of the reviewed articles to help us to understand why the researchers carried out their work as they did. We have also identified the researchers' perspectives on tables and the end-goal applications for their table processing work.

Different research communities, with different aims and applications in mind, have contributed to table processing research to design and implement systems capable of capturing and manipulating tables. Table processing originated in the Image Processing community in pre-web history, with the goal of converting the bitmaps of scanned documents into machine readable formats to classify and cluster them and provide the ability to search and query on the document [KLU14]. Following

that, a body of research focussed on table editing and formatting to support different levels of table composition including both logical and layout structure [Wan96]. The more recent works on table processing concentrate on understanding the content of tables based on analysis of the table's logical structure, and on understanding the reading order of tabular data [Lon10].

In this section, we first review the work in each research community that has addressed the processing of tables, and then categorise the work according to specific applications. The goal is to follow the trends in the literature to understand what different applications of table processing have in common and what features that a comprehensive table processing system should consider. This helps us to clarify the notions of end-to-end table processing and generic table representation, with the goal to design a table processing framework that will support a wide range of current and future applications.

2.1 Variety in Table Processing

In order to devise better table processing approaches, we need to understand the nature of tables, their containing documents and their usage context. Tables can be presented in various layout styles and also in different document formats, with different internal encoding and structure. Documents and their embedded tables can contain information from diverse domains as well. In this section, we summarise factors which introduce variety in table processing and make it a challenging problem.

2.1.1 Variety in Table Styles

Tables are a relatively simple and widely-used data presentation strategy. However, they are also very flexible and a single data set could be represented in a wide variety of layout styles and formats. While such table structures are easily understood by human readers, automatically determining table structure and extracting and understanding the data contained within it is a significant challenge. Different factors contribute to the variety of table structures [FMTG12], and we summarise some of these here:

- Dimension: Tables are able to encapsulate multi-dimensional information although they are restricted to a two-dimensional presentation in the most common document formats. There are many ways to "render" multi-dimensional data Even when tables are presenting low-dimensional data, the header cells can be arranged in different ways. Figure 2.1 (from [HS10]) shows an example of two possible arrangements for a given data set.

Ranges	Investors	Securities
1- 1000	4381	2887161
1001- 5000	6311	1416542
5001- 10000	815	5900643

a. Header as the first row

Ranges	1- 1000	1001- 5000	5001- 10000
Investors	4381	6311	815
Securities	2887161	1416542	5900643

b. Header as the first column

Figure 2.1: Different arrangement of table headers

- Separators and Borders: Tables can be presented with or without explicit external borders; this has a significant impact on the process of locating tables. Similarly, row, column and cell separators may be absent, which directly impacts on the process of internal segmentation. Even when explicit separators

are used, different delimiter characters (e.g. punctuation characters, hyphen, underscore,dot, etc.) can be used. Variation in the use or style of separators increases the variety of table styles [tBC04]. Figure 2.2 shows a table that uses different characters as row delimiters.

(thousands of euro)	2005	2004
Operating activities		
Net income for the year attributable to the Parent Company and minority interests	114,104	108,343
Income taxes expense	20,288	27,663
Income before taxes	**134,392**	**136,006**

Figure 2.2: Inconsistent Row Separator Character in a Table

- Spanning Cells: A spanning cell can spread over multiple rows or columns. Spanning cells are primarily used to convey the hierarchical grouping of the underlying information. Detecting the scope of a spanning cell is a challenging issue in table segmenting; for example, in the absence of separators, it is ambiguous whether a spanning cell is a genuine spanning cell, or whether it is simply a range of regular cells, some of which are empty. Vertically spanned cells (occupying multiple contiguous rows) are especially problematic [LDC05]. Figure 2.3 shows a table with spanning cell in the first row (header).

	Capitale sociale	Riserva legale	Riserva straordinaria	Ris. esercizio	Totale
Alla chiusura dell'esercizio precedente	301.600	34.179	757.409	184.780	1.277.968
Destinazione del risultato dell'esercizio: - altre destinazioni		9.280	175.500	(184.780)	
Altre variazioni: - distribuzione dividendi			(275.000)		
Risultato dell'esercizio corrente				210.046	
Alla chiusura dell'esercizio corrente	301.600	43.459	657.909	210.046	1.213.014

Figure 2.3: Sample of Spanning cell in Header Row

- Multi-Line Cells: In some cases, cell content is text that spans more than one line. This makes both cell boundary detection and row detection difficult. For

example, it can be problematic to determine whether the cell content should be treated as a single data item or should be treated as separate vertically-aligned cells [Han00]. In Figure 2.4, it can be seen that the simply using the distance between test lines to determine rows can be problematic. For example the *RM million* should be merged with the text above it in all cases; however, text items with the same vertical separation in other parts of the table are separate data items.

Indicator / Month	2002											
	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec
Sales of new passenger cars (unit)	34,379	25,535	32,037	33,325	33,555	30,953	34,303	35,746	28,965	32,832	30,299	23,429
Production of vehicles (unit)	42,026	30,442	41,582	42,860	39,594	40,501	45,300	41,967	36,444	38,700	31,585	25,821
Sales of motorcycles (unit)	19,663	15,477	19,151	20,040	18,715	20,373	22,612	21,413	18,350	19,382	18,515	13,040
Production of motorcycles (unit)	19,377	14,587	18,074	19,899	15,751	20,110	24,124	22,469	19,580	19,580	20,043	16,395
Imports of consumption goods (RM million)	1,517	1,278	1,503	1,601	1,679	1,467	1,722	1,681	1,563	1,713	1,763	1,777
Kuala Lumpur Stock Exchange (end of period)												
- Composite Index	718.8	708.9	756.1	794.0	741.8	725.4	721.6	711.4	638.0	659.6	629.2	646.3
- Market capitalization (RM billion)	486	479	518	552	518	506	534	530	476	490	475	482
Prices (% p.a.)												
- Consumer Price Index	1.1	1.2	2.1	1.9	1.9	2.1	2.1	2.1	2.1	2.1	1.6	1.7
- Producer Price Index	-1.9	-1.3	0.6	1.4	3.1	3.6	3.8	4.5	6.8	9.9	10.3	13.4
Labour Market												
- Retrenchment (no.)	3,259	2,864	3,960	1,441	2,236	1,003	1,390	1,893	2,266	1,793	1,478	2,975
- Vacancies (no.)	11,094	7,868	14,015	12,039	10,256	9,019	16,121	15,335	19,461	15,501	17,222	14,856

Figure 2.4: Multi-Line Cell challenges in Table Segmenting

- Long and Folded Tables: Tables may be too tall or too wide to fit in a single page and then extend over the page or page column. A common approach to presenting this is to fold the data vertically or horizontally (with duplicated column/row headers) [HKL+01]. In these cases the duplicate header column/rows can be detected and any cells that are split over over a page or page column can be identified and merged. Figure 2.5 shows a time table as a typical example of a folded table where the table is split into logical segments, with each segment having a copy of the header (i.e. header rows repeat in the middle of the table).

- Floating Tables: Sometimes tables are presented as floating objects, and can

Da/from (BRI) Bari per/to	From	To	Days	Flt num.	Operator	Dep time	Destination	Arrival time
	12-mag	25-ott	1030507	**5057**	Myair	14.50	VENEZIA*	16.00
	30-mar	25-apr	1234567	**1087**		20.25	CATANIA	21.30
	27-apr	25-ott	1234507	**1087**		20.25	CATANIA	21.30
	30-mar	24-apr	1234567	**1053**		21.30	VERONA	22.50
	25-apr	25-ott	1234060	**1053**		21.30	VERONA	22.50
	25-apr	25-ott	0000507	**1053**		21.40	VERONA	21.00
	30-mar	25-ott	1234567	**1055**		09.00	VENEZIA	10.20
	30-mar	25-ott	1234560	**1051**		09.20	VERONA	10.40
	30-mar	25-ott	0000007	**1051**		09.50	VERONA	11.10
Da/from (BGY) Milano Orio Al Serio per/to	**From**	**To**	**Days**	**Flt num.**	**Operator**	**Dep time**	**Destination**	**Arrival time**
	12-mag	25-ott	1234567	**5060**	Myair	9.00	REGGIO CALABRIA*	10.25
Da/from (CAG) Cagliari per/to	**From**	**To**	**Days**	**Flt num.**	**Operator**	**Dep time**	**Destination**	**Arrival time**
	12-mag	25-ott	1030507	**5059**	Myair	13.40	VENEZIA*	14.55

Figure 2.5: Sample of Repeated Header in a Long Table

appear at the right-hand or left-hand edge of the page, with text "flowing" around them. These kind of tables effectively change the text reading order, and make it challenging to locate and segment tables. Most current systems erroneously treat the text wrapped around the floating table as the part of table. As shown in Figure 2.6 a floating table is surrounded with the other text in the document and detecting the flow of the table text is a challenging task [Anj01].

The majority of the enrolled students in the fall of 2006 attended larger colleges and universities. Specifically, campuses boasting enrollment levels of 10,000 students or more represented only 12 percent of the institutions; however, they enrolled 55 percent of all college students.[14] By comparison, 41 percent of the institutions had enrollment levels of less than 1,000 students, and these institutions enrolled only 4 percent of all college students.

Table 1: Student Enrollment, by Age Group, Fall 2006

Age	Enrollment	%
14-17	231,000	1.3
18-19	3,769,000	21.2
20-21	3,648,000	20.5
22-24	3,193,000	18.0

Figure 2.6: Sample of a Float Table

2.1.2 Variety in Table Content

Tables can be used to present many different types of data, from summarising numerical and statistical data to images and graphical symbols. This variety increases the difficulty of the table understanding task. Some tables, especially in the HTML context, as simply used as devices for specifying page layout. Meaningful tables (i.e. tables which aim to convey information about a set of data items) generally have

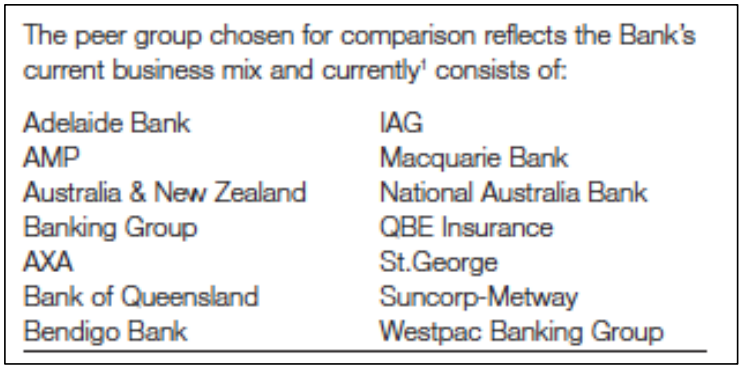
consistent data type patterns. This consistency can be exploited to improve the accuracy of row similarity detection and table locating [HS10]. Here we list some of the cases where particular table cell content increases the table processing difficulty.

- Graphics Symbols and Images: Most table processing and understanding algorithms rely on the font specification of the table content to determine the boundary of cells and then determine table segmentation. Since images and graphics symbols have different font specification in comparison to the usual text elements, tables which include such elements are more difficult to segment. Such elements also result in a higher level of difficulty in table content understanding and interpretation [HKL+01].

- Mathematical Notation/Equations: Differentiating between tables and equations is a recognised challenge in table processing [LBG11]. Table location and, especially, segmentation become more difficult when table cells contain equations. Another issue is that symbols in equation tend to be representated graphically rather than as text elements.

- Implicit Header Cells: Header cells play an important role in table understanding [FMTG12]. Sometimes the headers of some rows or columns in the table are omitted, with the idea that they can be inferred from the topic of the document or the surrounding text. These empty cells demands another level of text analysis to complete table understanding. Figure 2.7 shows an example of table without an explicit header row; in this case the header can be inferred from the surrounding text.

- Different Languages: Table content interpretation and, primarily, semantic analysis relies on understanding cell contents and the relationship between cell contents. Such understanding requires natural language processing techniques, and most language processing resources are for English and other left to right



The peer group chosen for comparison reflects the Bank's current business mix and currently[1] consists of:

Adelaide Bank	IAG
AMP	Macquarie Bank
Australia & New Zealand	National Australia Bank
Banking Group	QBE Insurance
AXA	St.George
Bank of Queensland	Suncorp-Metway
Bendigo Bank	Westpac Banking Group

Figure 2.7: A Table with Implicit Header Row

written languages.

2.1.3 Variety in Document Types

Tables are used in a variety of different document types, such as plain text, HTML, image and PDF. Every document format has its own internal structure and encoding which demands different algorithms for table detection and understanding. For the purpose of table processing, we consider various aspects of documents based on the depth of information they provide to access the document elements and objects.

- Object Information: Some of the document formats (e.g. HTML) provide an explicit indication of the logical and physical objects in the document. Physical objects are elements such as line, block, column, etc and logical objects are elements such as headings, figure captions, tables, chapters, etc. [GTL+11]. The existence of such information (tags in the case of HTML) makes the automatic detection and identification of tables easier (except when tables are used as a physical layout device rather than a logical structure). When such information is not provided by document format (e.g. text documents), some preprocessing is needed to extract this information or to convert the document to another format which is more amenable to processing (e.g. PDF

to HTML conversion). Such preprocessing is a potential source of errors which can impact the whole table processing process.

- Geometric Position Information: Some document formats provide a geometric position of the document elements which can be helpful for layout analysis in table processing. This positional information can be expressed either the absolute position of the element in the document (e.g. absolute position of each word segment in a page of a PDF document) or relative status of document elements towards each other (e.g. the position of the table caption and table tag in a HTML document). Each of these document types demands its own preprocessing method. For example for PDF documents (as a format which provides absolute positional information) word segments need to be assembled and ordered to build a correct reading order which is not always an easy task [eS10].

- Formatting Information: Formatting information is available in some document formats (e.g. formatting tags in HTML documents), and this information can be helpful in different steps of the table processing process [NSJ+11]. Some of the algorithms for table locating and segmenting rely heavily on heuristics based on formatting cues [PCS05]. However, some other document formats such as plain text do not contain any information about text formatting and this can make the preprocessing steps more difficult.

- Textual Information: When document formats provide access to text content in the document elements, it makes the use of bottom-up methodology for table analysis easier. It also facilitates the application of the language processing techniques for the purpose of the table understanding and semantic analysis. However, in document types such as images, it is difficult to access the textual information directly and the OCR processing required to extract text typically decreases the accuracy of the overall table analysis task [THYX02].

2.1.4 Variety in Domains

Tables are used in documents from a wide variety of domains. There is a tendency for documents in a particular domain (e.g. database research papers) to follow relatively similar styles and layouts, but the similarity is not reliable enough to be used as a basis for developing domain-specific table processing methods. Even within a single document, several table style may be used. However, domain-specific information, such as ontologies, can be usefully employed to improve the accuracy of tasks such as table understanding and semantic analysis. This has encouraged the development of domain-specific solutions for table processing which work well for the targeted domain, but do not adapt easily to other domains [GBH+07]. Here we summarise some of the issues in the table processing that have their origins in the documents being from different domains and contexts.

- Reusability: Many existing table processing systems have been developed which perform accurately only in very narrow domains. Since they rely heavily on linguistic techniques tuned to the domain of interest or domain-specific extraction patterns, they do not provide a general solution for table processing [Liu09].

- Generic Table Understanding: Most existing table understanding and interpretation solutions rely heavily on domain-dependent ontologies. The scope of a generic table understanding system is still open for debate and as is determining how to measure the accuracy of automated table understanding [LE08].

- Table Data Integration: Most existing table processing systems present the output of the processed tables in custom and domain-based representations. When it comes to applications based on table processing (e.g. finding similar

tables), some translation from the domain specific representation to a more generic one is typically required [TEL+05].

2.1.5 Variety in Applications

Table processing is looked at from different points of view and a range of potential applications. In addition, different research communities use the output of table processing at different levels of abstraction. This diversity in viewpoints and applications has led to various definitions of table processing which has made it difficult to set a standard for the processes involved, and thus allow for straightforward comparison of the various approaches. Here we summarise some of the main applications mentioned in the literature.

- Database Population: Transforming tables in documents to relational tables and using them to populate a relational database is one of the major applications of the table processing. DBMSs then provide a range of functionalities which can support querying and data merging. While tables in documents are constrained to a two-dimensional presentation, rows in relational DBMSs represent n-dimensional tuples, which makes them easier to analyse [NSE14].

- Data Integration: Data extracted from tables can be structured and then inserted into a database, despite starting from varying formats and schema. This can be an initial step for data mining which can potentially lead to the integration of data from widely different source documents, or the detection of similarity between tables from different sources [GHJ+10].

- Table Search: Searching for tables has been addressed in various document formats, such as searching within PDF documents in digital libraries [LBMG07c]

or finding tabular structures on the Web [DSFG+12]. Searching for tables inevitably leads to the notions of ranking discovered tables or building indexes for faster subsequent retrieval. Both of these suggest the need for a notion of table similarity.

- Table Editing and Formatting: This category of applications focuses on the editing and formatting of the logical structure and specification of table layout. This can be discussed at the level of table composition systems [Kah03] or used as a step in document layout analysis for accessing to the document logical elements [FSCG03].

- Information Extraction: Table extraction can be viewed as a special case of information extraction that deals with both locating tables in documents and with collecting the data presented in them. One example of a task associated with the latter would be Named Entity Recognition [WMC09], to assign semantics to the contents of particular cells. Another task might involve focusing on the table topology and trying to extract a set of relational tuples [GBH+07].

- Ontology Generation: This application exploits the fact that tables explicitly represent relationships among data items, for example by placing them in the same row, which can assist in ontology generation. Some applications of table processing in this area: ontology generation [LY10], ontology population [PCS+07] and ontology maintenance and change discovery [GLM03].

- Document Engineering: Table understanding is a major application among document engineering workers, especially those who work with unstructured documents. One use of table understanding is to produce a more structured and/or canonical representation of the data through reverse engineering [RBH+05] or to assist in converting documents form one format to another one [JY09].

2.2 Table Processing Terms and Concepts

There is a diversity in terminology amongst those who work on table processing. This often makes it difficult to compare works which are otherwise closely related. In order to organise our discussion of research works better, we review terminology, concepts and definitions from the table processing literature.

2.2.1 Table Definition

As noted previously, tables are a widely-used data summarisation representation used in documents from a wide variety of domains. Many works on table processing discuss detailed aspects of an approach to table analysis without addressing what a table actually is. Despite their wide usage, or perhaps because of it, there is no universally agreed formal definition for tables. Rather, tables are modelled from various viewpoints: geometry, hierarchies of values, or relational structures [Hur06]. While tables can be (and have been) defined from different perspectives, there are common features running through many of the definitions: the relations between tables elements [Hur06], the existence of the repeated content, or visual cues like borders and delimiters. Here are some of the table definitions from the literature:

- A table is comprised of a set of cells and relations between them to convey hierarchical concepts or categories [Hur06].
- Tables are objects that show logical connections between well-defined content entries which a content entry can be any visual symbol [eSJT06].
- Tables contain a regular and repetitive structure along at least one dimension in order for data type to be determined using either a horizontal or vertical index [LBMG07c].

- A table is a graphical grid of a matrix $M_{i,j}$, where (a) with atomic elements i, j (b) having linear visual clues such as elements in each row i tend to align horizontally where elements of a column j tend to align vertically and (c) with linear visual clues that depict some logical relationships [eS10].

Despite the definitions of tables in the literature being quite general in nature, most research work restricts table processing to *canonical* tables and considers a canonicalisation process on tables. A schema for a canonical table is a finite set $\{L_1, ..., L_n\}$ of labels. Each label L_i has an associated domain D_i. Let $D = D_1 \cup ...\cup D_n$. A canonical table T with table schema S is a set of functions $T = \{t_1, ..., t_m\}$ from S to D with the restriction that for each function $t \in T, t(L_i) \in D_i, 1 \leq i \leq n$ [TEL+05].

Tables are not the only way of presenting data in a structured format. Other common presentation strategies for structured data include lists, forms, vectors and matrices. When there is only one column, the table is more commonly called a list. If a label is missing, we may think of the label as being implicit. If all labels are missing and all domain values are numbers, we think of the table as a matrix. if a matrix has only one row or one column, we think of it as a vector [EHLN06].

When there is a column with all empty cells, and the table is not canonical, we can view the table as a form with slots to be filled in. If a label is missing, we may think of the label as being implicit. What differentiates tables from forms is that they have a regular, repetitive structure along at least one axis, so that the data type is determined by either the horizontal or the vertical index [LBMG07c]. Forms, on the other hand, show a one-to-one relationship represents between indices and data, in which there are no indication of the regularity [LN99].

Diagrams may be similarly differentiated from tables. Diagrams are not restricted to a two-dimensional grid in order to convey information, and visual cues (such as

boxes and lines) are more important for understanding the information rather than the content of the items and their relative position [eSJT06].

2.2.2 Table Dimensions

A table can be considered as multi-dimensional object which is presented as a two-dimensional structure within a document. Data in a table can be accessed by indexing over a set of categories, where each category corresponds to a dimension [Hur00a]. Tabular structures can be classified based on their dimensionality. Here we consider three main classes:

- One-Dimensional: In this class of tables, attribute cells and value (data) cells are typically arranged in a horizontal or vertical manner. Here, each attribute cell describes its underlying value cells, either in the same column or the same row. Value cells sitting in the same row or column are typically drawn from the same underlying domain. One-dimensional tables are mentioned with various names in the literature: [Yan02, SJKN10, HS10] call them *One-Dimensional Tables*, Norah [AL12] calls them *Relational Tables*, and they are also refereed to as *Flat Tables* in [WL06]. Figure 2.8 shows a one-dimensional table with four columns.

RANK	SHAREHOLDER NAME	NUMBER OF SHARES	PERCENT HOLDING
1	ANDREW TSANG & indirect interests *(Director)*	38,170,518	16.90%
2	DORAL MINERAL INDUSTRIES LIMITED	23,500,000	10.41%
3	HUNTLEY CUSTODIANS LIMITED	6,842,860	3.03%
4	GUOZHONG YU	5,133,333	2.27%
5	ZIRCON RESOURCES LIMITED	4,500,000	1.99%

Figure 2.8: A sample of One-Dimensional Table

- Two-dimensional: This class of tables has attribute cells (header cells) at the start of each column and each row. Each data cell in such a table represents a value associated with a specific pair of attributes. Besides *2-Dimensional Tables* [WWW+00, YL02, EHLN06], these tables are also refereed to as *Nested Tables* [WL06]. Figure 2.9 shows a 2-dimensional table which represents some grant options in the row headers and dates in the column headers.

	July 2003 – June 2004		July 2004 – June 2005	
Year of Grant	2000	2001	2000	2001
Total options:				
Held by participants at the start of the year	427,500	2,336,400	402,500	2,235,200
Granted during year	–	–	–	–
Exercised during year	–	–	155,000	403,900
Lapsed during year	–	101,200	–	29,700

Figure 2.9: A sample of Two-Dimensional Table

- Multi-dimensional: Tables in this class consist of three or more dimensions, and could, outside the context of a two-dimensional page, be mapped to a multidimensional array. Each cell in the multi-dimensional array contains a data item, called a measure, and corresponds to one data cell in the table. Data items in such tables are frequently numeric values. Data cells can be accessed via a combination of index (header) values, one for each dimension. Such a multidimensional array is generally called a data cube [AL12]. For example the information in the table shown in Figure 2.10 summarises the population of Canada [1] from the perspectives of marital status, sex and also by year.

Making an n-dimensional structure understandable to a human reader requires support from a table rendering system that provides a variety of layout features. Document formats such as HTML provide a means to explicitly specify the layout. Kahl

[1] http://www.statcan.gc.ca/tables-tableaux/sum-som/l01/cst01/famil01-eng.htm

Population by marital status and sex

	2003	2004	2005	2006	2007
	number of persons				
Total					
Both sexes	31,676,077	31,995,199	32,312,077	32,649,482	32,976,026
Male	15,688,977	15,846,832	16,003,804	16,170,723	16,332,277
Female	15,987,100	16,148,367	16,308,273	16,478,759	16,643,749
Single					
Both sexes	13,231,209	13,368,674	13,507,149	13,653,059	13,800,997
Male	7,078,089	7,155,622	7,233,428	7,314,611	7,396,835
Female	6,153,120	6,213,052	6,273,721	6,338,448	6,404,162
Married[1]					
Both sexes	15,438,972	15,558,054	15,675,089	15,802,300	15,916,860
Male	7,701,393	7,752,882	7,803,419	7,860,087	7,910,554
Female	7,737,579	7,805,172	7,871,670	7,942,213	8,006,306
Widowed					
Both sexes	1,532,940	1,544,226	1,553,488	1,563,856	1,573,455
Male	288,816	295,446	301,404	307,050	312,357
Female	1,244,124	1,248,780	1,252,084	1,256,806	1,261,098
Divorced					
Both sexes	1,472,956	1,524,245	1,576,351	1,630,267	1,684,714
Male	620,679	642,882	665,553	688,975	712,531
Female	852,277	881,363	910,798	941,292	972,183

Figure 2.10: A Multi-Dimensional Table

[Kah03], on the other hand, proposed a multi-dimensional table syntax which leads to a particular compositional view of table structure. In his view, an n-dimensional table has, as its central component, a grid, which is an n-dimensional array of cells. Each data cell can be addressed via a one-dimensional vector of n header cell values.

2.2.3 Table Models

Different table processing systems have been developed with different aims. For example, some aim to convert a scanned table into an editable Microsoft Excel or Word table, others aim to recover the grid and cell contents, while others aim to assign meaning to the data. To cater for these diverse needs, many different models of tables have been proposed. Each model aligns with the particular task addressed by the system or the particular philosophy of document encoding. Also, there are a number of different levels at which table processing can operate and this is reflected in further variety in table models. For example structural models are used for representing region and cell structures, while conceptual models enable the abstraction of content from presentation [PCS05]. A 2004 survey of different models [tBC04] for table recognition and table understanding provided observations on existing table models, and described inferences and transformations for tables [tBC04].

The most complete and generic table model in the literature so far is Wang's Abstract Table model [Wan96] which separates logical and structural table information from table layout characteristics. Wang's model presents table structure into three interrelated parts: an abstract indexing relation, a topology defining the arrangement and ordering of dimensions within the table headers, and formatting attributes which include fonts, separator types, and layout constraints.

Some background terminology is required in order to describe the Abstract Table model. Wang defines a *Labelled domain* recursively as being either a labelled empty set or a labelled set of labelled domains. A labelled domain can be represented by a tree of labels. Figure 2.11 demonstrates the relationship between a labelled domain D_1 and its tree. Every node in the tree of the labelled domain is called an *item* and is identified by the sequence of labels on the path from the root to the node.

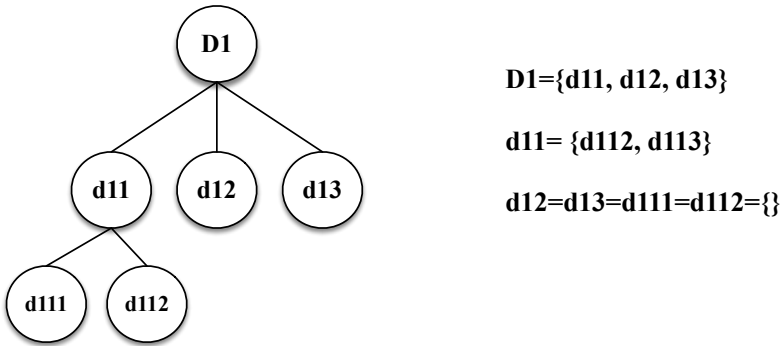


Figure 2.11: The Relationship Between a Labelled Domain and its Tree

For example, node D1.d11.d112 is an item in Figure 2.11. If i is an item of labelled domain D, we use P[i] to denote the parent item of i in the tree of D. A *frontier item* is an external node in the tree of a labelled domain. $F[D]$ is used to represent the set of all frontier items in a labelled domain D. $\otimes$ is an operator that takes a set of different labelled domains and returns a set in which each element is a set of n frontier items that originate from each of the n labelled domains. Therefore, it can be said that:

$\otimes\{D_1, ..., D_n\} = \{\{f_1, f_2, ..., f_n\} \mid f_1 \in F[D_1] \wedge f_2 \in F[D_2] \wedge, ..., \wedge f_n \in F[D_n]\}$. Note that the order of $D_1, ..., D_n$ is not critically important in $\otimes$.

An Abstract Table is formally defined by a tuple $(n, \{D_1, ..., D_n\}, E, \delta)$, where

- n is non-negative integer
- $D_1, ..., D_n$ are labelled domains.
- E is a set of entries.
- δ is a partial function from $\otimes\{D_1, ..., D_n\}$ onto E.

Using the formal definition of Abstract Table, table 2.12 can be abstracted by $(3, \{D1, D2, D3\}, E, \delta)$, where:

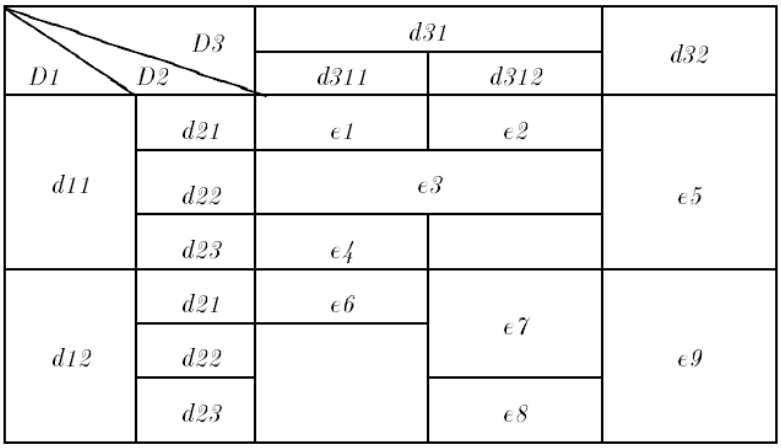


Figure 2.12: An Example of a Three-dimensional Table

$D1 = \{d11, d12\}$

$D2 = \{d21, d22, d23\}$

$D3 = \{d31, d32\}$

$d31 = \{d311, d312\}$

$d11 = d12 = d21 = d22 = d23 = d32 = d311 = d312 = \{\}$

$E = \{e1, e2, e3, e4, e5, e6, e7, e8, e9\}$

$\delta(\{D1.d11, D2.d21, D3.d31.d311\} = e1; \quad \delta(\{D1.d11, D2.d21, D3.d31.d312\} = e2;$

$\delta(\{D1.d11, D2.d22, D3.d31.d311\} = e3; \quad \delta(\{D1.d11, D2.d22, D3.d31.d312\} = e3;$

$\delta(\{D1.d11, D2.d23, D3.d31.d311\} = e4; \quad \delta(\{D1.d11, D2.d21, D3.d32\} = e5;$

$\delta(\{D1.d11, D2.d22, D3.d32\} = e5; \quad \delta(\{D1.d11, D2.d23, D3.d32\} = e5;$

$\delta(\{D1.d12, D2.d21, D3.d31.d311\} = e6; \quad \delta(\{D1.d12, D2.d21, D3.d31.d312\} = e7;$

$\delta(\{D1.d12, D2.d22, D3.d31.d312\} = e7; \quad \delta(\{D1.d12, D2.d23, D3.d31.d312\} = e8;$

$\delta(\{D1.d12, D2.d21, D3.d32\} = e9; \quad \delta(\{D1.d12, D2.d22, D3.d32\} = e9;$

$\delta(\{D1.d12, D2.d23, D3.d32\} = e9;$

2.2.4 Different Perspectives on Tables

Tables can be considered as a collection of relational data in a structured format which is reflected through table layout. Tables can provide information from three different perspectives [Lon10]:

- Table Layout Structure: which contains information about coordinates of the table elements and their related style and formatting attributes. This information can be provided at a very fine-grained level (e.g. token or character level for cell content) or at the table element level (e.g. table rows and columns coordinates). This perspective provides the elements for describing tables at *physical level* [Has10b]. Many document elements and their characteristics are reported in the literature to describe tables at the physical level. Here is a summary of such characteristics, which are mainly geometry based:
  - Distance between points: such as calculating height, width, top and left of elements, relative to the top-left corner of the page[YKM05]
  - Bounding box and region based parameters: segmenting pages into regions at different levels and defining bounding boxes for table elements [LCD06, GHOO12]
  - Density based parameters: such as pixel density for image documents [FSCG03] or count of characters in a specific region [WH02a] or whitespace density in particular horizontal or vertical position [HB05].
- Table Logical Structure: which contains information about the logical and structural relations between table elements and how their arrangement defines the reading order of the table. Based on the different element arrangements in tables, a logical structure can be mapped into various layout structures [HCHL07]. Tables can be defined at different logical levels. The most abstract

level of logical structure can be the indexing schema that relates table headers to table data [tBC04]. Logical tables are defined through various elements in the literature, including:

  - Composition of smaller regions/elements: for example cells to be arranged horizontally to form rows or vertically to specify the columns [eSJT06], or forming table grids [PCS+07].
  - Distance between elements: for example using tree edit distance to compare different data rows to find a table extraction region [TE09].
  - Table indexing structure: modelling tables to define row and column header paths uniquely to index each data cell in the table [SN13].

- Table Content and Semantics: which identifies the main content of the table data cells. Table content understanding can derive from shallow parsing of the content to assign a general concept to a cell, or from applying domain-based knowledge to table cells and annotating them semantically. This perspective on tables provides the main content for *Table Interpretation*. Semantic features of tables identified in the literature include:

  - Table ancillary information: Mainly used under the title of *Table Augmentation* or *Table Metadata* which is derived from features such as table title, caption, footnote or unit indicators [LMGB06, NPJ+10].
  - Named entity mapping: This perspective uses NLP and IE techniques to associate cells with pre-defined target semantic labels [LSC10].
  - Semantic tables: This approach considers the table logical structure and semantic knowledge bases at the same time and captures the intended meaning of the table by mapping header cells to classes, data cells to entities and pairs of columns to relations from the given ontology or knowledge base. This is also referred to as *Complete Interpretation* [Mul15].

2.2.5 Different Applications for Table Processing

A variety of applications for table processing are described in the literature. We consider the three main applications here:

- Editing and Formatting [tBC04]: Applications of this type focus on document layout analysis (either for composition or rendering purposes) in which tables are involved as one of the document objects. Table topological and style specifications are the main layout characteristics of interest. Table topology specifies the relative placement of tabular items in two dimension and style specification provides rules that lead the presentation of the table. The main goal for such applications is to enable manipulation for specific audience, such as audio access to tables [YSGH04] or table modification for display on different devices [NJ07].

- Data and Information management [KLU14]: Most of the applications originating from database and IR communities have the the goal of managing data and information embedded in tables more accurately. Some of the specific aims are question-answering, query by table, tabular browsing, table search, data integration, and information summarisation. Some work has concentrated on converting tables from one file format to another to increase the table data accessibility (e.g. from PDF to HTML or XML) [CPV13].

- Knowledge Discovery [LE08]: These applications focus on the content and logical structure of tables. Some specific applications are: relation discovery, attribute-value extraction, table interpretation, schema matching, and semantic search. The scale of knowledge extraction varies from focusing on one specific context [TE09] to open-domain and web-scale knowledge discovery [CP10]. Transforming tables to logical frames and mapping tables to ontologies are the basic processes involved in these kinds of applications [PCS05, LY10].

2.2.6 Input Media

Whatever the nature of the table processing tasks and applications, the starting point is dealing with input documents and finding table structures within them. Input media for table processing can be classified under the two main formats of 'electronic' and 'paper' [EHLN06]. The former can be further subdivided into *ASCII files* such as plain text which consist the linguistic content and character-level spacing and *page-descriptor files* such as Word, PDF and Latex which contain formatting attributes for the linguistic content. Since paper-based content does not lend itself to automatic processing, paper-based documents need to be scanned into *bitmap files* [JN08]for further processing.

Electronic documents can be considered as either structurally *tagged* (e.g. HTML) or *un-tagged* (e.g. plain text) [Liu09]. Tags potentially make it easier to locate tables. However, in the case of HTML, `<TABLE>` tags are often used for formatting convenience and detecting genuine tables becomes more challenging [VHM+11]. Most of the work in the table processing literature has focused on four main document formats:

- Plain Text Format: Plain text documents do not contain explicit (tagged) formatting information. However text documents use spacing and layout to convey structure, and, from such features, it is possible to approximate likely table location. Table content can be analysed by relying on features such as alignment. Putting these together, tables can often be identified at least partially [eS10].
- Image Format: Image document formats provide an accurate representation of the absolute position of each pixel on a page. However, associating pixels with the types of document elements that are useful in processing tables is challenging [SSKD10]. Image documents would typically be pre-processed by

OCR software in order to at least identify characters in the document. While the location of document objects can be determined reasonably precisely, the identity of objects is generally not as accurately determined.

- HTML Format: HTML and other mark-up formats have little information regarding the absolute position of items and the formatting of the document elements, leaving those aspects to the rendering engine. Considering that the primary source of HTML tables is Web pages, and many Web-tables are created just for layout, not for representing relational information, detecting genuine tables in Web documents is a challenge. Even if the detected tables are genuine, the other problem with this document format is that HTML is not designed for information processing, but for the visual rendering of Web documents. In order to deal effectively with HTML documents, a useful strategy is to map the HTML into a format that is more amenable to data manipulation (e.g. XML) [KL06].

- PDF Format: PDF format provides detailed positional and coordinate information about each word fragment in a page, along with font information. However these word fragments do not necessarily appear in the PDF document in an order that makes it easy to identify objects that are spatially related (e.g. the elements in a table row, while spatially adjacent, are not necessarily adjacent in the PDF). Therefore, to assist table analysis, the document content needs to be pre-processed (e.g. rebuilt in the correct reading order) and this is not always a straightforward task [JY09].

2.2.7 Table Processing Methodologies

Regardless of the variety of applications, there are three main approaches for table processing [YKM05]:

- Pre-defined templates and layout: These approaches use templates/schemas for possible table structures and the algorithms use a set of strict, pre-defined table layout information to detect tables based on the templates. This approach is mainly used for image document processing; the input documents are scanned and any part of the them which fits a template is identified as a table. The obvious limitation is that we can only identify tables for which have (manually) defined templates. Mohemad et. al [MHON11] proposed a predefined table layout structure for text which uses a combination of heuristics-based, rule-based and predefined layout approaches to detect paragraphs, lines and tables in PDF documents. This approach shows good performance for tender documents (e.g. for building construction projects), but is not a general solution.

- Heuristics-based approaches: These approaches use sets of pre-defined syntax rules for "table grammar" to identify objects of interest. The complex heuristics are usually based on local analysis and often require complicated post-processing. Heuristics-based techniques remain dominant in the literature, however. Some research has used heuristics-based techniques for table detection [YKM05, KL06, KH06, LMG08a, OR09, DSEG11, HS10, TAN+14, CL14, CHCG15, EKNS16].

- Statistical approaches: Statistical or optimisation based algorithms use training-based methods for developing rules for table recognition [eSJT06]. These methods can make use of parameters to tune their performance[eSJT06] or may be parameterless. This approach shows some promise, but only a few research works have so far made use of statistical approaches for table detection [CHJ02, WPH04, LTSX06, WCC08, EFBDM08, DMFE13].

In this thesis, we propose that table processing is best carried out using a collection of well-defined independent tasks.In any system involving a set of tasks, different

approaches my be used in the sequencing of tasks. Three main approaches are identified in the literature [eSJT06]:

- Top-down: Such systems begin by finding the whole table, then search for its lines and columns and finally assign the cells to the formats thus built a top-down approach.
- Bottom-up: Such systems begin by finding individual document elements and then combining them to form columns, rows, and ultimately, whole tables.
- Mixed: Some works follow a mixed approach, e.g. begin by finding columns or rows, then gather them into tables and finally search for cells.

2.2.8 Table Processing Methods

Different research communities have developed a wide range of methods and techniques for table processing. Zanibbi et.al. [tBC04] divided the techniques into three main groups:

- Classifiers: Classification techniques have been used in different table processing tasks: locating tables (e.g. training a classifier on content features of individual cells and non-text layout features to locate tables in HTML [LTSX06]), classifying text regions in a page to detect the table cells [Hur00b], table header classification [FMTG12] and detecting critical cells for table understanding [DMFE13]. Classification techniques also used for the purpose of table census and classification [CP11], table search [KHKL12], document classification [KL11]. Some of the most common classification techniques used for table processing are: decision trees [CP11], nearest neighbour [PGL06], neural networks [eS10], syntactic [LE08], and statistical methods [EFBDM08]. Decision trees are by far the most common technique

- Segmenters: These techniques determine if a specific type of table exists in a document. Segmentation techniques can either cluster or partition the data. Clustering methods are used to set parameters adaptively (e.g. using K-mean clustering) [KGK+07] or to cluster regions hierarchically based on a distance measure [SKB08]. Partitioning algorithms can range from simple segmentation (e.g. segmenting based on the blank lines) [EKNS11] to more sophisticated ones (e.g. based on XY cutting) [Jin13]. The differentiating factor in these models is the type of search used. The most common algorithms in the literature are recursive partitioning methods which are a variant of the XY cutting algorithm and eventually create a hierarchy of regions that may be used to determine the table grid or cell arrangement in the document [JKN+09, Has10b, HL14].

- Parsers: Parsing methods are used to produce graphs of the table structure based on syntax defined using a table grammar. Parsing techniques can be categorized by the type of grammar encoding: hidden Markov models [BOO15], attributed context-free grammars [NJ07], and graph grammars [AA03]. There are also techniques described in the literature for automatically inducing grammars from data [SJKN10].

2.3 Table Processing Research

Different research communities have tackled the problem of table processing, generally with different applications in mind. In this section we review these works based on the 4W+1H model. We break the works based on the research communities and in each community we describe the objective, research ideas, techniques and where the work sits in relation to the rest of the table processing literature. It should be mentioned that what we review in each community is simply their approach to table

processing; other goals considered by the research community are outside our scope.

2.3.1 Document Analysis

The focus of this research group is mainly on document layout analysis, document manipulation, and document management. Tables are involved in document analysis simply as one of many kinds of objects to be found in documents. They mainly consider tables from the table layout structure perspective and follow the editing and formatting application paradigm.

1. Layout Analysis: Mainly focuses on the spatial layout and geometric attributes of document objects [FSCG03]. Spatial relationships and positional relationships of objects are used as a basis for basis for detecting the object arrangement order [GTL+11] which can be done at both page and document level. In order to detect the tables, especially those not explicitly tagged, visual cues are exploited as well as the document hierarchical structure (e.g. DOM Tree representation) when available [KHG05]. The main aim is to detect the outer grid structure [LMG08a] or change the table size and layout based on styling and topological rules for better display and presentation [ELN06]. Techniques used for this application include: heuristics which give different weights to each class of layout features [ETL05], X-Y cutting algorithms which cut the document page recursively to establish the document's hierarchical representation.

2. Document Format Conversion: The focus in these applications is to develop a system to convert a specific types of document to another format, either to increase the accessibility [JY09] or to convert the document to a more popular and structured format (e.g. HTML) [GH13].Tables, as one of the elements in the document, should be detected and presented in the target

format. This can be done by preserving the exact layout and styling features of the document [Som04] or developing a wrapper system to provide selective conversion [Has09]. Regardless of the application, DOM trees and bounding boxes are typically used [CHJ02] to represent the structure of the document as a basis for mapping to the target format structure.

3. Document Structure Analysis: In these applications, the aim is to detect the logical structures within the documents which are usually untagged [MHON11]. They can be discussed at page level [RCVF03] or the document as a whole [LTSX06]. Logical structure detection is generally treated as a classification problem[GTL+11]. The approaches are either rule-based [Coü06] or follow learning-based techniques [WH02b].

4. Document Similarity Detection: The aim of these applications is to determine the relationship (or lack thereof) between two documents. One approach to measuring similarity is to pairwise compare different regions of the documents. The granularity of regions varies from character level to larger document structures like tables [WMD10]. A range of methods is available for this purpose, such as edit distance [HKLW02], graph similarity and matching techniques [AA03] and machine learning methods [WCC08].

2.3.2 Image Document Processing

Image document processing has its origins in the image processing research community and focuses on document image analysis, page segmentation, and the detection of forms and tables from scanned documents [DT+14]. Here we summarise some of the main works in this community related to table processing.

1. OCR improvement: The goal is to improve OCR techniques in order to detect

document structures rather than simply transforming them into a sequence of texts [EFBDM08]. For this purpose, the main focus is on enhancing page segmentation techniques [WPH04, SKB08]. The X-Y cutting algorithm [HL14] and white-space analysis [CL12] are the most discussed methods in this area.

2. Document layout description and understanding: Image documents, like other types of documents, contain various regions which can be assigned different functions and roles. Document layout description and understanding tries to decompose a given document into its functional regions [NJ07]. There has been significant work on historical and ancient document layout analysis [ACPP11]. Region classification methods, especially statistical based techniques, have also been used [WPH04] in the context of table location.

3. Table Detection: Table detection and identification from image documents has been discussed particularly in the context of digital library applications [MCDC06]. Different features have been considered for detecting tables, such as character bounding box, rulings [CL14], and document regions and blocks [MCDC06].

2.3.3 Information Retrieval (IR)

Research in the Information Retrieval community focuses on text content representation and linguistic analysis to provide better document indexing and retrieval models [Pan02]. Here we describe some of the works related to table processing that originate from the IR community:

1. Question Answering (QA): Since tables are typically used as a summary visual presentation of important information, they are also a potentially important source for question answering, to provide specific results rather than just listing

related documents [PBC+02]. Different approaches exist toward improving information retrieval tasks by enhancing table processing. Some work relies on accessing tabular data and integrating it with the rest of document information to improve the indexing [Bur07], other works uses table headers as metadata to enhance the scoring algorithms [LBMG07b]. Both heuristic [WCP04] and learning [PMWC03] based techniques are used for these purposes.

2. Table Indexing and Search: This work focuses on indexing tables to improve subsequent table search [LBMG07a]. Different approaches are discussed for this purpose but they mainly extract table metadata to use in table indexing. Metadata is gathered in different levels such as table context and environment, table affiliation, cell-content, etc. [OSO+14]. Others try to convert tables to database structures to enable searching table content [EKNS16].

3. Structured Querying: The aim of this work is to handle structured queries over unstructured documents by utilising a combination of information extraction and information retrieval techniques[CBC+07]. Padmanabhan et al. [PN08] use well-formed tables as a representation of structured queries; the tables, containing gaps to be filled, are mapped into Wang notation and then used to derive a query on an underlying database which produces values to file the "unknown" table slots.

4. Table Classification: This group of applications aims to improve table understanding and accessibility by classifying tables. Classification techniques are based on either the content of the table (e.g. scientific table classification) [KHKL12] or the table structure (e.g. lists, forms, matrix structure) [CP11]. Other work has investigated the use of a table's logical structure to relate it to other tables with similar logical structure [NEKS15] and on the classification of individual cells to improve the analysis of table structure [IFS05]

2.3.4 Database

The focus of the research in this community is on data management, schema mapping and matching and transferring tabular data to known databases to integrate with other data. Here we summarise the main research trends in the database community related to table processing.

1. Populating Databases: Inserting data from tables in documents into a database enables the data to be manipulated using the power provided by database technology, e.g. utilising OLAP or other related techniques to analyse the data [ESN14]. One approach is to extract attribute and value pairs from tables and use this information to drive insertion into the database [HS10]. End-to-end solutions, which take a document, discover tables, extract table data, and map it into a relational database have been developed. One example [NSE14] uses generic table structural analysis to populate databases from tables in HTML documents. Another example [ATV13] uses semantic or domain ontologies to understand table data and map it into a database.

2. Data Model and Language: Modelling the data contained in tables in documents, in a framework suitable for mapping to relational databases, is critical to exploiting this data. Pivk et.al. used a frame-based knowledge representation language to map table structures to Logical Frames to provide a semantic representation of tables [PCS+07]. The Wang Abstract Table model [Wan96] has been extensively used as a generic target model for tables (and is discussed in detail later). Ermilov et.al. [EAS13] proposed a canonical table model and also a mapping language to translate table data into RDF.

3. Data Integration and Schema Matching: Data integration (and its pre-cursor, schema matching) has been a long-standing and well-researched problem in the database community. In the context of table processing, it manifests as

converting document table data into a form that captures the logic structure of the table data [GHJ+10]. A generic solution, involving table similarity matching, was proposed in [DSFG+12]. Several domain-specific solutions, typically relying on pre-defined ontologies, have been proposed. For example Butch et.al. [BDBIS13] proposed a data integration method for web tabular data which is guided by a domain-specific ontological and terminology resource.

4. Multidimensional DB and Data Warehouse: Since tables contain multi-dimension information, they provide a good source of information to be considered as a data cube for a data warehouse or multidimensional database. Gagliardi et al.[GHPS05] proposed automatic construction of a domain-specific data warehouse from tabular data embedded in heterogeneous web documents. They extract tables and transfer them into generic XML representation called XTab, then enrich them semantically to store in the data warehouse. In [Shi15] tabular data from unstructured documents is transferred to a canonical form in Excel. Their CELL system uses a set of rules based on spatial, style and natural language information to extract table data, whch is then loaded into data warehouse via a post-processing step. Jandhyala et al.[JKN+09] extract relationships between nested table headers and the data cells, build an X-Y tree representation of the data, and finally store it in a multidimensional category tree structure. Norah et.al. [AL12] develops a system for converting multidimensional tables with non-overlapping hierarchies to data cubes. They first extract table metadata from table titles by using natural language processing and then attempt to integrate tables by matching common dimensional information.

2.3.5 Web

Since the Web is a massive source of information, a considerable amount of research has been conducted on Web table processing (some of which was mentioned above). Some work aimed to provide an understanding of the different types of tables on the web [CP11] and to distinguish between tables used for presentation and layout purposes and "genuine" (data) tables [YL02]. Here we summarise some of the main applications of table processing form the Web research community point of view.

1. Semantic Web: This group of applications aims to facilitate the construction of the semantic web by better extraction, interpretation and representation of tables, including both tables from web documents and tables from other sources. In [Mul10], tables are represented as linked data. Every table column header is assigned a label from an a related ontology and then table cells are linked to an entity from the Linked Open Data cloud. Machine learning techniques are used to rank the labels and entities that column header and table cells should link to. Lushan et.al. [HFP+08] propose an open-source tool to map and translate tables in spreadsheets to RDF. In [EAS13] a function for mapping relational data (canonical tables) to RDF (so called R2R) is discussed. In their mapping, each table is considered as an SQL table and the expressions specifies which RDF term type to generate form the underlying SQL expression.

2. Web Table Conversion: Since tables represent a significant portion of web data, web table conversion to more structured format is an important application of table processing in web community. Embley et al.[ESN14] aimed to map data from HTML tables to a relational database. This involves two table conversions: mapping the HTML table to CSV, essentially maintaining the original grid structure, and then converting to canonical form based on

the extracted categories. An approach to automated web table understanding through interactive table conversion was proposed by [PJK+10]. Their method is to convert HTML tables to a layout-independent representation via an interactive tool (Table Abstraction Tool (TAT)) which produces XML which is then used for constructing narrow-domain ontologies. Shared et al. [SJKN10] use an XY cutting algorithm based on some specific rules on table column headers. They transform and convert the geometric structure of tables to a linear string which contains the cell attributes with the direction of the cut. The result feeds to a system for creating ontologies.

3. Information Extraction: While HTML is a tagged language, and while `<table>`, `<th>` and `<td>` tags look like a convenient way of identifying table components, much use of tables in Web documents is as a layout control mechanism. One difficulty in extracting information from web documents is distinguishing genuine tables from table tags used for layout. In an attempt to deal with this, [GBH+07] proposes a two dimensional visual box model (as used by web browsers) rather than a tree structure to represent the web tables. Pranit et al. [PCC12] review the challenges of information extraction from web tables; they define what is a web table and how in "meaningful tables" the content is related to the structure. They propose five modules for information extraction from web tables: web page parsing, table validation, table normalisation, table interpretation and attribute-value pair formation. Other works look at tables as attribute-value pair structures. For example, [WWLN09] describes a methodology for extracting value-pairs from web tables in two phases: candidate generation, where syntactically likely attribute-value pairs are annotated; and candidate filtering, in which the improper semantic annotations are removed.

4. Web Search: Recent research on Web search has considered how to automat-

ically construct a structured database derived from the Web; if this can be achieved at scale, the existence of a structured database, along with a formalised query system based on this, would offer the chance for considerably more powerful Web searching that simple keyword-based searching. In order to provide a more general and domain-independent web search, [Caf09] proposed a system named Omnivore which builds a large web database by relying different domain independent extractors over a web crawler. The result is combined into the large entity relationship database. This system uses a specific extractor (WebTables) to extract relational information from tables. Other approaches to improving web search through table processing rely on table context extraction and table augmentation. [BTEL15] proposes a context extraction system for web table columns. Context information is drawn from table caption, headline, surrounding texts and table full text . A matching algorithm then decides whether context labels are either directly or indirectly related to table columns.

2.3.6 Semantic and Knowledge Management

A considerable amount of research on table processing focuses on facilitating semantic understanding of table, either for extracting linked data or improving knowledge management. The application are varied: creating an ontology, semantically annotating tables, or enabling semantic search. Here we summarise some of the main trends in this community.

1. Ontology generation: Considering the attribute-value structure of tables, they seem to be a valuable source of information for ontology creation [LY10]. TANGO (Table ANalysis for Generating Ontologies) is one of the approaches for ontology generation that focuses on table processing [TEL+05]. Table anal-

ysis starts with table recognition and canonicalisation to provide an abstract representation of tables, often semi-automatically [JN08, NT12]. Canonical tables can then be used to create mini-ontologies which can then be matched and merged with an existing ontolgy to incrementally form an ontology for a specific domain [Emb04]. Another approach is embodied in TARTAR (Transforming Arbitrary Tables into Frames) [PCS+07]. TARTAR follows four main steps: cleaning and normalisation of Web tables by representing them on DOM tree; structure extraction to detect table orientation, cell function and logical units and regions; arranging table regions into trees (capturing the table hierarchy); semantically enriching the table functional model by discovering semantic labels for trees elements and then mapping them into a frame structure[Piv06].

2. Knowledge Representation (KR): Research in KR applies to several phases of the table extraction process, from how to represent information extracted directly from tables, through to representing the semantics of the table data. Oro et al. [OR08] propose a system called XONTO which follows a semantic approach for information extraction from PDF documents relying on Self-Describing Ontologies (SDO) for knowledge representation. Table extraction is accomplished via their PDF-TREX system [OR09]. Long [Lon10] proposes an agent-based approach for table interpretation relying on an RDF-based blackboard framework. The choice of a blackboard framework enables the development of different knowledge representations as needed during the table analysis process. RDF is used for knowledge representation since it allows access to both low-level content (e.g. position and coordinate information) and high-level information (e.g. descriptions of tables). RDF also enables subsequent integration of table structural analysis with table interpretation [Lon09].

3. Semantic Annotating and Augmentation: Different approaches have been dis-

cussed for employing semantic annotation to make table content more understandable. Nagy et al. [NPJ+10] define table augmentations as attributes of a table which are not necessarily incorporated into the table's physical or functional structure. Examples include: Table Title, Table Caption, Note, Unit. Augmentation may apply to the whole table or to a single cell. They also described table aggregates like total and average which often serve as proxy headers. Lynn et al. [LE08] propose a semantic-enrichment procedure, employing annotation, that transforms syntactic table structures to semantically coherent ontological concepts, relationships and constraints. Their proposed system exploits auxiliary world knowledge to recognise concepts and data, to identify relationships among concepts, and to determine constraints on the concepts and data. Tao et al. [TE09] propose a solution for detecting so called sibling tables in sibling web pages, by which they mean pages generated from the same underlying database. For this purpose, they introduce a table preprocessing phase to transform web tables into DOM tree representation and then through a table matching procedure, tables are classified into three class of exact, false, and sibling match. Sibling matches are investigated to recognise the similarity patterns and analysis of the patterns lead to either mapping to a known pattern, or a new pattern generated by combining similar patterns. Patterns are subsequently used to interpret tables.

4. Semantic Search: The aim of the semantic search is to improve search effectiveness by better understanding the "meaning" of the user's query, including drawing on context to disambiguate search terms. This typically relies on having some structured representation of the domain of interest. Pivk et al. [PGL06] aim to extend semantic search to include searching on tables in web documents. In order to do this, they process tables to extract ontologies and incorporate these in a knowledge base which can be used in subsequent searching. Lupiani et al. [LRGMVG+11] proposed a semantic search

engine for financial news due to the increase of large amount of financial information available via a number of heterogeneous business sources. Part of their system was a module for performing semantic transformation on tables.

2.4 Gap Analysis

Much current research has focused on designing and implementing specialised 'black-box' systems to address different aspects of the table processing problem in specific domain contexts. Although the technical and algorithmic requirements of the tasks involved in table processing are well researched and analysed, a systematic view of the end-to-end process is missing. This approach leads to significant re-development costs when applying table processing in a new context.

Our discussion on the application of table processing from the point of view of different research communities indicates that table processing is an important component of many applications. The review also revealed that tables are described at several different levels, either physical, logical or abstract. Many current systems are manual or semi-automatic, and reports on their effectiveness are often accompanied by a discussion on the necessity of a more automated solution [GJJH12].

There are some efforts to incorporate the concept of reusability and modular design into table processing systems under the name of "End-to-End table processing system design" [eSJT06]. Such an approach is beneficial from a number of perspectives, including enhancing the ability to develop and test alternative approaches to table processing. Looking at table processing from a wider perspective, it would be extremely useful to have not just end-to-end table processing but also a general model for tabular data management to contain the storage, indexing, querying, and maintenance of both the original raw data and the extracted information. (The

same challenge has been discussed for managing information extraction at scale to provide a unified unstructured data management system [DNR+09].)

2.5 GMTPS: General Model for Table Processing Systems

Considering the literature gap and the necessity for repeatable and reusable process for table processing across different domain, we propose GMTPS (General Model for Table Processing). A possible design for a general-purpose table processing and management system is presented in Figure 2.13.

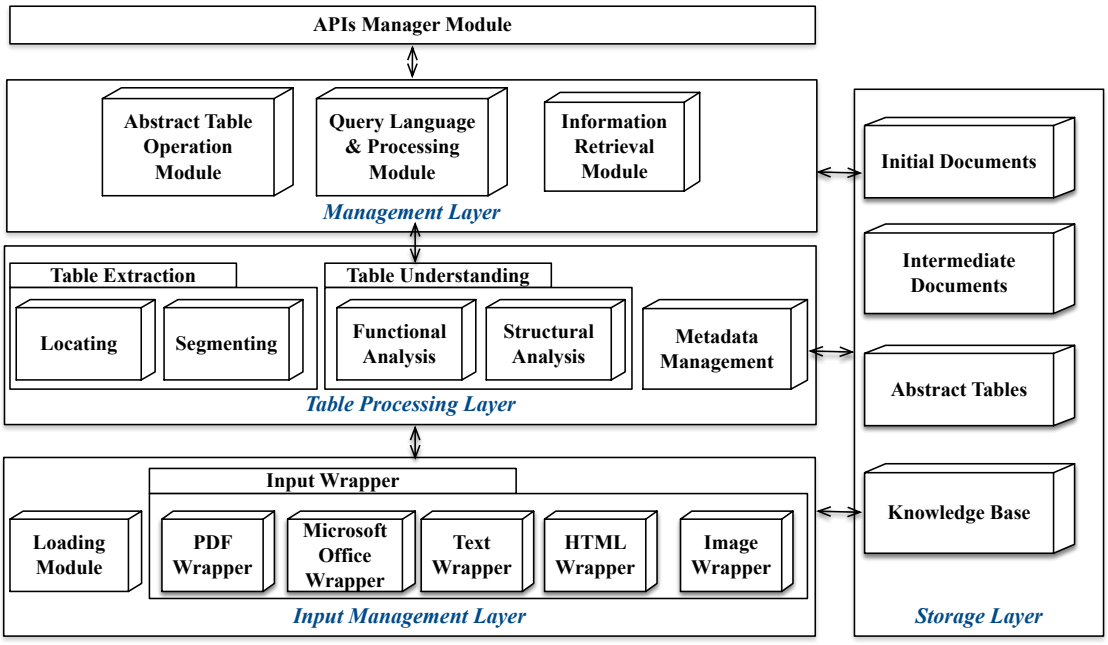


Figure 2.13: GMTPS components

The framework consists of four main layers:

- Input Management Layer: This layer manages the input documents. The

central elements of this layer is a 'Table-Oriented Document Model' (TODM). The idea is to load documents through the loading module and apply a set of wrapper systems to transform the documents to TODM. The details of the TODM and a sample of this wrapper for PDF documents is discussed and explained in Chapter 4.

- Table Processing Layer: This layer is responsible for specifying and executing table processing tasks. The overall table processing task is divided into the sub-tasks of Table Extraction and Table Understanding. The ultimate output of table processing is a model of each table in a layout independent notation known in table processing community as Wang's Abstract Table (AT) model [Wan96]. All metadata related directly to the table structure or the table context will be extracted and managed through the metadata management module. The standard definitions of tasks, interfaces and related data models are discussed in depth in Chapter 3.

- Management Layer: The management layer is responsible for storing, indexing and managing the extracted tables and data within them. This layer works based on the Abstract Table model (tables and relevant meta-data). For these purposes, the concept of managing tables as first-class-citizens will be introduced and atomic operations within and between abstract tables will be defined.A query processing module implements an algebra-like query language and provides a wide range of operations (see details below), and an information retrieval module provides the capability to explore tables by content. This layer will ultimately expose the operations as a suite of APIs to support custom application-building activities by end users (e.g. "table mashups") whereby the tables and data contained within them can be combined, visualised, tracked, and exported.

- Storage Layer: This layer stores all forms of data before processing as the initial

unstructured documents (i.e. PDF files), during the processing as intermediate documents containing the output of different components of table extraction and understanding and final extracted structures in the form of abstract tables and knowledge base containing necessary information for semantic analysis. Since these different forms of data have variant characteristics and frequency of use during running of system, as mentioned in [DNR+09] it seems better to keep them in different storage systems.

Operations on tables: One category of operations manipulates individual ATs ($AT \rightarrow AT$). This category includes transformation operations (create a new AT, rename, insert/delete, duplicate an AT, change the AT at data and metadata level) and also filtering operation (selection, projection). The second category of operations applies to a pair of ATs ($AT \times AT \rightarrow AT$). This category includes operations such as similarity detection, edit distance, and different types of join, union, intersect, merge. The third category is defined on a set of ATs ($\{AT\} \rightarrow \{\{AT\}, ...\{AT\}\}$) and contains operations like filtering and clustering. The query processing module will also incorporate several different notions of similarity (structure, data and metadata) for use in with the operations, all with associated distance measures. By using these operations, we will be able to, for example, merge or join abstract tables, group or summarise data within a collection of tables according to a set of criteria. Also, we can define semantic relationships between two abstract tables.

The above provides a big picture view of a generic table processing system. We cannot hope to accomplish all of this in this thesis, and so focus on the table processing, since it is at the heart of the model. Our primary goal is to improve end-to-end table processing, to make it more generic and automatic, although we will also consider some requirements from the other levels, such as input management.

The above model drives the structure of the rest of the thesis. We give a formal

definition of End-to-End table processing system in Chapter 3. Then the PDFWrapper system and the detail of TODM is described in Chapter 4. Improvements to make table understanding more automatic and domain-independent are discussed in Chapter 6. Chapter 7 reviews some of the current challenges in automatic table processing and discusses how our proposed model can be exploited to make progress towards solving them.

Chapter 3

Task-Based Table Processing Framework

This chapter introduces the concept of Task-based Table Processing and the atomic tasks are comprised in an End-to-End Table Processing. The design of the system is explained in details. The proposed system design can be realised in different way to fulfill the requirements of the 'Table Processing Layer' in our proposed GMTPS[1].

3.1 Motivation and Background

Various research communities, such as machine learning and information retrieval, have worked on the problem of table processing and many approaches have been proposed. However, existing approaches almost always tackle only a subset of the problem (e.g. tables in a specific domain or with particular layout), or focus on sub-tasks of the complete table processing problem (e.g. locating tables).

[1]General Model for Table Processing Systems

There has been some work towards building a coherent, systematic view of the table processing problem. An important seminal work is the PhD thesis by Wang [Wan96] which, while not describing a table processing system, gives a table abstraction model which has been widely cited in the literature [JN08, TE09, PN08, PJK+10]. This model separates a table's logical structure from its layout structure, which consists of tabular topology and typographic style. The notation of an abstract table to describe the logical relationships among tabular items is formally defined, and a set of logical operations is proposed to manipulate tables based on these logical relationships. Because the model describes tables in a layout-independent way, it is used as the target model for many table processing systems which aim towards information extraction [JN08]. Despite the widespread use of Wang's model, it is still a challenge to develop systems which can automatically derive a Wang-representation from a table in a document.

The pioneering work in end-to-end table processing is Hurst's PhD thesis [Hur00a] which gives a general table model with different abstraction levels (Physical, Functional, Structural, Semantic) to facilitate the interpretation of tables. Although Hurst provided a comprehensive table model, his focus is on the model rather than on the details of the process of table extraction and understanding. Additionally, his discussion of process deals only with determining the internal structure of an identified table, and not with locating the table in the document.

Hurst's work was followed by Silva [eSJT06, eS10] which described the steps involved in extracting information from tables based on the Hurst's model. Silve proposed an end-to-end method by focusing on the interaction between the different steps and by considering feedback loops between the steps to reduce the possibility of errors. Silva's work is important in identifying a set of basic tasks for table processing and reducing the monolithic, black-box view of the problem. However, no formal definition of these tasks and their inputs/outputs was given and only some components

of the proposed design were implemented.

Long [Lon10] was the first to implement a table processing architecture that encourages collaboration and component re-use. Long defined a multi-agent blackboard architecture, along with guidelines for table boundary identification and table interpretation. Individual agents tackle different partial solutions like processing input characters and processing input lines. Although the system design is sound, and suggests that components from other table processing systems might be incorporated, it is not specified how to achieve this, and the issue of conflict resolution over heterogeneous agents is noted as an open problem.

A range of diverse implementation approaches are employed in the literature for detecting tables. Some systems use predefined layout-based schemas. In this approach, which is aimed mostly at images, parts of the document having the same structure as the predefined template are captured and reported as a table. Of course, defining a wide enough range of useful table schemas is a tedious and inefficient task [MHON11]. Another approach is heuristic-based techniques [YKM05] which, while slightly more adaptable than schema-based approaches, are generally targeted at particular classes of tables, and may not be able to handle table instances outside those classes. Others have developed statistical-based methods to detect tables, including Probabilistic Modelling [eS10], Naive Bayes Classifier [OR08], and Support Vector Machines [LMG08b]. According to the analysis presented in [eS10], most of these systems applied off-the-shelf techniques, which needs lots of improvement to be optimised for document and table processing problems.

Taken together, these works almost provide a complete framework for discussion of table processing. However, they were developed independently and do not dtailed well enough nor provide sufficient detail to support the development and evaluation of full end-to-end table processing systems.

Our goal is to integrate aspects of the above related work to produce a framework for end-to-end table processing which: (a) identifies the sub-tasks involved (as in [eS10]), (b) gives precisely defined models to describe the input and output of each sub-task (as in [Hur00a]), and (c) packages this as a collection of modules (as in [Lon10]), with well-defined interfaces, to provide a "workbench" for developing tools and techniques to further the state-of-the-art in table processing. The input of a system built using this framework is a PDF document and the output is a set of abstract table models (as in [Wan96]) describing each of the identified tables in the document. Our original contribution is the integration of these ideas, the development of the models for each sub-task, and extensions to some of the methods/models outlined above.

We believe that this approach provides the following benefits: (1) a well-defined decomposition of the task into generally agreed components (2) a consistent vocabulary for describing system scope and goals, (3) reusablity and repeatability in system design and development (4) the opportunity for interoperability of different implementation techniques and approaches. Our end-point is an application-independent model of the tables in the document (a mapping from headings to data cells, in Zanibbi et al.'s [tBC04] terms). We believe this provides a suitable starting point for semantic analysis and other kinds of domain-dependent analysis, and so we adopt the well-known Wang abstract table model [Wan96], extended to handle tables with no obvious stub.

3.2 Preliminaries

3.2.1 Different Level of Table Description

In a domain-independent approach, tables can be defined in two main levels:

- Physical Table: Tables as physical tables contain a number of cells encoded in terms of *relative position*. **Cells** are the main meaningful elements in this level which can be defined as areas within a table delimited explicit by line arts or implicitly by spacing and co-linearity of text. In order to describe tables in physical level most of the approaches rely on coordinate system to indicate the location and relative area of any grid of rectilinear shape.

  Although the main focus is the positioning of the cells, there are some other factors at the physical level (mainly typographic phenomena) which may effect the meaning of the table and are considered in table modelling in the physical level in some of the table processing systems:

  - Cell Shapes: Mainly most of the models for physical tables, represent the table cells in rectilinear shape and allows no explicit representation of alternative shapes such as triangles and parallelograms [eS10]. There are some colour-based techniques (mainly for image documents) [SSKD10] which embed the cells arrangement information into the colour channels and support various type of shapes in colouring. However, Hurst [Hur00a] suggested that the qualities of the cell shapes can still be preserved in coordinated based systems if required.
  - Text Orientation: The physical table encode what the cell content is and not really preserve the orientation of the text. Text content can be set vertically or horizontally in the cell or with any number of line breaks. However, these features mainly used for the typesetting purposes and not really convey information about the cell content.
  - Text Justification: It contains any vertical text alignment (justification, indentation, centering, etc.) of the cell contents with the respect to the cell space [ELN06]. Considering this information in the scale of table columns and rows can convey useful information. For example, in some

tables, the header hierarchy is shown by indentation in the cell content. We show in Chapter 6 how we use this type of information in our system to improve automatic table understanding. However, the justification of cell content gets more important in deducing cell delimitation in the absence of any line-art.

- Text Font: In some table layout design, different font (type or face) is used to show different functionality of the table cells. Therefore, if this information is present and used consistently may will indicate useful information for better table understanding. Font information is used in mainly most of the table processing systems for PDF input format in different ways [JY09, FMTG12].
- Line-Art: Line-arts may be considered as graphical properties of the tables [CCF+06] in the means of horizontal and vertical lines to specify different boundaries in the tables. Line-arts typically consist punctuation characters (hyphen and under score are the most common) and used as delimiters (for rows, columns or cells) in the table. Some of the table processing systems captured them in their underlying model for specially locating and segmenting tables [LDC05].
- Colour: In some tables, colors are used either for the presentation of the text or the background and line art to highlight some information presented in the table. In some systems these features are considered as a part of physical table modelling [GWWS01].

- Logical Table: While physical level describes where different regions of tables are located in the document, logical level defines the type of these regions and how they make a meaningful structure to convey information. Tables logical modelling should clarify the exploitation of the reader from table and address issues such as how the information should be read from table, which cells in the

the table are the real target data cells and which ones provide access routes to the data [NT12]. In this way, a reading path can be described for tables which indicates a path which the reader takes through the cells to read or locate the information in the table. The reading path is essentially a logical path which means the cells may be logically adjacent not and indication of a navigation through the table via cells which are physically adjacent to each other [Hur01]. It connects cells which are related through the meaningful organisation of the table.

According to the [tBC04], tables can be described in different logical levels.

- Functional Level: Mainly in this level, the table is divided into areas that play similar function (e.g. data cells versus header cells in the table). The underlying relations between different functional cells may not be represented in this logical level. Some assumptions about the location of the cells in the tables are made in most of table processing systems to assign function roles to table cells. (e.g. cells at the first rows of tables are more likely to be header cells or cells at the last line of tables are probably data cells [SN13]).
- Structural Level: The most abstract and domain-independent level of logical structure for tables in table processing literature is defined as the indexing scheme from header cells to data cells located in the body of a table [JN08], which separates table layout structure from table logical structure and represent the logical relationship between table cells [Wan96]. This level provides information about the logical relationship between table cells which can be extracted from the structure of the table. In most of the current modelling, transformed tables to canonical tables to reduce table relations to the ones can be found in a relational database [EKNS16]. In this way the logical connectivity of cells is preserved (e.g. to be shown in reading path) while discarding other structural arrangement

of cells. For this purpose some redundancy may be added to the logical representation of tables which change the initial physical organisation of table cells [NEKS15].

- Semantic Level: The goal in this level is to semantic interpretation of information presented in the table and understanding the content of the table cells on a reading path and their relationship. Semantic level view of a table requires language processing and domain objects and relations understanding. Semantic understanding can be defined in various levels. Some models just restrict the semantic analysis to cell content and map them to known ontology objects which in combination of table functional model can improve table content understanding [QR13]. Some other include the semantic of the relations in the table in different levels of inter-cell relationship understanding [LSC10] or the interpretation of the relationship between table dimensions [BTEL15].

3.2.2 End-to-End Table Processing System Design

Considering the related work and its drawbacks, the design requirements of an end-to-end table processing system can be summarised as follow:

- The design should be domain independent as possible to cover variety of applications and goals in table processing. As mentioned in [GBH+07], many of the current systems are for a single domain and can not be re-used for any other applications.
- Different functionalities of the system should be able to independently designed and implemented, but still able to cooperate towards a table processing goal [Lon10]. This leads us to functionally decompose an end-to-end table pro-

cessing first and consider a component-based design then to reduce the system integration overhead.

- The system should be able to process tables having different layout and structure styles [FTT+12, LN00]. Most of the systems just consider a specific type of tables (simple tables, no hierarchy in headers, no spanning cells,...) and it even limits the system design principles.

- A comprehensive design should be able to consider different type of input document format [Liu09]. In order to process web tables, most of the current systems focus on HTML document and design their system based on the characteristics of the format.

- The table processing tasks should be repeatable. The system should provide a process which can be executed repeatedly without much overhead cost [eSJT06].

3.2.3 Design Principles

The process of extracting and understanding tables from a document can be considered as a multi-step information transformation process [RPS15, eSJT06] that takes a document as input and produces an abstract representation of the tables in the document as the final output.

In choosing the theoretical design framework for the system, besides the essential requirements identified above, we have considered the following. The design framework should:

- first, innately support the notion of self-contained, interface-based functioning

modules so that different table processing tasks can be represented as independent software modules,

- second, enable the notion of repeatability through a well-defined description of a task and a flow of tasks (conventionally referred to as workflow [CDKL07]).
- third, possess the actual implementation technologies available, and those technologies should be widely and actively supported by community to ensure that our implementation can be relevant for the long term.

Service Oriented Architecture (SOA) [Erl05] is an architectural approach to designing and creating software as a network of modular components. Each component implements a discrete, self-contained function. These components are called *services* and can be distributed across disparate IT systems. These services are built on open standards and loosely coupled, allowing them to be easily combined. Another important characteristic of these services is that they can be reconfigured into new processes as needed.

These characteristics promoted by SOA allow us to add componentry, extensibility and repeatability requirements to the modular nature of table processing. Reflecting SOA principles, in our design, we provide standard definitions for the tasks involved in the end-to-end table processing, and in turn those tasks are implemented and linked as services. This SOA-based design not only facilitates extensibility but also improves the interoperability and a systematic integration of the available functionalities [Bel08].

3.3 TEXUS: Task-Based Table Extraction and Understanding

We briefly describe the steps in the process with a simulated example. The process and the format of the intermediate results will be described in more detail in the following chapters. We define a series of data models to describe the inputs and outputs of the tasks. The input PDF document is initially partitioned into a sequence of *text chunks*, and these form the atomic elements of further processing. The final output of the table processing is represented by an extension of Wang's *abstract table model* [Wan96]).

Our core table processing tasks in TEXUS are as follows: (1) *Document Converting*: convert the PDF input document to our proposed document model, (2) *Locating*: find the tables in the document (their outer boundaries), (3) *Segmenting*: recognise the inner boundaries of each table (cells, rows and columns), (4) *Functional Analysis:* identify the role of each cell in each table (data or access), and (5) *Structural Analysis*: detect the logical relationships between table cells and provide the result as an abstract table. The last four tasks correspond to the first four tasks in Silva [eS10]. We omit Silva's fifth task (Interpretation), since it is inherently application-dependent, and our end-point is intended to be application agnostic.

The tasks would typically be connected in a simple pipeline, but more complex controls such as looping and nested composition can also be considered (although not in our work). Figure 3.1 shows a pipeline of these tasks implementing an end-to-end table processing system. In order to discuss table structures, we use Wang's table terminology [Wan96]. According to Wang, tables are divided into four main regions, delineated by means of a *stub separator* and a *boxhead separator* which are frequently, but not always, shown as physical lines. The lower-right region of the table (the *body*) contains the data. The upper-right region (the *boxhead*) contains

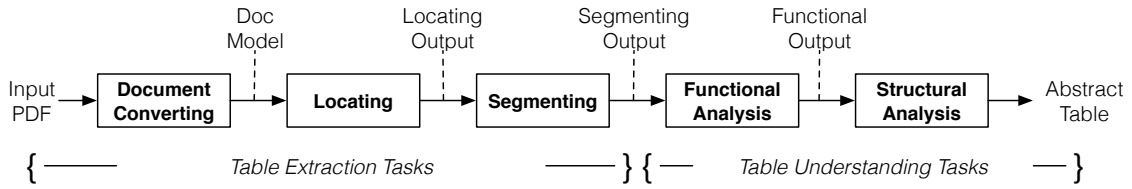


Figure 3.1: End-to-end Table Processing Pipeline based on TEXUS Tasks

column headings and sub-headings. The lower-left region (the *stub*) contains labels which provide access to the data rows. The upper-left region (*stubhead*) contains the headings for the columns in the stub. Figure 6.2 shows an example table with these regions marked.

Recall that the output of our system is a set of abstract table instances, one for each located table. Wang [Wan96] defines an *abstract table* by an ordered pair (C, δ) where C is a finite set of labelled domains and δ is a mapping from C to the universe of possible data values. The categories appear in the table as headings. The δ mapping relates the categories to the data values in the table body.

We illustrate the various concepts in Wang's notation via the example table in Figure 3.2. This table has two dimensions and therefore, two top-level categories. The first category is Faculty Cluster, with five subcategories (Sciences, Social Sciences, ... Total). Female Students is the next category with Sample and Population as its subcategories. The following gives examples of the kind of abstract table output obtained by processing this table:

Category set (hierarchy):

$C = \{ (Faculty\ cluster, \{(Sciences, \Phi), (Social\ Sciences, \Phi), (Humanities, \Phi),$
$(Civil\ Sciences, \Phi), (Total, \Phi)\}), (Female\ students, \{(Sample, \Phi), (Population, \Phi)\}) \}$

Two examples from the δ mapping:

$\delta(Faculty\ cluster.Sciences, Female\ students.Sample)$ = "63 (18.5)%"

$\delta(Faculty\ cluster.Sciences, Female\ students.Population)$ = "597 (16.4)%"

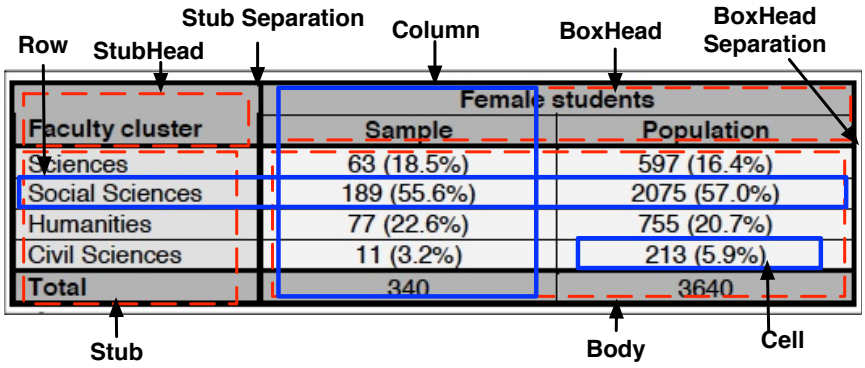


Figure 3.2: Wang Table Terminology

Note that tables typically appear in documents adorned by a title, caption, notes etc. These are not considered as part of the table, and hence do not appear in the abstract table model, but are included in the final output as *table metadata*.

adherents. People with no religion with 24% follow Christians, representing 67% of the population. Islam is the second largest religion in Canada, practised by 3.2% of the population. Rates of religious adherence are steadily decreasing. The preamble to the Canadian Charter of Rights and Freedoms refers to God and the monarch carries the title of "Defender of the Faith". However, Canada has no official religion, and support for religious pluralism and freedom of religion is an important part of Canada's political culture.

majority of Canadians consider religion to be unimportant, but still believe in God

Year	Provision	Religion Percent			Population
		Muslim	Christian	Jewish	
2001	Newfoundland	15%	35%	2%	196,966
	Nova Scotia	22%	14%	6%	390,328
2002	Newfoundland	17%	37%	5%	234,128
	Nova Scotia	21%	12%	8%	401,523

Canada today has no official church, and the

Figure 3.3: Part of Multi-column Document with Embedded Table

Figure 3.3 shows a two-column page with a table embedded amongst the text. The first step is to *Locate* the table in the document. As with most of the tasks in table-processing, a range of methods and techniques can be employed (we describe one such technique later). The locating step should ultimately result in identification of the boundaries that separate the table from the rest of the document (Figure 3.4).

Once the table is located (i.e. its outer-boundaries are detected), the table region should be *Segmented*, i.e. have its internal boundaries identified. After this phase, the table is viewed as a collection of cells, where cells are aligned horizontally in rows and aligned vertically in columns. Examples of the detected components are

Year	Provision	Religion Percent			Population
		Muslim	Christian	Jewish	
2001	Newfoundland	15%	35%	2%	196,966
	Nova Scotia	22%	14%	6%	390,328
2002	Newfoundland	17%	37%	5%	234,128
	Nova Scotia	21%	12%	8%	401,523

Figure 3.4: Located Table

Year	Provision Cell	Religion Percent Column			Population
		Muslim	Christian	Jewish	
2001	Newfoundland	15%	35%	2%	196,966
	Nova Scotia	22%	14% Row	6%	390,328
2002	Newfoundland	17%	37%	5%	234,128
	Nova Scotia	21%	12%	8%	401,523

Figure 3.5: Segmented Table

shown in Figure 3.5. The sequence locating-then-segmenting is termed the"table extraction phase".

Following table extraction, we try to *understand* the extracted tables by focusing on the contents of the cells and relying on hints from the table structure. The first step is to provide a *Functional* representation of the table. Every cell in the table plays a functional role; either it contains a data value or it acts as index to provide access to the data cells. Therefore, in functional analysis, each cell is categorised as belonging to one of the three functional roles: *Data*, *Access*, *Header*. Examples of each cell type are in Figure 3.6.

Year Access	Provision Header	Religion Percent			Population
		Muslim	Christian	Jewish	
2001	Newfoundland	15%	35%	2%	196,966
	Nova Scotia	22%	14%	6% Data	390,328
2002	Newfoundland	17%	37%	5%	234,128
	Nova Scotia	21%	12%	8%	401,523

Figure 3.6: Functional Table

Recognising the function of cells in the table determines a unique access path for each data cell. This, in turn, provides a representation of the table based on its logical structure and which is independent of the table layout. Every header cell is the starting point for a set of *access path*s leading to data cells. For some tables, we may need to add a *virtual header cell* to make it possible to define access paths from all header cells. Figure 3.7 shows two data cells in our example; the set of access paths to these data cells (as the result of structural analysis) are:

$$(\{Year.2001\}, \{Provision.Newfoundland\}, \{Info.ReligionPercent.Muslim\}) = 15\%$$

$$(\{Year.2002\}, \{Provision.Newfoundland\}, \{Info.ReligionPercent.Jewish\}) = 5\%$$

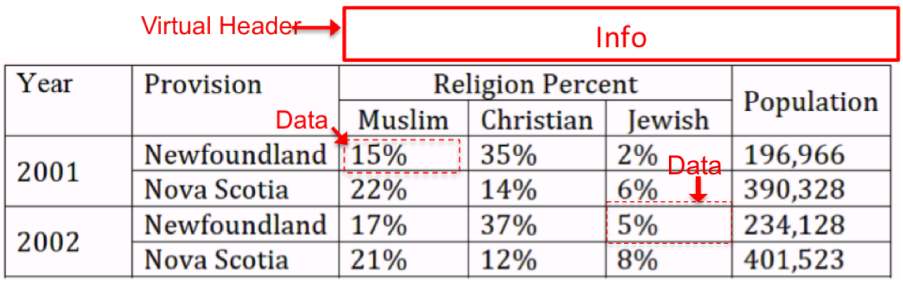


Figure 3.7: Structural Tables

The above processes provide an understanding of the table based on the table structure and the hierarchical relationship between cells. Table understanding can be extended to the semantic level by employing domain-specific knowledge about the table content. Data cells could be related to an underlying domain-specific-ontology, which could also provide a basis for determining relationships between cells. This part of the process is inherently problem-specific and requires special-purpose mapping modules to be constructed. Since our goal is to develop a generic framework, we consider table understanding only to the end of the analysis of the table's logical structure.

3.4 A Realisation of TEXUS

Through TEXUS, we intend to facilitate a systematic development and reuse of the concepts and their implementation defined within it. The tasks defined in TEXUS can be implemented and utilised as a set of components. Since the models in TEXUS only specify the expected output of each task, the exact details of how the outcome is achieved are left to the developers. They can easily choose compatible table processing standards, leverage existing component implementations and integrate them to build their own solutions, develop new components, or provide alternative implementation of the same components.

We have designed a service-based architecture for the realisation of TEXUS, which we briefly explain in this chapter. The details of the implementation of each component is described in the following chapters.

3.4.1 System Design

Figure 3.8 shows our own system design based on the models in TEXUS. In order to design a system based on the data model we consider each step of the transformation flow as one component, and use these components as a basis for a modular system which could be implemented via e.g. a service-oriented architecture.

The services can be consumed by end-users via directly programming the access to the services or through a well-established service composition tool. We have developed two types of the access to the components to create a pipeline of table processing. Figure 3.8 shows the different layers of the architecture. The proposed design allows for any previously constructed component to be replaced by another

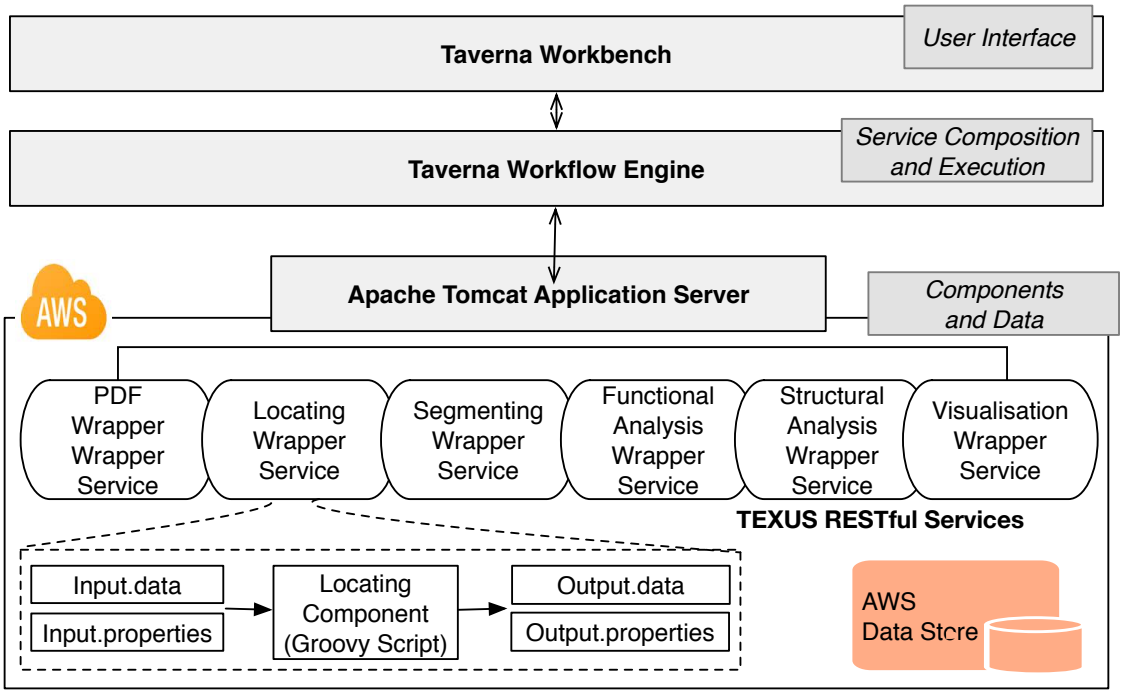


Figure 3.8: Accessing to TEXUS Tasks via service composition tools

"equivalent" component if needed. Each component is implemented using Groovy[2]. It is then exposed to the Web through a REST service which acts as a HTTP request/response wrapper of the component's input and output.

The wrapper services are implemented with Java and run in the Apache Tomcat application server environment. The behaviour of the wrapper services is uniform. Each accepts HTTP POST requests with two input parameters: the location of the data to be processed and a list of configuration options (if any). It then determines the location of the Groovy script (i.e., the component) to be called, reads the inputs to create "input.data" and "input.properties" files, then launches the component. When the component has produced "output.data" and "output.properties", the locations of the two files are returned in a HTTP response.

We have deployed the REST services and components on the Amazon Cloud plat-

[2]Groovy, Dynamic Language for Java, `http://groovy.codehaus.org`

form to provide easy access to the TEXUS components. Figure 3.9 shows this design of components.

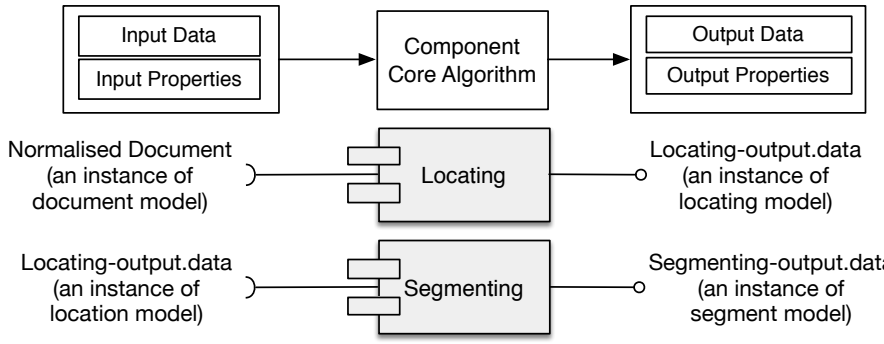


Figure 3.9: Design of the Components: (e.g., Locating and Segmenting)

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[3] script.

The important thing to note about these models is that they provide precise definitions for the input and output of each task in the table extraction and understanding process. This means that the implementation of each task can proceed

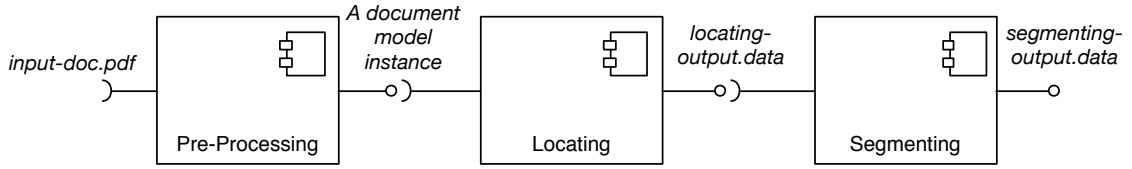


Figure 3.10: Linking Components

independently, and the output from each task gives a meaningful set of data which

[3]XSLT, `http://www.w3.org/TR/xslt`

could either simply be passed to the next task, or considered in its own right (e.g. for evaluating the effectiveness of the task that produced it). Figure 3.10 shows the main components of table extraction and their pipelined composition to form an end-to-end table processing system. Adding more components (e.g., to produce functional and structural tables is straightforward with this design.

Providing the components as services allows the client applications to be built in a flexible manner. For example, a client application can choose to use one component from the available pool, or use a well-known Web service composition technique to wire the multiple components together.

Our approach for implementing this layer allows for easy coupling and decoupling of the components to/from the execution processes, thereby facilitating comparison of alternative configurations of the components or different implementations of the modules. The components can be local to the user's machine (e.g., Groovy scripts) or remote (e.g., REST services), and all can be extended without modification to TEXUS itself.

As mentioned above, the TEXUS components are accessible through manual programming/scripting by skilled end-users to develop a custom table extraction process. However, it is also possible to utilise the services via a well-established service composition tool. Amongst the possible options (JOpera, Bonita, Activiti [JOp, Inca, Act] to name a few) we have chosen the scientific workflow engine Taverna [Tav, WHF+13] for its active research community support.

Using Taverna, one can incorporate services without coding. A TEXUS user can define the data flows between the services, without having to worry about the technical details of the services (i.e., how to invoke them). The Taverna Workflow Engine then automates the pipeline processing of the data. It is also possible to supply custom data conversion/transformation modules in Taverna in case users want to

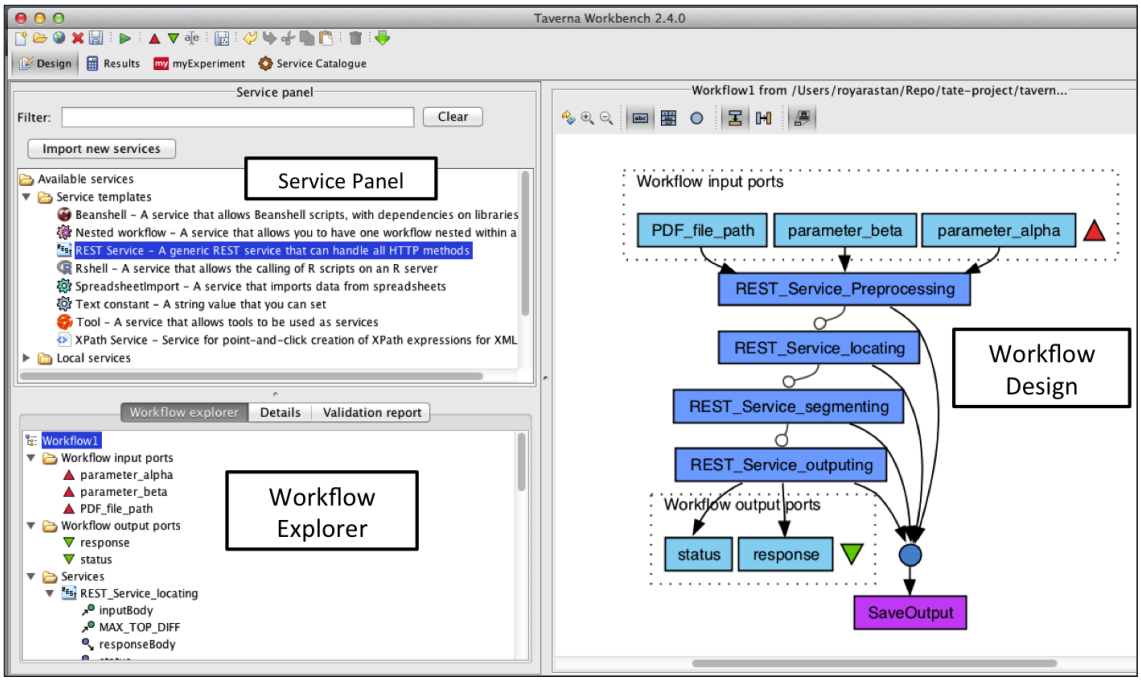


Figure 3.11: A Sample of TEXUS Workflow in Taverna Workbench

feed the output of an arbitrary program into the TEXUS services.

To complete the system, we have linked Taverna Workbench to Taverna Workflow Engine. As shown in Figure 3.11, The workbench allows the REST services to be dragged into the design panel and be configured based on the its URL template and HTTP method. Once placed in the design panel, the service's input/output ports can be defined in the design panel and viewed in the explorer panel. After specifying the required parameters of the workflow, the user can see the output and trace the execution step by step in the results tab. The output of each service as well as the final output of the workflow can be viewed and saved separately. The only customisation required to use the workbench was to implement our own PDF file reader for the pre-processing task. The rest of the components are utilised as they are designed and implemented.

We have also provide access to the services through a simple web interface and

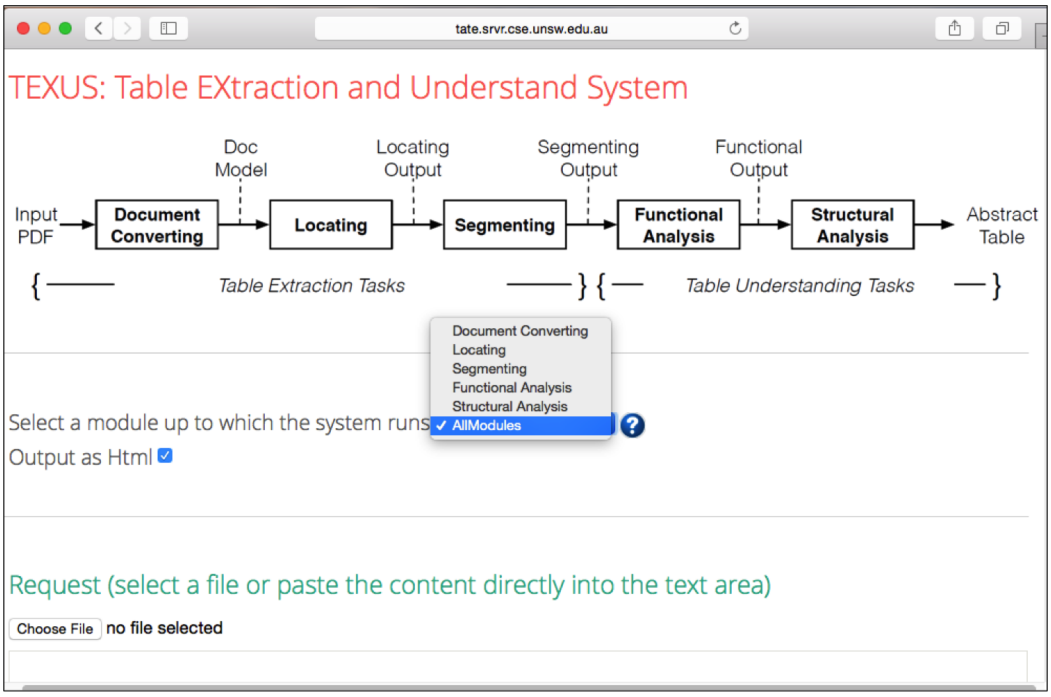


Figure 3.12: The UI to Access the TEXUS Pipleline

give the users the opportunity to run the system up to any services they tend to. The intermediate results can be saved in the XML format for further use or any evaluation purpose. They can feed the system their input in each step by relying on the standard XML interface provided. Users can specify to receive the final result in the HTML format as well. Figure 3.12 shows this simple UI for the users interaction.

3.4.2 System Implementation

We have implemented six components: document converting, locating, segmenting, functional analysis, structural analysis and finally a visualisation component that renders the output of any given component. In this chapter we review all the components. However, explain the implementation of locating and segmenting in details. Document converting component will be described under the title of PDFWrapper system in Chapter 4. Table Extraction sub-system is explained in Chapter 5 and

Table understanding sub-system will be detailed in Chapter 6. Here we briefly explain the implementation of the components.

Document Converting. The purpose of this component is to create an instance of our document model. We first convert a PDF document to an XML document to partitions the document into a sequence of `<page>`s, where each page is presented in `<pageColumn>`s and each pageColumn is a sequence of `<textChunk>`s, and each `<textChunk>` is an XML `TEXT` element with the following attributes: `Content` (text string, text type and text semantic concept); `Coordinates` (top, left, height, width of the text chunk); `Format` (font family, color, size and face).

We then perform further optimisation on the XML before producing output to make further table processing easier: Multi-line table cells detection, Page columns and Table columns differentiation, Line identification, Bullet and indentation Detection.

Locating Component. Taking an instance of TEXUS document model as input and detects `<table region>` in the document with specified `<upper boundary` and `<lower boundary>`. We cluster the lines in the document into `<table line>`s or normal text lines. Based on the heuristic methods we use the transitions from text lines to (potential) table lines, and vice versa, to determine table boundaries. A sequence of similar table lines based on type and spatial patterns form a potential table region.

Segmenting Component. In this component, the main aim is to detect the inner boundaries of the table as cells, rows and columns. Dominant table line pattern determines table `<row>`s in a table region and their spatial pattern, form table `<column>` boundaries basis. Consequently `<cell>`s boundary can be detected after rows and columns boundaries finalisation. We then tagged different type of cells (e.g. spanned, blank).

Functional Analysis Component. In this component, the goal is to detect each

cell function as <data>, <header> or <access>. In our system a bottom-up and a top-down classification algorithm is used at the same time to detect the boundary of table body containing data cells and the table box head enclosing the header cells. The functional analysis component adds function attribute to each <td> element of a table row <tr>.

Structural Analysis Component. Having detected the function of cells in the table by functional analysis in our system, the structural analysis component provides <access path>s for each data cell encompassing one unique path from each table <category> which originate from a header or access cell in the boxhead or stub and terminates to a <leaf path cell>.

Visualisation Component. This component shows the output data models in a convenient format. It parses the XML output of a component and provides an HTML representation of the content. We used color coding to present different functional regions in the table and also a tree representation of the access and header paths for structural analysis. And also it provides the CSV presentation of the final Abstract table. Details of this component will be described in Chapter 7.

3.5 Conclusion

This chapter presents a precise and comprehensive model for describing end-to-end table processing systems. The individual steps in the process are defined via a set of layered models. We demonstrate the utility of TEXUS by showing how it can precisely describe the inputs and outputs of individual components. We also showed that it can provide a basis for a generic table processing architecture and implement a prototype which includes the first components of end-to-end table processing and is layered on top of a generic scientific workflow engine.

There are some of the drawbacks and gaps in the current systems which can be addressed by employing our system design concept.

- *Domain dependent design issue*: Most of the successful systems in table processing follow domain-specific design and implementation. For example, *TableSeer* [Liu09], as a complete toolkit for table extraction, is only applicable to scientific papers. It can neither be used in other contexts nor extended easily. When the design is based on a set of well-defined, fine-grained tasks and a plug-and-play component-based paradigm like TEXUS, the system can be relatively easily adapted to add new tasks or re-use/re-purpose existing tasks to suit the needs of different domains.

- *Black-box system design issue*: While a black-box approach to system design may lead to a single integrated package (e.g. *pdf2table*), the fact that the system internals are not visible makes adapting or extending the system almost impossible. In TEXUS, we promote glass-box design where a standard breakdown of functional elements, along with well-defined interfaces, is visible in the system. This should help any enhancements and re-engineering tasks.

- *Ambiguous goals and processes issue*: The lack of standard "language" for specifying the main table-processing tasks, their goals and contained processes cause ambiguity in describing the scope of a table processing system. Take the table extraction task as an example; some research has considered it as locating and segmenting tables into cells [WCM06], while other work includes differentiating between data and header cells as well [PMWC03]. TEXUS aims to provide a standard definition of the entire process and defines it precisely via the TEXUS data models.

Chapter 4

PDF Wrapper System for Table Processing Purpose

This chapter explains the details of the design and implementation of the first component in TEXUS pipeline. Figure 4.1 shows 'Document Converting' in TEXUS pipeline. This component in our implementation is wrapped as a REST service which receives PDF documents as input and presents the document elements in our proposed Table-Oriented Document Model. In the following, we detail the challenges

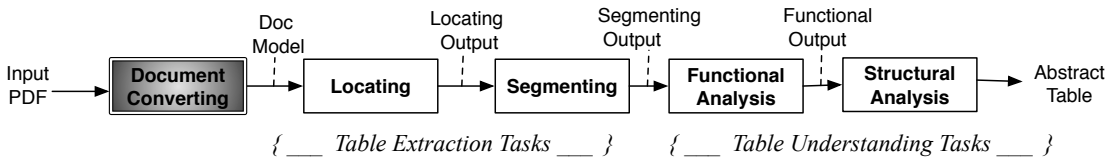


Figure 4.1: Document Converting component in TEXUS pipeline

in the task of document converting, and what other technologies and solutions have been developed in this regard and highlight our contribution in converting PDF documents.

4.1 Motivation and Background

Tables in documents are a rich and widely-available source of inter-related data. It would be useful if their contents could be automatically extracted and manipulated by computers. However, the task of automatic extraction of tables still remains a challenging and difficult one, and a large set of these potentially useful data sources is still manually handled.

One of the better understood reasons for this difficulty is attributed to the nature of aesthetic-focused and free-style presentation of tabular data, especially in business documents.

However, another key inhibitor is the fact that the tables are embedded in various document formats including semi-structured or unstructured ones. The vast majority of them are a sub collection of Microsoft (e.g.,.doc, .docx, .ppt, .pptx), PDF or HTML. Each type of document contains different internal representations (e.g., plain text, XML, binary) with different approaches for rendering.

Therefore, many systems aiming to analyse text in documents incorporate a software component designed to give access to the low-level document objects. These type of software components, referred to as "text extraction tools", tend to be specific to their intended input document type (e.g., XPDF for PDF files).

For table processing systems, it is common to use an existing text extraction tools and process the output to build a *pre-processor* for further analysis to recognise or extract tables. There are a few pre-processors in the area, but they tend to ignore some features present in the low-level objects which could help more effective table processing [PS11].

Recently, there has been a shift to developing application-oriented (or goal-oriented)

pre-processing systems [FFT+08] that selectively choose and process objects that are deemed *relevant* for the downstream application.

We refer to these kind of systems as "*application-oriented wrappers*".

In fact, wrapping in the Information Extraction field is generally known as the process of extracting data containing information relevant to a specific application, and organizing the extracted data into a machine-readable format.

We focus on processing PDF documents as the input type, as PDF is the most common standard for *print-oriented* formats in textual documents. A PDF document is described by a *content stream* which is a sequence of graphical and textual objects. Those objects are located at precise positions inside the document pages and are, in most cases, untagged. Such print-oriented nature of PDF documents raises many issues that make wrapping from PDF documents challenging [MHON11].

We present the design and implementation of a PDF wrapper named `PDF2TableDoc`, an application-oriented wrapper whose purpose is to enrich the capability of table processing systems. For instance, tables can be described in different levels: physical, logical and abstract [Lon10]. Our wrapper helps capture the relevant atomic elements in documents so that it is possible to express every level of description for a table.

Based on the observation and analysis, we propose a document model that can capture the essential elements required by a table processing system. The model is a structured XML document which can be used in composition with any table processing system.

As mentioned before, with the help of many commercial or open source text extraction tools, the page objects can be extracted from PDF sources directly. Most of PDF understanding systems obtain the low-level document objects using these

tools first. Some of the well known text extraction tools that provide the low-level document objects are XPDF library, the Apache PDFBox library, PDFlib Text Extraction Toolkit (TET) [1], PDF2Text [2].

The major effort in pre-processing in wrappers goes to segmenting the text elements in a PDF page into regions containing text with similar attributes [PS11]. These attributes are used as key features with predefined heuristic-based methods for tagging the various segments.

There are three main approaches to analyse the PDF document structure: top-down, bottom-up, or hybrid of the previous two[Mar08]. The top-down approach is suitable for the cases that we know the PDF document structure and the page model in advance. Having the whole PDF document page, this approach segments it into smaller objects based on paragraph breaks or inter-column gutters in an iterative manner. The segmenting procedure stops when some criterion is satisfied. [HL14].

If there is no predefined page template, a bottom-up method is adopted. Bottom-up algorithms usually start from the low-level document objects (e.g., the characters) and merge them into larger objects such as words,lines or zones [Anj01].

Various techniques are used in the literature to understand document content and structures. Since most of the text extraction tools provide low-level objects information, the bottom-up methods are mostly applied. As mentioned in [HL14] most of the systems following the bottom-up approach, go through three main steps :

- Step 1: Accessing to the low-level objects in the PDF. For this purpose, various PDF extraction tools are available to be used for extracting the PDF document atomic objects [Som04, LBG11].

[1]https://www.pdflib.com/products/tet/

[2]http://www.pdf2text.com

- Step 2: Merging low-level object to form compound object such as creating words, lines, paragraphs and other layout structures by relying on geometric and formatting attributes [Nur13, CL14, WPH04, JN08].
- Step 3: Detecting logical objects in the document. Usually the logical objects (such as tables, lists, scientific charts, etc.) are identified based on the simple and compound objects extracted in previous steps [LMGB06, KLU14].

We follow the similar first two main steps to process a text extraction tool output. However, at every step, the guidelines for the decision making in object extraction and segmenting are informed by our need to enhance the table extraction/understanding capabilities. The final output of the wrapper then is mapped to our own document model which is suitable for table processing.

4.2 Preliminaries

4.2.1 Portable Document Format

Portable Document Format (PDF) is a document format which is independent of the operating system and can be viewed on any computer. It has played an important role in the information storage and transmission in many domains.

PDF is based on the Postscript page-descriptive language which describes the pages of a document using objects (numeric, boolean, string, name, array, dictionary, stream) [Incb]. The text content of a PDF document is described through *Text String* objects. Every text string (a character or sequence of characters) is described by its font attributes through a *text state* function. Text state considers each text string as a *Bounding Box (BBOX)* with certain positioning in the page and its related glyph.

While a character is an abstract symbol, a glyph is a specific graphical rendering of a character specification. Glyphs are organised into fonts. Therefore it is the font that defines the glyphs for a particular character set.

There are some metric and positioning parameters for glyphs in PDF documents. The glyph bounding box is the smallest rectangle (oriented with the axes of the glyph coordinate system) that just encloses the entire glyph shape. The bounding box is expressed in terms of its left, bottom, right, and top coordinates relative to the glyph origin in the glyph coordinate system. A glyph's width (formally its horizontal displacement) is the amount of space it occupies along the baseline of a line of the text that is written horizontally. The glyph coordinate system gives the space in which an individual character's glyph is defined. The glyph origin is the point (0, 0) in the glyph coordinate system. For each BBOX in a PDF file, state array contains four numbers giving the coordinates of the left, bottom, right and top edges. Figure 4.2 shows a gylph with its metrics and positioning on a page.

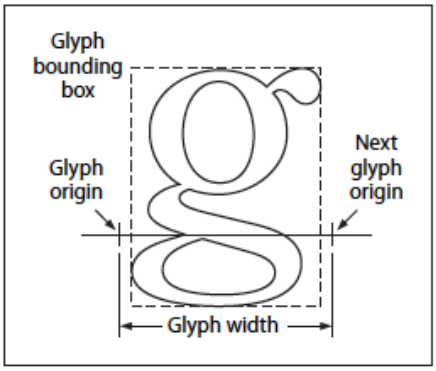


Figure 4.2: Gylph metrics and positioning

4.3 Wrapping PDF for Table Processing

To process text tables, text extracting tools for PDF such as XPDF[3] or PDFBOX[4] are commonly used to obtain the low-level document objects. Then, based on the output, the wrappers work further to segment the elements into regions containing text with similar attributes.

The information obtained with the help of these tools can be divided into two categories: the text content and the text style. The text content refers to the text strings; The text style includes the corresponding text attributes: the font, the size, line spacing and colour, etc; The text streams extracted from PDF files may correspond to various objects: a character, a partial word, a complete word, a line, etc. In addition, the order of these text streams does not always correlated with the reading order. A word reconstruction and a reading order re-sorting steps are necessary in order to correctly extract the coherent text from a PDF file.

Although PDF has become the common standard for the distribution of electronic documents, the print-oriented nature of PDF leads to serious drawbacks for table processing.

Here, we discuss the main issues and requirements in building a PDF wrapper to enhance table processing results.

Lack of structural information Since PDF was originally designed for the final presentation of a document, most PDF documents are untagged. They contain very little or no explicit structural information about the content.

[3]http://www.foolabs.com/xpdf

[4]http://pdfbox.apache.org/

4. PDF Wrapper System for Table Processing Purpose

According to the literature, this creates difficulties in table processing applications as follows:

- Table boundary detection [LMG08b, OR09]: Since there is little or no markup for tables in most PDF documents (in comparison with HTML and XML), table region identification and boundary detection is not straightforward. To successfully extract data from tables in such documents, detecting the table boundary is a crucial step.

- page column vs. table column [HB05, FGB+11]: One of the issues in table detection in multi-column PDF documents is differentiating a page column from a table column. Since page columns are not stated explicitly in PDF, various indications (e.g. graphical lines, white space) have to be detected and analysed prior to table spotting to properly recognise page columns in a document.

- Multi-line cell detection [ELN06, FMTG12]: Since the text elements are described based on their positions on the page and there are no tags for higher level elements (e.g. table cells), finding the logical relations between texts is another challenge, especially when multiple text lines must logically form a single table cell.

Implicit information about formatting and styling In most of the cases formatting and styling could convey information about the logical relationships between the texts. For example, the font style and size for headers are normally different to paragraphs. Bullets are used to show hierarchies. There are aspects in table processing that could benefit from having explicit access to this kind of information.

- Hierarchical header detection: In some tables, the hierarchy of row headers is shown with paragraph styling such as bullets or numbering or indentation

[RPSH16]. Having access to this information is necessary for better table understanding and tabular data relation extraction [SN13].

- Formatted headers: In some tables, the hierarchy of table headers is marked by differences in the font sizes and styles [AL12]. Again, these features should be preserved by the pre-processor for further processing.

Rendering order problem The main advantage of PDF documents is to preserve the original document presentation. As a user who views the document, the final PDF layout and content is more important than the process through which the page is produced.

Many different PDF generating systems can generate the same document appearances and rendered effect on the screen. However, they may follow different rendering order. A given PDF document might be structured to write out text a word/character at a time, or render specific types of text (e.g., footnotes) first. Very often, the rendering order is totally different from the 'reading order'. For multi-column text, the document is sometimes rendered across the columns by hopping the inter-column gutter.

Although the lack of standard rendering order does not affect the PDF document displaying and reading, it heavily impacts the performance of document structure analysis and understanding. Specifically, it affects the table locating task which relies on relative positions and sequence of text objects in the page [LBMG09].

In `PDF2TableDoc`, we aim to resolve these issues. For example, we incorporate purposely designed algorithms to detect page columns, logically related text lines, etc. This enhances the performance of the table locating and segmenting tasks. We perform detailed analysis of the styling features to accurately recognise them for future use in the processing pipeline. This enhances the performance of the functional

and structural analysis tasks. The design and implementation of the `PDF2TableDoc` is underpinned by our own document model suitable for table processing which is introduced in Section 4.4.

4.4 Table-Oriented Document Model

In this section, we introduce our document model which becomes the target schema of `PDF2TableDoc` in terms of extracting *table processing document elements*.

As described before, a PDF document is represented by a series of objects and their associated structure information. Therefore, to capture the necessary elements to facilitate further table processing, we focus not only on the text content of a PDF document but also on formatting and layout features.

Figure 4.3 shows the hierarchical structure of the elements in our proposed document model. *Text Chunk*s are the basic elements of a PDF document. A Text Chunk is an atomic object (i.e. textual element such as a character, a word or a sentence), which is totally contained within a document page. The graphical representation of a Text Chunk on the page layout takes up a certain room delimited by the Text Chunk bounding box.

In order to formally define the Text Chunk for our model, we rely on the *Text String* Object introduced in the PDF reference document [GW06]. We also consider and define the features that highly applicable to our table processing pipeline (such as font attribute and text styling features). To illustrate the elements of our document model, we use the sample document shown in Fig. 4.4. For example, we have highlighted some of the Text Chunks as Ch_i).

Text Chunk: A `Text Chunk` Ch is an instance of a *Text String* with certain bound-

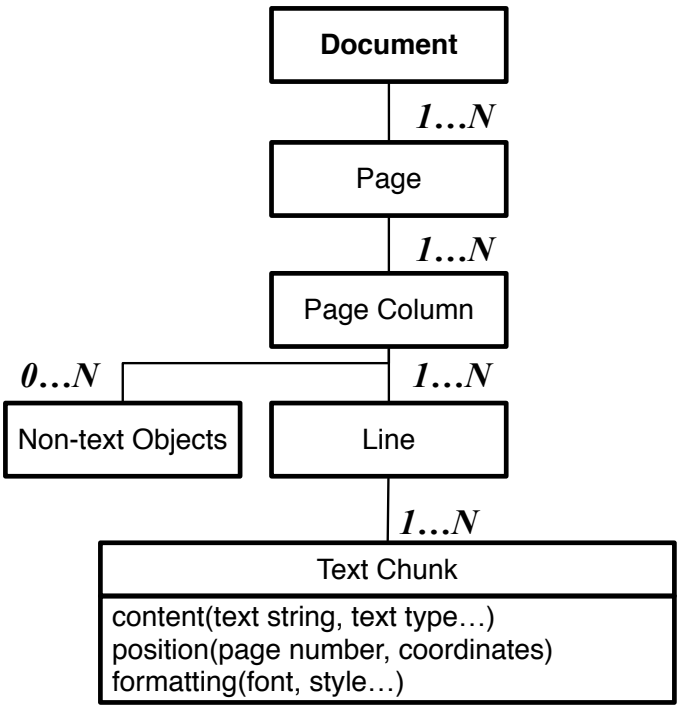


Figure 4.3: Elements of table-oriented document model

ing box and font specifications. We define a A `Text Chunk` as a tuple $\langle C, S, F \rangle$, where:

- C contains the information about the content of the `Text Chunk`.

  $C = \langle Ts, Tt, Tc \rangle$, Ts shows the text string, Tt represent the generic type of the string (e.g. Numeric, Alphabetic, Alphanumeric, Date, Percent, ...) and Tc is to show the corresponding ontology/semantic concept for the text content, if applicable.

- S represents the spatial attributes of the `Text Chunk`.

  $S = \langle Top, Left, Height, Width, P \rangle$ to show the bounding box of the Text Chunk by determining its distance from the top and left of the page P and also the height and width of the Text Chunk.

- F represents the formatting attributes of the `Text Chunk`.

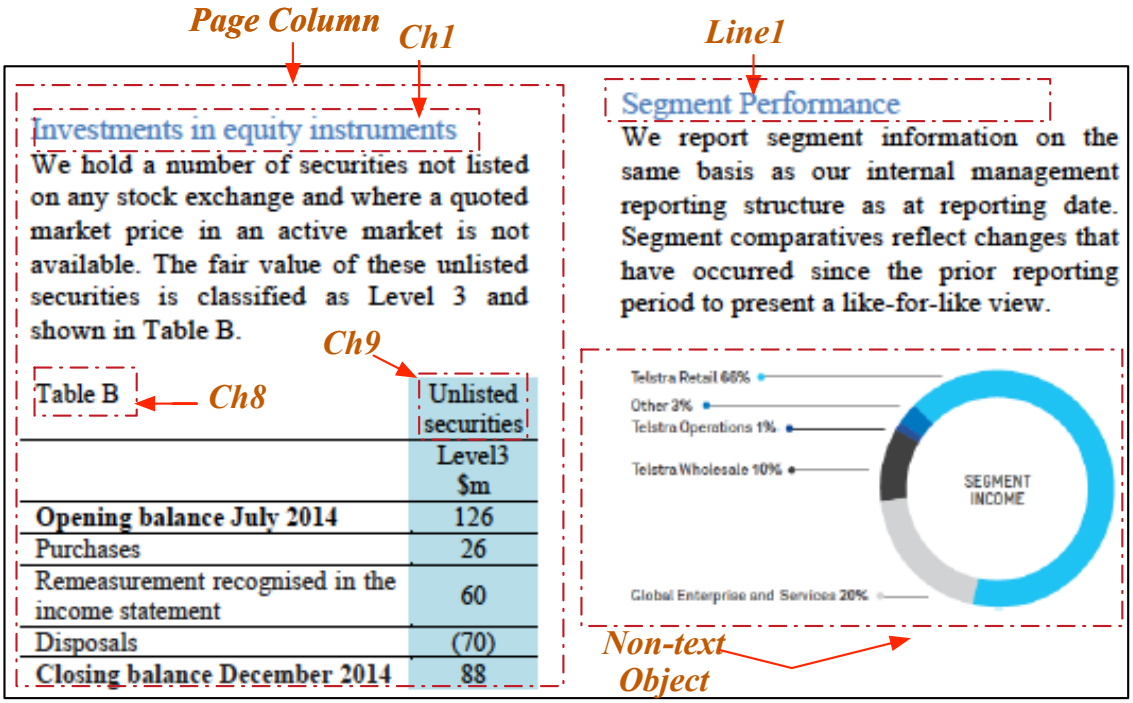


Figure 4.4: A sample of a document page

$F = \langle Font, Style \rangle$, where $Font = \langle Family, Color, Size, Face \rangle$ is to convey the font information and *Style* describes the layout styling of the `Text Chunk` (e.g. Bullet and Numbering).

Line: A `Line` L_j is a sequence of horizontally consecutive `Text Chunk(s)` $L_j = \langle Ch_1, Ch_2, ..., Ch_n \rangle$, terminated by an End-of-Line delimiter. The coordinates of the bounding boxes for Ch_1 and Ch_n are used to determine whether the `Text Chunks` are horizontally aligned.

Non-Text Object: Any element, such as an image, that is not composed of text is treated as part of a `Non-Text Object`. A $NTxO_j$ has a bounding box on the page. Since we are only interested in text tables, we do not retrieve information about such elements from PDF document, except the bounding box.

Page Column: A `Page Column` PaC_j is an ordered sequence of one or more `Lines` and zero or more `Non-text objects`. Page column boundaries can be detected by noting the resetting of the *Top* values of subsequent document elements.

Page: A `Page` P_j is an ordered sequence of `Page Columns` $P_j = \langle PaC_1, PaC_2, ..., PaC_k \rangle$, which terminates to an End-of-Page marker.

Document: A `Document` *Doc* is an ordered list of `Pages`, $Doc = \langle P_1, P_2, ..., P_q \rangle$.

In above definition, we assume that each table is contained within a `Page` and do not span multiple pages.

4.5 Design Overview of the Wrapper

The PDF document content stream lists all of the page objects, such as text, image, and path. Therefore in order to discover the logical components of a PDF document we should analyse and interpret the layouts and attributes of the page objects so as to correctly break or group them into different logical components.

Our design of `PDF2TableDoc` starts from a text extraction tool which provides the low-level information (characters, words, coordinates, etc.). Since we would like to build our own structure information for table processing, this tool gives enough information as our starting point.

There are many text extractor tools available "off-the-shelf". We have developed our system based on XPDF which is also used by other common PDF converter utilities (e.g. PDFtohtml, Greenstone) [PS11].

We designed `PDF2TableDoc`, which is the implementation of the "Document Converting Component" in the TEXUS pipeline (see Figure 3.1), as a web service which receives a PDF document as input and provides the all elements of our document

model. There are two sub modules operating inside `PDF2TableDoc`: `PDFtoXML` and `TableDocWrapper`. The PDF is passed to the `PDFtoXML` module which is implemented using XPDF as the core. XPDF utility reads the document characters along with their state function. `PDFtoXML` then merges characters to detect *Words* and *Text Chunks*. The list of text chunks will be passed to the `TableDocWrapper` module to identify more elements relevant to our document model. Figure4.5 shows the design of `PDF2TableDoc` as our document converting component.

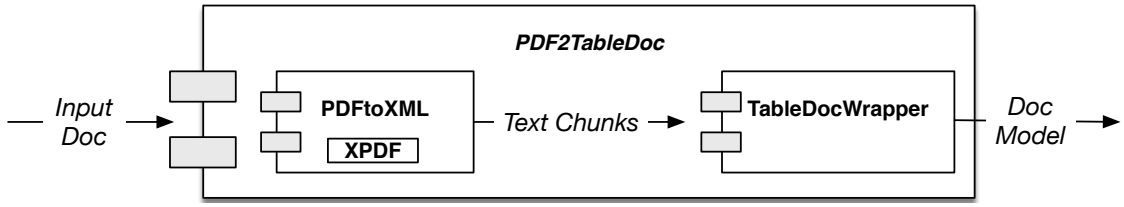


Figure 4.5: Document Converting Component:`PDF2TableDoc`

We represent the output of the document converting task in XML format. We chose XML because, if it is utilized correctly, it can be applied to create identical representations of the original documents and also can provide better search-ability and flexibility when analysing documents.

4.6 Implementation

In this section, we explain the steps undertaken in each sub module of `PDF2TableDoc`.

4.6.1 PDFtoXML

After receiving the document information at individual character level from XPDF, we take the following main steps to present the document at the Text Chunk level.

Word Detection

We rely on the BBOX information for every character in a PDF file, to form *words*. Algorithm 1 describes the main steps in detecting words from characters. AddChar() takes inputs, text state, x y coordinates (Top, Left), width and height of the character, the character (in Unicode) and the length of the character (in Unicode). First, we actualise the coordinates based on the text state parameters (character space and horizontal spacing) to be able to compare the character coordinates with the current string (Line 2). Then, we check if the text direction of the character is the same as the direction of current word. If they are not in the same direction we consider the newly seen character as the start a new word (Line 3-5). Otherwise, we check if the character is horizontally close enough to the current word to be considered as part of that word. Since the character is presented in Unicode its length may be more than 1 byte, therefore we consider the average width and length of the Unicode encoding of the character (Line 7-9) and then we calculate the horizontal distance between the character and the right most coordinates of the current word string (Line 12). If we see a whitespace character or the distance is bigger than a predefined parameter (α) we start a new word (Line 13-15) otherwise, we add the character to the current word string (Line 17). The reason we consider α is that sometimes the horizontal distance between characters is less than a whitespace. e.g. Ex.1 in Figure 4.10

Text Chunk Detection

The main aim of this step is to merge words represented as text strings based on the intersections between coordinates and the similarity in font attributes. we consider four positioning of horizontally consequent words in the page to be merged as text chunks.

Algorithm 1 addChar(textState state,x_1, y_1, w_1, h_1, Unicode char, int charLen)

```
1: n = curStr.len
2: actualise coordinates (state)
3: if (char.getDirection != curStr.dir) then
4:    endString()
5:    beginString(state, NULL)
6:    return
7: end if
8: if  (charLen ≠ 0) then
9:    w1 /= charLen
10:   h1/= charLen
11: end if
12: for (i=0, i ≤ charLen, ++i) do
13:   gap = x1 + i * w1 − curStr.xMax
14:   if (char[i] = ' ') —— (gap >α) then
15:      endString()
16:      beginString(state, NULL)
17:      return;
18:   else
19:      curStr.appendChar(state, x1 + i × w1, y1 + i × h1, w1, h1, char[i])
20:   end if
21: end for
```

- str_1 is `vertically lower than` str_2 (Figure 4.6-a):
  $(str_2.y_{Min} \geq str_1.y_{Min}$ & $str_2.y_{Min} \leq str_1.y_{Max})$.
- str_1 is `vertically higher than` str_2 (Figure 4.6-b):
  $(str_2.y_{Max} \geq str_1.y_{Min}$ & $str_2.y_{Max} \leq str_1.y_Max)$
- str_1 is `entirely contained by` str_2 (Figure 4.6-c):
  $(str_2.y_{Min} \geq str_1.y_{Min}$ & $str_2.y_{Max} \leq str_1.y_{Max})$
- str_1 `entirely covers` str_2 (Figure 4.6-d):
  $(str_2.y_{Min} \leq str_1.y_{Min}$ & $str_2.y_{Max} \geq str_1.y_{Max})$

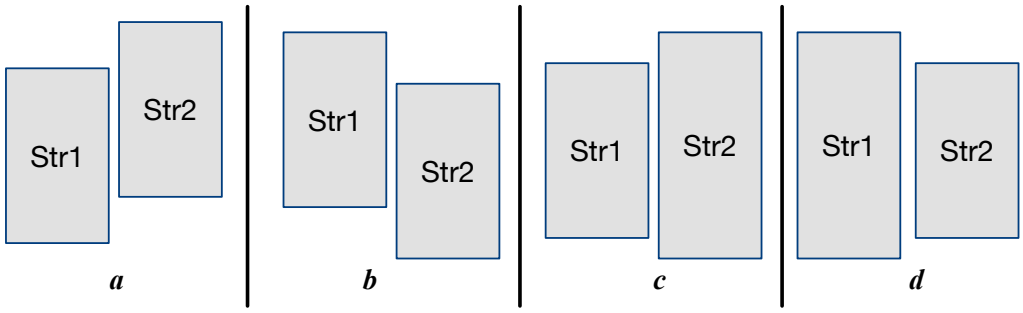


Figure 4.6: vertical positioning of horizontal consequent words

Algorithm 2 lists the details of *formTextChunks()* which takes two strings and performs a merge if one of the positioning checking rules matches.

4.6.2 TableDocWrapper

XPDF reads characters in the raw order and the `PDFtoXML` module merges them to form text chunks preserving the raw order. However, text sequence error is still a common problem in the existing text extraction tools. This means the extracted text chunks follow a different sequence from its original appearance in PDF documents. If such a text sequence error happens in the table contents, it could generate wrong

Algorithm 2 $formTextChunks(Str_1, Str_2)$

1: **while** (Exists(str_1) & ($str_2 = str_1$.Nextstring)) **do**
2: height= $str_1.y_{Max} - str_1.y_{Min}$
3: horSpace = $str_2.x_{Min} - str_1.x_{Max}$
4: **if** (!addLineBreak) &($str_1.x_{Min} - str_2.x_{Min} < \alpha$) **then**
5: vertSpace = $str_2.y_{Min} - str_1.y_{Max}$
6: **end if**
7: **if** (str_2 is vertically lower than str_1) **then**
8: vertOverlap = $str_1.y_{Max} - str_2.y_{Min}$
9: **else if** str_2 is vertically higher than str_1 **then**
10: vertOverlap = $str_2.y_{Max} - str_1.y_{Min}$
11: **else if** str_2 covers str_1 entirely **then**
12: vertOverlap = $str_1.y_{Max} - str_1.y_{Min}$
13: **else if** str_2 is entirely contained by str_1 **then**
14: vertOverlap = $str_2.y_{Max} - str_2.y_{Min}$
15: **else**
16: vertOverlap = 0
17: **end if**
18: **if** ((vertOverlap >0.5 × height) & (horSpace <β) & (0 <vertSpace <0.5 × height) & (str_1.dir = str_2.dir) & (str_1.font = str_2.font)) **then**
19: str_1.append(str_2)
20: **end if**
21: **end while**

results during the table detection, such as segmenting a single table into several pieces, wrong column or row information, and omitting cells or missing a whole table.

In this section, we review the steps after receiving the output of `PDFtoXML` to transform it into our target document model. The main task of `TableDocWrapper` module is to go through the text chunks produced bu `PDFtoXML` and analyse/detect the relevant elements of the target document model.

Related Text Merging

One of the common cases in tables is the existence of *Multi-Line Cells*. In which multiple text chunks form a logically single table cell. These text chunks however may appear in different lines Since we have the text chunks in raw order, we have a chance to detect these cases before sorting the text chunks based on the coordinates in the page.

We consider each text chunk as a rectangular object represented by its four coordinates attributes and font specifications. Every vertically consequent text chunk which is in the close top distance from the page is compared to see whether there is any horizontal intersection between them or not.

Figure 4.7 shows different scenarios of horizontal intersections between text chunks and how the merged text chunks are actualised.

In order to make the merge more accurate, we check if the text chunks share the same font and also the same alignment. Algorithm 3 shows the detail of this merging. It receives two text chunks (the last seen text chunk lCh and a newly seen text chunk nCh), α as a predefined threshold parameter for calculating the vertical closeness of the text chunks and β a threshold for calculating the horizontal intersection between

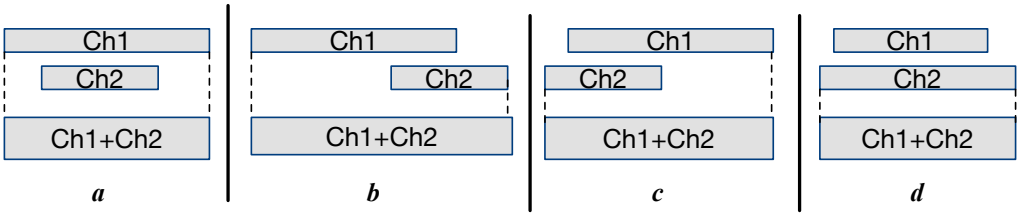


Figure 4.7: Different interval intersection to merge text chunks

the text chunks. We set some variables for the coordinates and font information of nCh and lCh in Line 1 to Line 9. Then in Line 10 we check whether the lCh and nCh are in the acceptable vertical distance to be considered as potential cases to be merged. If so we check their horizontal intersection between them and their font and alignment compatibility in Line 11. Provided all checking is satisfied, text chunk will be merged and the attributes (coordinates and font) will be updated for the new merged text chunk (Line 12-13).

Page Column Detection

As the next step, we mark page columns. Since the text chunks are created in the raw order, the texts in a page column are supposed to appear sequentially (in reading order). The start of a new page column can be detected by a large decrease (bigger than a predefined threshold α) in the `Top` value in the following text chunks. Algorithm 4) shows the detail of page column detection. For all text chunks in the page, we compare the top difference between two consequent text chunks and if the decrease in the distance between them is more than α we add a new column to the page (Line 8-12). Otherwise the attributes of the page column get updated based on the newly seen text chunks. The page column gets the top attribute from the first text chunk in the column boundary, the left most text chunk form the left attribute and the height of the page column is the coming from the differences of the top

Algorithm 3 *mergeTextChunks*(Text chunk nCh,Text Chunk lCh, α, β)

```
1: ttop= nCh.Top()
2: t_l top =lCh.Top()
3: t_l height = lCh.Height()
4: tleft= nCh.Left()
5: twidth= nCh.width()
6: t_l left =lCh.left()
7: tfont= nCh.font()
8: t_l width =lCh.width()
9: t_l font =lCh.font()
10: if (t_l top <ttop) & (ttop ≤ (t_l top + t_l height + α) then
11:    if isIntervalContains(t_l eft, t_w idth, tl_l eft, tl_w idth, β) & (tfont =t_l font) &
       (talignment=t_l alignment) then
12:       lCh.setValue(lCh.text() + nCh.text())
13:       updateNodeAttributes(lCh, nCh)
14:    end if
15: end if
```

between the first and the last text chunk in the page column boundary.

Algorithm 4 *detectPageColumns*(Page P, α)

```
1: formerTextChunkTop = 0
2: currentColumn = newPageColumn(P, "pageColumn")
3: ArrayList Columns = new ArrayList¡pageColumn¿()
4: Columns.add (currentColumn)
5: chNumber= P.Chunks.Size()
6: for (i = 0, i <chNumber, ++i) do
7:    if (P.Chunks[i].top <formerTextChunkTop) & (formerTextChunkTop -
      P.Chunks[i].top >α) then
8:       currentColumn = newPageColumn(P, "pageColumn")
9:       Columns.add (currentColumn)
10:      currentColumn.setAttribute()
11:      formerTextChunkTop = P.Chunks[i].Top()
12:   else
13:      currentColumn.updateAttribute()
14:   end if
15: end for
```

Line Detection

After detecting the page columns, we mark lines in each column. The main idea of the line formation is to consider the text chunks in the same top distance in each page column belong to the same line. Therefore, Algorithm 5 receives a page column and a predefined threshold parameter (β) to check the top distance of the text chunks in the page column. Since we have text chunks in raw order and also we have merged some text chunks in `Related Text Merging` (introduced new coordinates for the merged text chunks) we need to sort the text chunks. Line 4 and 5 sorts all text

chunks in the page column first based on the `Top` and then `Left` attributes. Starting from the first text chunk in the page column if the vertical distance of the following text chunk and the former text chunk is bigger than a threshold, it indicates the start of a new line (Line 7-13). As we create a new line we update its coordinate attributes by considering its contained text chunks (Line 15).

Algorithm 5 *buildLinesInColumn*(Page Column *column*, β)

```
1: formerTextChunkTop = -1
2: ArrayList Lines = new ArrayList(Line);
3: chNumber= column.Chunks.Size()
4: for (i = 0, i <chNumber, ++i) do
5:    sortChunks(column.Chunks[i],Top, Left)
6: end for
7: for (i = 0, i <chNumber, ++i) do
8:    if (column.Chunks[i].Top() >(formerTextChunkTop + β) then
9:       currentLine = new Line(null, "line")
10:      currentLine.setAttributes()
11:      currentLine.append (column.Chunks[i])
12:      Lines.add (currentLine)
13:      formerTextChunkTop = column.Chunks[i].Top()
14:   else
15:      currentLine.updateAttributes()
16:   end if
17: end for
18: for (Line n:Lines) do
19:    column.add(n)
20: end for
```

Bullet Detection

Most of the current PDF wrappers are unable to detect the detailed formatting of the text elements in the PDF file, such as bullets or numbered text elements. Since these formatting information can convey important hints about hierarchical structure of the content it is useful to properly recognise them in a table processing system [SN13, RPSH16]. Algorithm 7 shows the bullet detection for an input text chunk. It gets the first character in the text chunk and checks whether its corresponding Hexadecimal number belongs to a list of bullet numbers (Line 4-7).

Algorithm 6 shows the processing of bulleted and numbered text chunks. It receives a page column and go through each text chunks in each line of the page column to detect the numbered and bulleted text and add the style attribute to the text chunk (Line 3-6). The pattern for a numbered text is described in Line 1.

Algorithm 6 *processBulletAndNumbered*(Page Column *column*)

```
1: numberedTextPattern = ‘ ∧ [(]?([a − zA − Z]|[0 − 9]+)[.\\− :])′
2: Pattern p = Pattern.compile(numberedTextPattern)
3: for each line ∈ pagecolumn do
4:    for each textChunk ∈ line do
5:       if (textChunk!= null)&(startsWithBullet(textChunk) then
6:          textChunk.att.add(‘style’, ‘Bulleted’)
7:       end if
8:    end for
9: end for
```

Algorithm 7 *startWithBullet*(String *text*)

```
1: if (text = null) —— (text.length() ≤ 0) then
2:     return false
3: end if
4: char c = text.charAt(0)
5: String s = Integer.toHexString(c — 0 × 10000)
6: if (s.equals('12022') or s.equals('100b7') or s.equals('100a1') then
7:     return true
8: end if
```

4.7 Evaluation

We present two scenarios for the evaluation, first to compare the functionality of our core module PDFtoXML and second to evaluate the performance of TableDocWrapper module. For this purpose, we use the known measures in Table Processing community [GHOO13].

- Completeness and Purity [GHOO13]: A table is classified as *complete if it includes all containing cells in the table region; a table is classified as pure if it does not include any cells which are not in the table region. A correctly detected table is therefore both complete and pure.*

- *Recall and Precision: we use the following definitions for Bullets and Numbering, Page column and Merged text. We refer to these as document elements.*

$$Recall_{Loc} = \frac{Number\ of\ Correctly\ Identified\ document\ element}{Number\ of\ the\ elements\ in\ the\ document} \tag{4.1}$$

$$Precision_{Loc} = \frac{Number\ of\ Correctly\ Identified\ document\ element}{Number\ of\ Identified\ element\ in\ the\ document} \tag{4.2}$$

We first evaluate the performance of our implementation regarding the Completeness and Purity on the total number of tables in the corpus. Then, for unsuccessful cases,

we record the results of the precision and recall values for each document. To report the final result, we then average the recorded values as 'Per-Document Average'.

4.7.1 Corpus Characteristics

To evaluate our implemented system, we used the dataset introduced in 'ICDAR 2013 Table Competition'. The dataset consists of 67 documents with 156 tables, and contains ground-truth for the locating and segmenting tasks. The tables are in different styles and from various domains. Since we focus on document converting performance evaluation, we report on the varieties influence the PDFWrapper performance which are:

- Merged texts in table cells
- Bullet and Numbering
- Number of page column

Regardless of these table characteristics, tables in ICDAR corpus show variety from the following perspectives [5]:

- Language of the text: Corpus divides the documents into two parts of English and European (mainly Italian)
- Direction of the text: in some cases the text direction is vertical (top to bottom) instead of horizontal.
- Fully Ruled Tables: Tables with border for all cells (both inner and outter table borders)

[5]These features are used for evaluation of other part of system, extraction and understanding in chapter 5 and 6

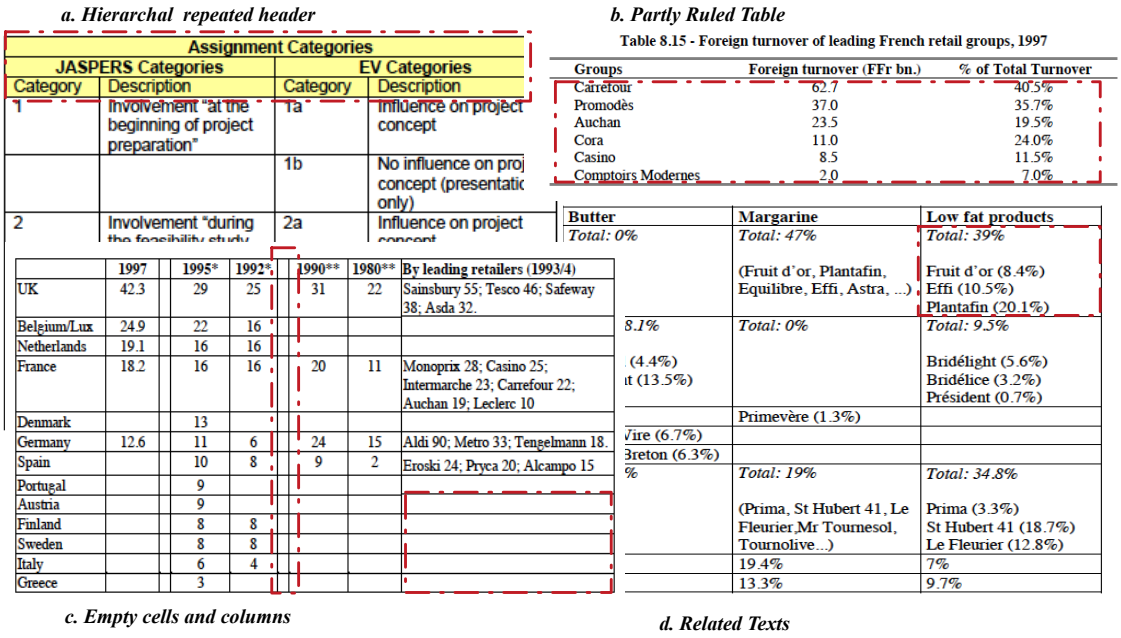


Figure 4.8: variety of tables in ICDAR corpus

- Partly Ruled Tables: All table cells do not have border.
- Spanned cells
- Hierarchical table header
- Tables with empty cells or columns
- Tables with header in the middle of the table
- Tables with special characters in the cells
- Float Tables
- Diagrams with repeated text patterns in

Figures 4.8 and 4.9 show some of the mentioned varieties in the ICDAR corpus.

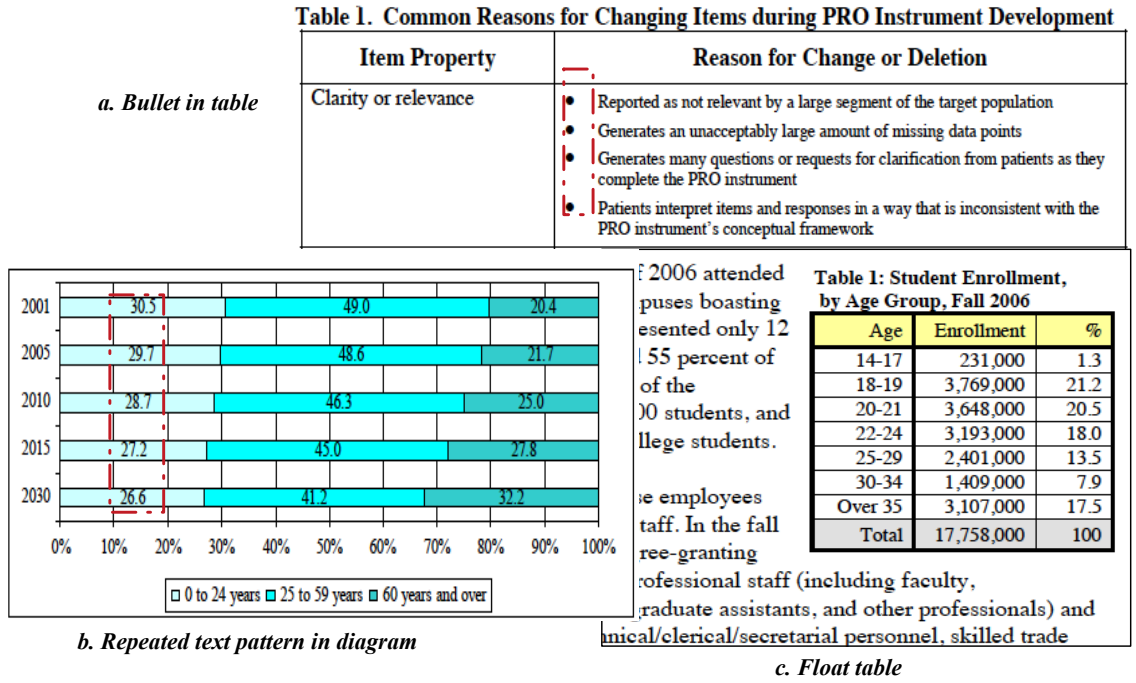


Figure 4.9: variety of tables in ICDAR corpus

4.7.2 PDFtoXML

First, we compare the functionality of our core module `PDFtoXML` with a similar, open system `PDFtoHTML`[6]. `PDFtoHTML` is a utility which is also developed based on XPDF and can be configured to create XML output and list text chunks from PDF documents along with their coordinates.

To compare the performance of the two utilities we run our `TableDocWrapper` module with the outputs from `PDFtoXML`, and also with the outputs from `PDFtoHTML`. In each run, we then feed the output of `TableDocWrapper` to our table extraction system to locate and segment tables. We then compare the locating and segmenting results of each run with the ground-truth.

Here, we are able to compare the performance of `PDFtoXML` and `PDFtoHTML` on

[6]https://sourceforge.net/projects/pdftohtml

their ability to accurately extract the text chunks, as more accurate extraction of text chunks leads to more accurate extraction of the final document model by `TableDocWrapper`.

Table 4.1 shows the evaluation results. We have compared the performance of two runs by utilising two measures:

- *Completeness* and *Purity*, measured for the whole document set (total number of tables 156). A table is considered as *complete*ly extracted if it includes all cells in the table region; and a detected table is called *pure* if it does not include any cells which are not in the table region. A correctly detected table is therefore both complete and pure [Sil07].
- *Recall* and *Precision*, measured specifically for the unsuccessful cases and reported in a cell-by-cell comparison with the ground-truth.

Table 4.1: `PDFtoXML` and `PDFtoHTML` performance comparison

Utility	**Per-document average**			**Whole DocSet #Tables=156**	
	Recall	Prec.	F-meas.	Complete	Pure
`PDFtoXML`	0.9971	0.9729	0.9848	142	148
`PDFtoHTML`	0.9644	0.9569	0.9606	138	130

Our own detailed study of the results showed that the main improvements obtained by `PDF2XML` is from how the gap analysis between characters is done during the word detection phase. `PDFtoHTML` seems to simply rely on a whitespace character to decide whether to merge characters or not. We check the size of the gap as well as the existence of the whitespace character. If the size is more than a threshold we do not merge characters into one words (Line 11-15 in Algorithm 1).

Figure 4.10 shows some tables that PDFtoHTML fails to detect as separate text chunks and merge them as one text chunk because of the small horizontal distance between them (marked by dotted lines).

Ex.1

Age(years)	Non-Hispanic white		Non-Hispanic black		Mexican American	
	Male	Female	Male	Female	Male	Female
2-11months	1,087,948	1,022,490	292,652	255,744	188,980	150,760
1-2	2,586,688	2,568,738	647,701	639,327	409,038	392,640
3-5	3,867,692	3,576,723	935,862	938,510	563,286	563,183
6-11	7,808,033	7,401,349	1,770,525	1,732,954	998,192	999,217
12-19	9,795,497	9,208,607	2,191,327	2,218,406	1,180,160	1,173,272
20-29	13,340,788	14,032,118	2,194,990	2,776,284	1,785,795	1,462,678
30-39	15,492,738	15,745,424	2,433,567	2,902,296	1,318,832	1,170,452

Ex.2

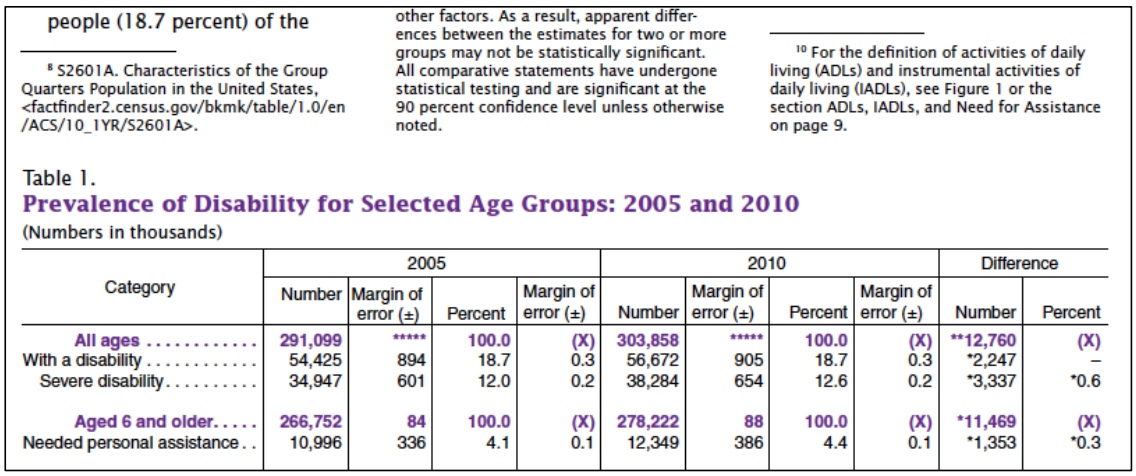
	Actual					
Region and state	2003–04	2004–05	2005–06	2006–07	2007–08	2008–09
United States	2,753,438	2,799,250	2,815,544	2,893,045	3,001,337	3,039,015
Northeast	485,670	503,528	521,015	536,697	552,289	552,973
Connecticut	34,573	35,515	36,222	37,541	38,419	34,968

Ex.3

Loans to nondepository Fin.Inst.	4,958,000	29.9	4,512,000
All other Loans	611,000	3.7	602,000
Total Gross Loans	16,604,000	100.0	14,871,000

Figure 4.10: Examples where PDF2HTML failed due to insufficient gap analysis

As pointed out, PDFtoXML failed in detecting some text chunks. In our analysis, this mainly occurs when tables "float" in the page or span across page columns. This means that the implementation is not able to relate the coordinates of the text chunks in the page correctly. An example of float table is shown in Figure 4.13. Another example is shown in Figure 4.11, which demonstrates a table spanning through three page columns.



people (18.7 percent) of the

[8] S2601A. Characteristics of the Group Quarters Population in the United States, <factfinder2.census.gov/bkmk/table/1.0/en/ACS/10_1YR/S2601A>.

other factors. As a result, apparent differences between the estimates for two or more groups may not be statistically significant. All comparative statements have undergone statistical testing and are significant at the 90 percent confidence level unless otherwise noted.

[10] For the definition of activities of daily living (ADLs) and instrumental activities of daily living (IADLs), see Figure 1 or the section ADLs, IADLs, and Need for Assistance on page 9.

Table 1.
Prevalence of Disability for Selected Age Groups: 2005 and 2010
(Numbers in thousands)

Category	2005				2010				Difference	
	Number	Margin of error (±)	Percent	Margin of error (±)	Number	Margin of error (±)	Percent	Margin of error (±)	Number	Percent
All ages	291,099	*****	100.0	(X)	303,858	*****	100.0	(X)	**12,760	(X)
With a disability	54,425	894	18.7	0.3	56,672	905	18.7	0.3	*2,247	–
Severe disability	34,947	601	12.0	0.2	38,284	654	12.6	0.2	*3,337	*0.6
Aged 6 and older	266,752	84	100.0	(X)	278,222	88	100.0	(X)	*11,469	(X)
Needed personal assistance	10,996	336	4.1	0.1	12,349	386	4.4	0.1	*1,353	*0.3

Figure 4.11: Table spans three page columns

4.7.3 TableDocWrapper

In order to evaluate the performance of the TableDocWrapper module, we categorise the documents in the corpus based on the key elements according to the document model (e.g., page columns, bullets, numbering) and then investigate the accuracy of the output at the detected element level. We compare the result with the ground truth documents reviewed and validated by two experts. Table 4.2 reports on the accuracy of *page column* and *bullet/numbering* detection as well as *related text merge*. The results are calculated per document and then the overall performance is shown by the average on the whole documents. Table 4.2 shows the results.

Table 4.2: TableDocWrapper Performance Evaluation

Element Type	Recall	Prec.	F-meas.
Merged text	0.91	0.95	0.93
Bullet and Numbering	1	1	1
Page Column	1	0.92	0.96

The system has 100% accuracy in detecting bullets and numbering in the documents.

Particularly, it successfully recognised a bullet point applied on a merged text chunk. Figure 4.12 shows one example of this.

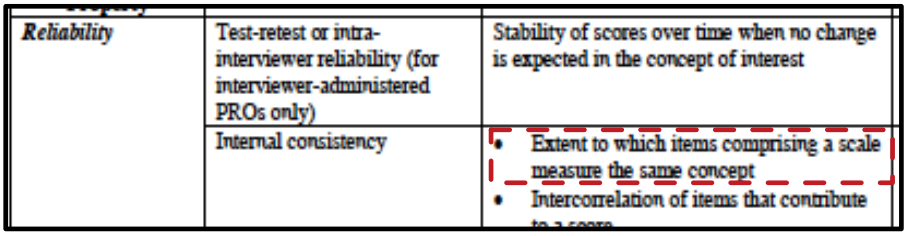


Figure 4.12: Correct bullet detection for merged text

The system generally performed well in detecting page columns. However in some cases, such as floating tables, the system reports false positive page columns. As shown in Figure 4.13, the floating table on the right caused the system to treat the page as a two-column page.

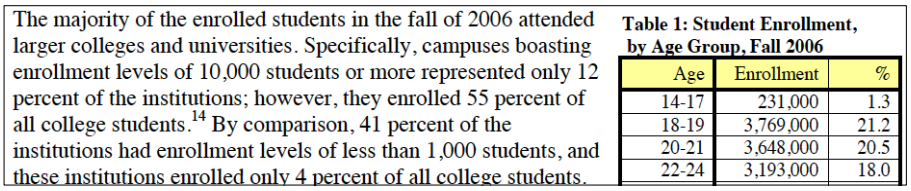


Figure 4.13: Incorrectly detected page column

For related text merging, the system achieved over 90% performance in all three measures. However, cases with irregular styling and spacing between text chunks result in poor performance. As can be seen in Figure 4.14, there is a vertical spacing between "Fraction of" and "Wealth Lost" which was slightly larger than the *vertical closeness threshold* and the chunks were not merged.

.

Percent	Mean Loss	Fraction of Wealth Lost
75.2	12196	17.4%
11.6	23518	22.5%

Figure 4.14: Failed to correctly merge related text

4.8 Conclusion

We presented `PDF2TableDoc` a PDF wrapper designed for table processing applications. We reviewed the specific challenges and issues for wrapping PDF with the focus on table extraction and understanding. We then presented the document model that captures (i) logical groups of objects such as page columns and lines, (ii) more complete text chunk metadata such as styling features (e.g., bullets, numbering) and (iii) merged text chunks for better recognition of table cells. We also presented the implementation of the `PDF2TableDoc` to produce the model. Our evaluation showed that using the `PDFtoXML` module resulted in better table locating and segmenting, compared to using `PDFtoHTML`. The `TableDocWrapper` module was able to detect important features such as page columns, bullets and numbering in all measures, recording over 90% accuracy.

This PDF wrapping approach underpinned by our model helps focus the task of the pre-processor on producing features that are most useful for table processing.

Chapter 5

Automated Table Extraction System

This chapter explains the details of the design and implementation of the Table Extraction part in the TEXUS pipeline. Figure 5.1 shows Locating and Segmenting components in the TEXUS pipeline. These two modules perform Table Extraction tasks in TEXUS. This sub-system receives a document model, which is produced by Document Converting module in Chapter 4, as the input and provides segmented tables as output.

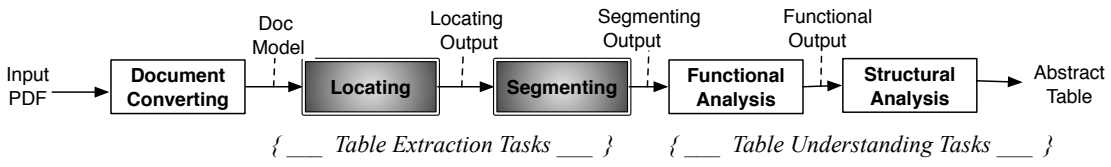


Figure 5.1: Table Extraction related components in TEXUS

In this chapter, we propose a standard and generic definition of table extraction tasks (locating and segmenting) and their related interfaces to enable developing reusable and repeatable systems regardless of the choice of implementation.

5.1 Motivation and Background

Table extraction is defined as a sequence of locating tables in documents and extracting their entries along with the arrangement of the entries inside the tables [Hur00a, eSJT06]. It is also refereed to as *Table Recognition* which contains table detection and table physical structure recognition [tBC04].

A large number of different approaches for table extraction have been described in literature and also there are lots of commercial systems that claim to deal with tables especially on images and unstructured documents. However, the problem of extracting tables is not yet solved [KLU14]. Each approach has its own strengths and weaknesses.

An extensive overview of the state-of-the-art in table extraction is given in [tBC04, EHLN06, KLU14]. The early approaches focused on the detection of table delineation (e.g. detecting line arts as the outer boundary of tables), large column spacing or already-known column headings. Thus, these have been applicable only to a small type of tables [LDC05, LMG08b, AG02].

In [KD05], a new approach, which cluster blocks and potential column candidates by considering the overlapping word segments in adjacent lines, is suggested. The blocks that satisfy predefined criteria are tagged as candidate table elements. The candidate table regions are then decomposed into rows and columns based on the alignment and overlap of individual cells.

There is a group of work which employed statistical learning approaches for table extraction problem. One approach is described in [WPH04], which classifies lines to table line and text line based on the word spacing. Then vertically adjacent lines with large gaps and horizontally adjacent words are grouped together to make table entity candidates. Finally, a statistical based learning algorithm is used to refine

the table candidates and reduce false detections. Machine learning algorithms and classification techniques are used commonly for table extraction task. In [LTSX06], a unified machine learning approach and Naive Bayesian classification are used to detect the unformatted table rows which, in conjunction with document structure information, form the bases for table extraction.

Since table extraction is the starting point in most of table processing systems, different approaches and techniques are proposed for specific document formats. Many systems deals with table locating and segmenting in HTML documents [WH02a, ETL05, KHG05, KL08, PCC12]. Text documents [eSJT03, LDC05, LTSX06, WWLN09] and image files [GDPP05, MCDC06, KGK+07] are other formats targeted by specific table extraction solutions. Different preprocessing steps are suggested for each format to make table detection easier.

Regardless of the variety of the document formats and techniques applied for table extraction, the various format styling and layout arrangement make table extraction more challenging. Most of the current systems are restricted to specific layout setting in the document for table locating. For example focusing just on scientific papers with a standard page setting and formatting [RPHB12, CPV13], single column pages [YKM05, OR09], or tables with proper captions and titles [WCM06, LBMG07a].

Systems that aim to process a wide range of possible layout options in tables invariably demand more human intervention to reach an acceptable accuracy, especially the location task. For example, systems like WNT [JN08] or Vericlick [NT12] require human interaction to produce the system output. Some other systems relied on semantic analysis [PCS05] or NER [LCZW10] techniques to improve the table extraction performance and consequently are more domain-dependent.

Putting all these together, there are many ways to implement a table extraction system which performs accurately for restricted document formats or pre-defined

table layout. In this chapter, we introduce our table extraction sub-tasks (locating and segmenting) data model to make them as generic as possible to enable developing repeatable and extensible systems. In our implemented system we also address the mentioned challenges for a table extraction system.

Here, we (i) define table locating and segmenting model, (ii) explain the details of our implementation of automatic and task-based table extraction system, (iii) evaluate our system on various table layouts from different domains and known corpus in the table processing community.

5.2 Preliminaries

5.2.1 Automatic Table Extraction

Considering the recent advances in techniques for processing unstructured documents and the vast use of data in more applications and domains, automatic solutions are contributed to facilitate the design and implementation of user-driven and ad-hoc systems [RYG12]. The idea of the text processing pipeline automation is used in many current language processing systems (e.g., GATE [Cun02]).

From the perspective of the automation, current table extraction systems are in two main categories:

- Semi-automated: These systems either need human intervention or rely on supervised and semi-supervised (adaptive) machine learning techniques to locate and segment table. The intervention could happen before, during or after table processing main steps. For example, in [FSCG03] manually labelled document regions are fed to train the system before table processing, In Vericlick

[NT12], a user intervention step after table segmenting assures the segmenting result is accurate before the next step is performed In WNT [Pad09], a GUI is provided for the user to check the result of header rows segmentation. The same strategy is used for Pdf2table [YKM05] where a user can check the final output of table extraction and correct the result by adding or removing rows and columns.

Although this approach provides the opportunity to make the output more accurate, it is not efficient for a large-scale table extraction task and requires the user to be able to modify or manipulate the system input or output, and be familiar with the domain.

- Fully-automatic: In this approach, systems are taking documents as input and provide the segmented table as the output. Although the intermediate output may be provided, the whole process runs without any human intervention. In some systems it is also referred to as *algorithmic* approach as it considers the task of table extraction in isolation from further usage of the extracted information.

  Although this approach is more scalable than the semi-automatic one, the accuracy could suffer when the input documents vary in formats and table layouts [GHOO13].

Also, in full-automated system, the scope of their functionality vary a lot. Most of the systems are domain-specific and do not provide the extensibility and repeatability of table extraction tasks for other domains and applications.

In TEXUS, we propose an automated approach to table extraction, but we aim to improve the black-box approach of the many automated systems by providing a modular and task-based design. The design facilitates a domain/application-independent implementation of table processing, as well as increasing the repeatability of the

process, and extensibility of the current developed systems. We also provide all the intermediate results in a standard data format for use in further analysis. These output can be modified by the user to make their correction and be fed into the system again.

5.2.2 Table-Oriented Document model

In this section, we explain how table extraction uses the table-oriented document model....

PDF2TableDoc provides us the document elements which are useful for table processing and it is then used in table extraction.

5.3 TEXUS-based Table Extraction

Table extraction goes beyond simply detecting where tables are in a document; it separates tables from the rest of the document and represents each of them in the form of a *Physical* abstraction model. Table extraction consists of two main processes: *Locating* and *Segmenting* and they are mostly described based on the layout attributes of the document and table.

Our system does not rely on user-specified examples, and does not require any interaction with the user during the process; this means that extraction is in a completely automatic way. The system has no a priori knowledge about the page contents either (e.g. schema based approaches).

5.3.1 Locating Model

The aim of locating is to find the starting line and ending line of each table in the document. The location of each table is identified by its boundaries. Lines in the input document are either *table lines* or *text lines*.

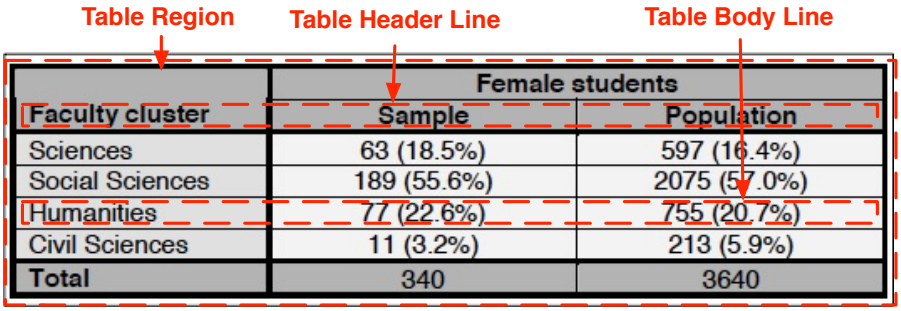


Figure 5.2: Table Location model

Tables contain two distinct types of `Table Line`:

Table Body Line: A potential `Table body Line` TBL_i is a `Line` containing two or more `Text Chunks` and located in the Table Body Region.

Table Header Line: A `Table Header Line` THL_i is a `Line` containing one or more `Text Chunk(s)` and is placed in Table BoxHead.

The Table location model is then defined in terms of `Table Lines`:

Physical Table Location (T_{Loc}^{Phy}): The model for the i^{th} table $T_{Loc\ i}^{Phy}$ is contained in a `Text Region` and spans a sequence of lines $\langle L_j, L_{j+1}, ..., L_k \rangle$, where zero or more `Table Header Lines` are followed by one or more `Table Body Lines`. $T_{Loc\ i}^{Phy}$ is defined by a `Table Region` TR_i whose bounding box is determined by $(Left(L_j), Top(L_j))$ as `Upper Boundary` UB and $(Bottom(L_k), Right(L_k))$ as `Lower Boundary` LB.

5.3.2 Segmenting Model

The aim of segmenting is to recognise and detect the inner boundaries of the table (i.e. the rows, columns and individual table cells). Each table cell can be identified by at least a (row,column) position. Table segmentation (e.g. Figure 5.3) is defined via a `Physical Table Segments` model.

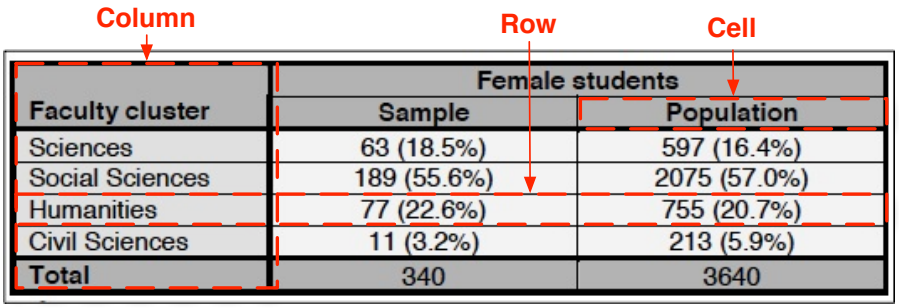


Figure 5.3: Table Segment model

Physical Table Segments (T_{Seg}^{Phy}): A `Cell` is a `Text Chunk` which is located in a T_{Loc}^{Phy}. A `Row` is a sequence of *horizontally aligned* cells, and a `Col` is a sequence of *vertically-aligned* cells. T_{Seg}^{Phy} is a triple $(Cells, Rows, Cols)$ where $Cells$ is set of two or more `Cells`, $Rows$ is a set of one or more `Rows`, $Cols$ is a set of one or more `Cols`. One cell may span multiple cells in a different row/column either horizontally or vertically.

$Cell_i$ and $Cell_j$ are `horizontally aligned` if either $((|Top(Cell_i) - Top(Cell_j)| \sim 0) \wedge (|Bottom(Cell_i) - Bottom(Cell_j)| \sim 0))$ or spanned vertically.

$Cell_i$ and $Cell_j$ are `vertically aligned` if either $((|Left(Cell_i) - Left(Cell_j)| \sim 0) \wedge (|Right(Cell_i) - Right(Cell_j)| \sim 0))$ or spanned horizontally.

$Cell_i$ `spans horizontally` $Cell_j$ if $((Left(Cell_i) - Left(Cell_j) < 0) \wedge (Right(Cell_i) - Right(Cell_j) \geq 0)) \vee ((Left(Cell_i) - Left(Cell_j) \leq 0) \wedge (Right(Cell_i) - Right(Cell_j) > 0))$.

$Cell_i$ `spans Vertically` $Cell_j$ if $((Top(Cell_i) - Top(Cell_j) < 0) \wedge (Bottom(Cell_i) - Bottom(Cell_j) \geq 0)) \vee ((Top(Cell_i) - Top(Cell_j) \leq 0) \wedge (Bottom(Cell_i) - Bottom(Cell_j) > 0))$.

5.4 Implementation

In this section, we detail how we implement the TEXUS task modules for locating and segmenting tables to realise the Locating and Segmenting models, respectively.

5.4.1 Locating Implementation

There are different approaches and techniques used for the purpose of table location. Top-down [RCVF03], bottom-up [KH06] or mixed approaches have been utilised in the literature [eS10]. They rely on different features such as line-arts [LMG08b], space distribution [MHON11], relative word positions [SS10], coordinates on the page [YKM05], document structure [LTSX06] and semantic analysis [WH02a]. Heuristic classifiers, stochastic and probabilistic models are the main methods employed for the developing this component [OR09, eS10, eSJT06].

Locating first attempts to separate *table lines* from *normal text lines* through the clustering method. It looks for lines with more than one text chunk as a potential table line. *Locating* then uses the transitions from text lines to (potential) table lines, and vice versa, to determine table boundaries. For potential table lines, our implementation scans each table line and classifies each text chunk into one of the categories: numeric, alphabetic, alphanumeric, percent, date, currency, plain text, blank, or unknown 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.

Algorithm 8 Locating (Doc Model *doc*)

```
1: for (Element t : pagecolumnNodes) do
2:     clusterLines(t)
3:     assignPatternToLineNodes(t)
4: end for
5: for (Element t : tableRegionNodes) do
6:     rebuildPatternForTr(t)
7:     String pattern = computeTRPattern(t)
8:     t.setAttribute("pattern", pattern)
9:     teaseTableRegionNode(t)
10: end for
11: for (index = 0; index ≤ tableRegionNodes.size(); index++) do
12:     Element trNode = tableRegionNodes.get(index)
13:     String pattern = trNode.getAttributeValue("pattern")
14:     generateTRspPattern(trNode)
15: end for
16: for (Element pc : pagecolumnNodes) do
17:     scanAndMergeTR(pc)
18: end for
```

We also look for spatial patterns. The algorithm scans each line in a sequence of similar type-pattern table lines to determine the extent of each text chunk (hence each potential table cell).

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 forms a *spatial pattern* for the data in the table.

In order to cover some special cases such as existence of sparse lines in table region, or not including table title in the table boundary, or considering table headers in the boundary although generally they have different patterns to data rows we have considered a loop of scanning and merging to the locating algorithm. As Algorithm 9 shows two type of merging is considered: (i) Add lines to a table region, (ii) merge table regions to one region.

Finally, we eliminate any potential table regions where the lines within the region do not exhibit sufficient similarity in their type and spatial patterns. The algorithm also deletes potential table regions which are too small (e.g. containing less than two table lines).

The Locating component outputs XML file which encloses table lines in `<table>` elements, and adds `pattern` attributes to existing `<line>` elements.

In order to locate the potential merge point from an specific line (start) in the `pagecolumn`, we search for the first `tableregion` tag occurrence as the potential merge point. Then, if the lines between the start and the potential point is less than a predefined threshold and also lines in between the start and the table region has the exact number of text chunk, the potential point is considered as a merge point.

To recognise more merging points accurately, we also check data types and spatial patterns of the two table regions. They should have similar spatial pattern considering the empty text chunks. However, if the gap between two regions is more than a threshold we do not consider them for merging. After merging, the start point is updated and the same logic will continue.

Algorithm 9 scanAndMergeTR (PageColumn *pc*)

```
1: start=0
2: int[] mergePoint= locatePotentialMergePoint(pc, start)
3: List pcLines = pc.getChildren()
4: while (start < pcLines.size()) do
5:    if (mergePoint[0] < 0 or mergePoint[1] < 0 or mergePoint[0] == mergePoint[1]
      or mergePoint[1] > pc.getChildren().size() then
6:       return
7:    else
8:       List bufferNodes = new ArrayList()
9:       for (int j = mergePoint[0]; j ≤ mergePoint[1]; j++) do
10:         bufferNodes.add(pcLines.get(j))
11:      end for
12:      if checkMerge(bufferNodes) then
13:         mergeNodes(bufferNodes)
14:      end if
15:      mergePoint = locatePotentialMergePoint(pc, start)
16:   end if
17: end while
```

5.4.2 Segmenting Implementation

In this component, the main aim is to detect the inner boundaries of the table as cells, rows and columns. Different features can be considered for segmenting a given table; such as line-art [HB07], character features [LDC05], keywords [Liu09], language model [Hur00b] and position of the words on the document page [WCM06]. Different classifiers are employed from heuristic models to data induced and probabilistic methods [WHHP02, LMG08b]. Lots of the segmenting components are developed specifically for image files and rely on image based features [KGK+07].

As can be seen in the Algorithm 10, we first search 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').

Then 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 text node boundaries against these, allowing it to both detect spanned cells and, by considering 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 an expected cell location has no corresponding text chunk and labels it as a blank cell.

Algorithm 10 Segmenting (Table Region $Treg$)

```
1: for (Element e : tableNodes) do
2:     detectPotentialHeaders(e)
3:     detectUncertainLines(e)
4:     while (existNextTextNide(e)) do
5:         textNodesColumnize(e)
6:         checkHorizontalpattern(e)
7:         spPatternRefining(e)
8:     end while
9:     spPatternFinalizing(e)
10:    rowGeneration(e)
11:    validateTable(e)
12: end for
```

Considering text chunks and the columns as rectangular objects, four main scenarios may happen in assigning text chunks to columns:

1. ($columnleft \leq textleft$ and $columnright \geq textright$) $\rightarrow$ The column boundary does not change.
2. ($columnleft \leq textleft$ and $columnright \leq textright$) $\rightarrow$ $Columnright = textright$
3. ($textleft \leq columnleft$ and $columnright \geq textright$) $\rightarrow$ $Columnleft = textleft$
4. ($textleft \leq columnleft$ and $textright \geq columnright$) $\rightarrow$ $Columnleft = textleft$ and $Columnright = textright$

Another checking to make columnizing much accurate is to give some confidence weight to the columns that have the same text alignment in each row which is either left, center or right.

After detecting the columns boundaries and updating the spatial patterns recursively, at the end the patterns get finalised for the tables and table lines. Then each table line will be considered as a table row and the text chunks will be considered as table cells. Segmenting produces an XML file which encloses detected cells in `<td>...</td>` and places the text chunks from each table row in `<tr>..</tr>`.

5.5 Evaluation

We have evaluated the performance of our locating and segmenting module implementation against two data sets.

5.5.1 ICDAR Table Competition

To evaluate our implemented system, we used the dataset introduced in 'ICDAR 2013 Table Competition[1]'. The dataset consists of 67 documents with 156 tables, and contains ground-truth for the locating and segmenting tasks. The tables are in different styles and from various domains.

The performance comparison of seven academic systems and four commercial products (FineReader, Acrobat, OmniPage, Nitro) are presented in the competition [GHOO13]. We follow the same evaluation strategy and metrics for reporting our results against the published results of the competition.

The measures *Completeness* and *Purity* are used over the whole dataset (The measure are defined in detailed in Chapter 4, Section 4.7). The measures *Recall* and *Precision* are calculated for the unsuccessful cases, counting individual characters for locating, and considering the adjacency relations for segmenting task.

Overall, TEXUS performed better than most of academic system (except the Nurminen) in all three different evaluations, and even better than Acrobat and Nitro in the commercial system in table extraction. Our performance seems acceptable in locating, however we have difficulties in segmenting especially when the header hierarchy structures are very complex. In the following, we discuss the results of locating and segmenting separately.

Table 5.1 shows the results of the locating task. The ICDAR competition reported that in general, the commercial systems performed better than academic ones. Since the details of the algorithms used by commercial systems are not publicly available, it can not be said whether their advantage originates from a better approach to the problem or from having access to a large amount of data which allow them to

[1]http://www.tamirhassan.com/dataset.html

fine-tune the heuristics in their system overtime. Our system stands nearly on top of academic systems. TEXUS has a problem in locating 'small tables' (tables with less than four rows), and tables with summary lines in the middle of the table. However, it is worth pointing out that we performed well in unruled table. As it is mentioned in the competition result [GHOO13], most of the currently available systems do not perform well with unruled tables. Locating floating tables are another category that TEXUS handles well.

Table 5.1: Result of the Locating task

System	**Per-document average**			**Table Found Total=156**	
	Recall	Precision	F-meas.	Complete	Pure
FineReader	0.9971	0.9729	0.9848	142	148
OmniPage	0.9644	0.9569	0.9606	141	130
Silva	0.9831	0.9292	0.9554	149	137
Nitro	0.9323	0.9397	0.936	124	144
Nurminen	0.9077	0.921	0.9143	114	151
Acrobat	0.8738	0.9365	0.904	110	141
TEXUS	**0.9023**	**0.8832**	**0.8926**	**114**	**138**
Yildiz	0.853	0.6399	0.7313	100	94
Stoffel	0.6991	0.7536	0.7253	79	66
Liu et al.2	0.3355	0.8836	0.4864	0	29
Hsu et al.	0.4601	0.3666	0.408	39	95
Fang et al.	0.2697	0.7496	0.3967	28	41
Liu et al.1	0.2207	0.8885	0.3536	0	25

Table 5.2 shows the results of segmenting task based the correct results of locating as input. We performed well in segmenting the vertical spanned cells and multi-line cell boundary detection. However, in some cases when tables have very irregular alignments in cells content, we ended up adding extra blank column in segmenting.

Finally, Table 5.3 shows the results of table extraction as a sequence of table locating and segmenting (i.e., segmenting component directly taking input from locating

Table 5.2: Result of the Segmenting task

System	Per-document average		
	Recall	Precision	F-meas.
Nurminen	0.9409	0.9515	0.9460
TEXUS	**0.8423**	**0.8102**	**0.8259**
Silva	0.6401	0.6144	0.6270
Hsu et al.	0.4811	0.5704	0.5220

without corrections). Our performance is comparable to other academic systems.

The task-based design of our system provides the opportunity to evaluate and refine the components separately and then try a new composition to improve the performance. It should be mentioned that participants in the ICDAR competition had the opportunity to test their systems beforehand on a practice dataset for bug fixing or training. We did not have access to that dataset and did not use the test dataset before the performance evaluation.

Table 5.3: Result of Table Extraction

System	Per-document average		
	Recall	Precision	F-meas.
FineReader	0.8835	0.871	0.8772
OmniPage	0.838	0.846	0.842
Nurminen	0.8078	0.8693	0.8374
TEXUS	**0.7823**	**0.8071**	**0.7945**
Acrobat	0.7262	0.8159	0.7685
Nitro	0.6793	0.8459	0.7535
Silva	0.7052	0.6874	0.6962
Yildiz	0.5951	0.5752	0.585

5.5.2 SIRCA Dataset

SIRCA is an independent provider of data services in Australia[2]. It provides different types of data sources for academic research purposes to its member universities. The data source we used to build the dataset is 'Australian Company Announcement'[3], which contains public announcements from Australian corporates since 1992 to present. The announcements are stored as PDF documents.

We built a corpus of 156 documents (business announcements and financial reports) consisted of 173 tables. Documents are selected to be part of the corpus based on the variety of tables contain. The documents were then ground-truthed by two domain experts. The tables are classified based on their layout and structure[4]: *simple tables, tables with multi-line cells, tables with nested headers, tables with spanned cells, fully ruled tables* and *partly ruled tables*. A table can belong to more than one category. Figure 5.4 shows some examples of the tables in the SIRCA corpus.

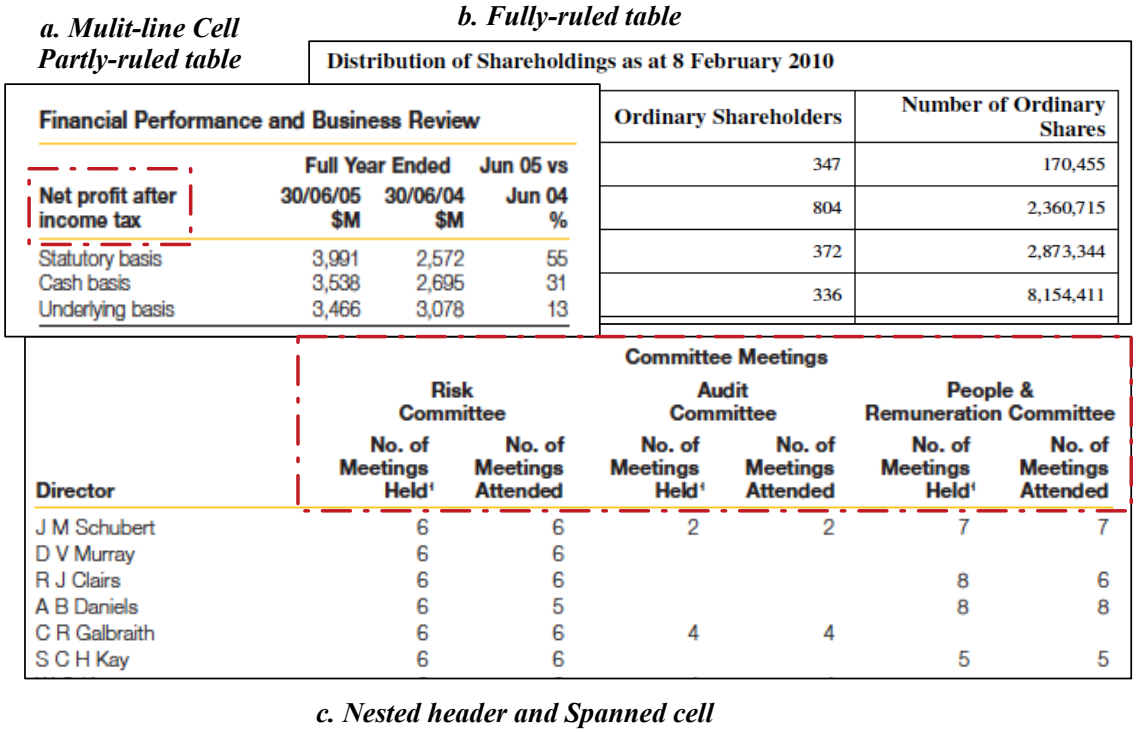
Besides reporting on the performance of TEXUS-Table Extraction subsystem itself, we also compare the results with `pdf2table` [YKM05]. There are many other systems mentioned in the literature; however, few of them are open-source and available to be run over the same dataset. Also, most systems are optimised for a specific application domain or for particular types of table, and hence not well-suited for general business documents.

The `pdf2table` system is similar to TEXUS-Table Extraction subsystem in that it is based on a heuristics model. However, to make fair comparison, we considered only single-column pages as `pdf2table` cannot handle multi-column pages. We report

[2] `http://www.sirca.org.au`

[3] SIRCA, Australian Company Announcements Database, `http://www.sirca.org.au/products/`

[4] This classification is based on table types identified in the literature.



a. Mulit-line Cell Partly-ruled table

Financial Performance and Business Review

Net profit after income tax	Full Year Ended 30/06/05 $M	30/06/04 $M	Jun 05 vs Jun 04 %
Statutory basis	3,991	2,572	55
Cash basis	3,538	2,695	31
Underlying basis	3,466	3,078	13

b. Fully-ruled table

Distribution of Shareholdings as at 8 February 2010

Ordinary Shareholders	Number of Ordinary Shares
347	170,455
804	2,360,715
372	2,873,344
336	8,154,411

c. Nested header and Spanned cell

Director	Committee Meetings: Risk Committee No. of Meetings Held[4]	Risk Committee No. of Meetings Attended	Audit Committee No. of Meetings Held[4]	Audit Committee No. of Meetings Attended	People & Remuneration Committee No. of Meetings Held[4]	People & Remuneration Committee No. of Meetings Attended
J M Schubert	6	6	2	2	7	7
D V Murray	6	6				
R J Clairs	6	6			8	6
A B Daniels	6	5			8	8
C R Galbraith	6	6	4	4		
S C H Kay	6	6			5	5

Figure 5.4: Sample tables from SIRCA corpus

separately on how TEXUS handles multi-column pages.

Performance Results of Locating.

To measure the performance of the Locating component, we use two sets of metrics: (i) *Completeness* and *Purity* (based on ICDAR 2013 table competition[GHOO13]) and (ii) *Precision*, *Recall* and *F-measure*.

According to the ICDAR 2013 table competition metrics, an extracted table is classified as '*Complete*' if it includes all sub-objects in the table region; it is classified as '*Pure*' if it does not include any sub-objects which are not also in the table region. A correctly detected table region is therefore both complete and pure.

For *Precision* and *Recall*, we used the following definitions:

$$Recall_{Loc} = \frac{Number\ of\ Correctly\ Identified\ Text\ Chunks}{Number\ of\ the\ Text\ Chunks\ in\ the\ Table} \tag{5.1}$$

$$Precision_{Loc} = \frac{Number\ of\ Correctly\ Identified\ Text\ Chunks}{Number\ of\ Identified\ Text\ Chunks} \tag{5.2}$$

We collected all four measures on each Locating output. Tables 5.4 and 5.6 presents the results of the Locating component performance on the different types of tables in the dataset. The reported values for each metric are the average of the measured values for each table type. Table 5.5 shows the results on the whole dataset (a global average).

Table 5.4: Locating Results for Different Table Types- Part1

Table Types	**# of Tables**	**Results of Locating**			
		Complete		**Pure**	
		TEXUS	pdf2table	TEXUS	pdf2table
Simple Tables	94	83	67	89	71
Tables + Multi-Line Cells	138	121	118	128	119
Tables + Nested Headers	67	49	37	53	41
Tables + Spanned Cells	127	95	91	118	105
Fully Ruled Tables	48	41	36	43	40
Partly Ruled Tables	125	101	78	114	86

Table 5.5: Table Locating Results

System	**Complete**	**Pure**	**Recall**	**Precision**	**F-measure**
TEXUS	152	157	0.9023	0.9177	0.9041
pdf2table	117	114	0.8057	0.7638	0.7840

Performance Results of Segmenting.

For the Segmenting component, we used precision and recall metrics at the cell level using the following equations.

$$Recall_{Seg} = \frac{Number\ of\ Correctly\ Extracted\ Cells}{Number\ of\ the\ Cells\ in\ the\ Table} \tag{5.3}$$

Table 5.6: Locating Results for Different Table Types- Part2

Table Types	**# of Tables**	**Results of Locating**			
		Recall		**Precision**	
		TEXUS	pdf2table	TEXUS	pdf2table
Simple Tables	94	0.912	0.864	0.947	0.726
Tables + Multi-Line Cells	138	0.876	0.852	0.819	0.798
Tables + Nested Headers	67	0.821	0.727	0.785	0.611
Tables + Spanned Cells	127	0.894	0.789	0.825	0.749
Fully Ruled Tables	48	0.924	0.832	0.927	0.812
Partly Ruled Tables	125	0.908	0.798	0.939	0.713

Table 5.7: Segmenting Results for Different Table Types

Table Type	**# of Tables**	**Results of Segmenting**			
		Recall		**Precision**	
		TEXUS	pdf2table	TEXUS	pdf2table
Simple Tables	94	0.942	0.795	0.934	0.769
Tables + Multi-Line Cells	138	0.808	0.674	0.753	0.615
Tables + Nested Headers	67	0.788	0.643	0.761	0.542
Tables + Spanned Cells	127	0.812	0.694	0.806	0.592
Fully Ruled Tables	48	0.921	0.792	0.886	0.701
Partly Ruled Tables	125	0.784	0.596	0.837	0.578

$$Precision_{Seg} = \frac{Number\ of\ Correctly\ Extracted\ Cells}{Number\ of\ Extracted\ Cells} \tag{5.4}$$

We measured both values for each Segmenting output obtained. Table 5.7 presents the results on the different types of tables in the dataset. Table 5.8 shows the results on the whole dataset.

Table 5.8: Table Segmenting Results

System	**Recall**	**Precision**	**F-measure**
TEXUS	0.8428	0.7953	0.8182
Pdf2table	0.5943	0.5281	0.5591

TEXUS consistently performed better than `pdf2table`. Within the results of TEXUS

outputs, *Simple* and *Fully Ruled* table types showed high recall and precision. As expected, tables with more sophisticated layouts (*Multi-line Cells* and *Nested Headers* table types) showed lower recall and precision. The TEXUS components already have heuristics to deal with these cases, but clearly more work is needed to improve them.

5.6 Conclusion

In this chapter, we reviewed the current challenges of table extraction systems. Most of the systems follow a black-box design and are mainly domain-specific solutions with a semi-automatic implementation.

In TEXUS, we designed a modular, task-based automatic table extraction system with standard locating and segmenting interfaces. We have evaluated our system on known corpus in table processing community and compared our result regards the academic and commercial systems. We performed well in comparison with academic systems. The modular design and implementation enabled us to compare our output at different stages of locating and segmenting.

Chapter 6

Automated Table Understanding System

This chapter explains the details of the design and implementation of the Table Understanding part in the TEXUS pipeline. Figure 6.1 shows Locating and Segmenting components in the TEXUS pipeline. The Table Understanding sub-system receives the segmented tables, produced by the Table Extraction sub-system in the previous chapter, as the input and provides Abstract Tables as output.

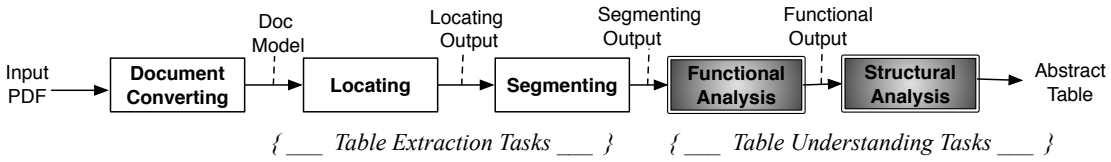


Figure 6.1: Table understanding related components in TEXUS

This chapter proposes a standard and generic definition of table understanding tasks (functional and structural analysis) and their related interfaces. We also propose the concept of domain-independent 'Stub Analysis' to improve generic table understanding. Stub patterns are classified and defined and the translation steps from

the detected patterns to Abstract Tables are described in details.

6.1 Motivation and Background

Most of the research work on table processing has been devoted to table extraction part of the problems. Table understanding remains less studied especially in unstructured tabular information. However further use of the extracted tabular data necessitate table understanding. Some works define the aim of table understanding as populating the tabular data into a database. Since the DBMS provides query and retrieval functions that allow combining information from several tables in the form of new tables [GBH+07, NSE14]. Another trend considers table understanding as the prerequisite of table interpretation and semantic analysis [PCS+07]. Because making use of table contents requires the comprehension of the logical structure of the table on the one hand as well as it semantic interpretation on the other hand.

Considering the variety of applications for table understanding in the literature, one main chain of work is to consider table understanding as *Table Logical Structure Analysis* [PCS+07, KL08, SN13] in contrast with table extraction which focuses on table physical structure analysis. Therefore the set of information used for this type of analysis is limited to table itself and not considering any external knowledge (such as ontologies). Therefore, tables can be understood more in a generic and domain-independent approach.

Following the domain-independent table understanding paradigm, two main important decision have to be made [GBH+07]:

1. What is the appropriate output (model) of table understanding?
2. How can the transmission from the table topology into the desired model be

automated?

Literature [JN08, TE09, PJK+10] suggests that the appropriate target format for table understanding is a representation based on the abstract table proposed by Wang [Wan96]. This is a multi-dimensional model of a table, separating the logical structure from the layout of a table.

On the other hand, for the task of *Automatic Table Understanding*, researchers try to exploit the same features that humans use to try to extract the relationships amongst table cells to ultimately obtain Wang Abstract Table. Following the domain-independent and generic solution [SN13, NSE14], only the extant features of the table (e.g., layout, cell content) are considered as a basis for designing table understanding systems.

Considering the diversity of table structures, it may seem challenging to formulate a one-fits-all solution for understanding tables. However, it is generally accepted that tables can be *understood* if one can detect the hierarchical structure of table headers (both row headers and column headers) properly and determine how each table data cell can be uniquely accessed through them [PCS+07, DMFE13] .

A range of different approaches have been taken to deal with this complexity. These include, for instance, only processing specific types of table such as *Well-Formed Tables* [Nag12] to differentiate between tables, lists and forms and *Multi-Dimensional Tables* [AL12]. or involving a human in the header detection process [Pad09], or removing layout features and analysing tables based on the cell arrangement of their headers [SN13]. Some works exclude nested and concatenated tables [NT12] from their analysis.

The work by Fang et al. [FMTG12] employs machine learning techniques to classify tables into different types (e.g. complex, long, folded, simple) using layout features

in the headers. However, they do not attempt to extract categories as defined in Abstract Table. Seth et al. [SJKN10] describe a taxonomy of table column headers and propose a model based on XY cutting technique to convert the geometric structure of the headers to a representation equivalent to Wang notation. However, their work is applicable only to column headers.

The most cited work in automated *Table Understanding* comes from research groups led by George Nagy and David Embley. They began with a semi-automated approach where human assistance is required during the category detection phase. This approach is used in many tools like WNT [JN08], TAT [Pad09], and VeriClick [NT12]. 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) [Nag12]. In their recent work, they propose an algorithmic solution to index table columns and rows so that detection of the critical cells can be automated [SN13].

Much of the above work has focused on discovering categories in the boxhead, with less attention paid to the stub (in particular, similar processing strategies are used for both). Our work focuses on extracting category information from the stub. Specifically, we note that table designers often use layout and styling conventions (e.g. bullet points, indentation) in the stub to encode the category hierarchy. This information has been disregarded in previous work, but we believe that it can be exploited to accurately extract category hierarchies from the stub. Doing so extends the variety of tables that can be automatically mapped to Wang's Abstract Table model.

While these approaches all contributed to advancing table understanding techniques, they did not consider additional header features that enable more accurate discovery of the hierarchical structure of table headers.

In this work, we identify patterns of layout features that commonly occur in the table stubs, and which enable us to detect the *header hierarchy* in many cases that would be missed by existing methods.

The patterns give us a concrete basis for building automatic table understanding algorithms that can be effectively applied to a wider range of tables than existing methods. Specifically, we do not assume that tables are *well-formed* (in the sense of [Nag12]), and we do not require human intervention during the process. We simply require a segmented table as input and determine the header hierarchy, and all *access paths* for each data cell from it. We evaluate the performance of our methods against the state-of-the-art on three well-known public datasets.

Therefore we make the following contributions: (i) we provide a classification of layout features in table stubs and an interpretation of the header hierarchy 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 approach leads to a straight-forward transformation of source tables to the well-known Wang Abstract Table [Wan96] model.

We first review the differences between table understanding and table interpretation. Then the TEXUS data model for functional and structural analysis is discussed and our improvement to enhance the automatic table understanding forms the rest of the chapter.

6.2 Preliminaries

6.2.1 Semantic Table Interpretation

In many applications, the logical abstraction obtained from the *Function* and *Structure* views is sufficient to provide effective access to the data. However, other applications may wish to consider domain- and application-specific knowledge. A *Semantic* level allows users to model the meaning of the data contained in the table and to find semantic relationships between table components. This abstraction level is often linked to a domain-specific knowledge base or ontology and provide a *Semantic Interpretation* of the tabular content [Zha14b].

Most of the current methods for semantic table interpretation relies on knowledge bases. A typical process for table semantic interpretation involves two main tasks [Zha14a]: (i) mapping table header cells to domain-specific concepts which is known as *NE-Column Classification* and (ii) linking the data cells to the named entities form the knowledge base which is refereed to as *Content Cell Disambiguation*.

Different features are considered for the mapping and linking sub-tasks contained in these system [TE09, DV11, VHM$^+$11, Emb04]. For example, Limaye et al. [LSC10] proposed a machine learning technique to annotate tables based on a catalog (e.g. YAGO) which comprises types, entities and relations. They defined families of features to measure the association between source tables (tables with detected data and header cells) and catalogs. Similarly, in [Mul10] a framework for interpreting tabular data is presented to enable presenting tables as linked data. Based on a hybrid knowledge base (Wikitology), they predict classes for each table column, link cell values to entities, associate properties that represent the implied relations between columns and selecting the best ones (Relation Enumeration and Selection) and Analyse the table globally decompose or normalise it into one or more sub-

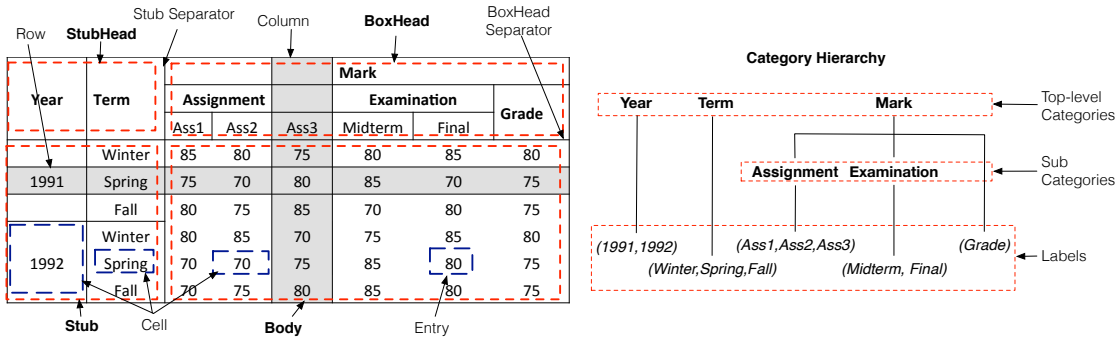


Figure 6.2: Wang Table Terminology

tables, if appropriate.

Considering that most of the semantic table interpretation systems rely on the table corpus with known logical structure, the final output of TEXUS can be fed as a perfectly structured input for these system.

6.3 TEXUS-based Table Understanding

6.3.1 Functional Model

As shown in Figure 6.2, Wang divides a table into four 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.

The body (lower-right) contains the basic data (called *entries*). The rest of the table (stub, stubhead, boxhead) contains *labels* which are used to locate the basic data. The boxhead (upper-right) consists of labels whose values are used to access individual columns. The stubhead (upper-left) and stub (lower-left) contain labels whose values are used to access individual rows.

In our model, we follow the same terminology and define two kinds of headers: *attribute cells* which are normally placed in the table Boxhead, as the top-level headers, and *access cells* placed in the stub in the role of indexes. Data cells are contained in the table body and are the target information in the table. Figure 6.3 gives examples of these.

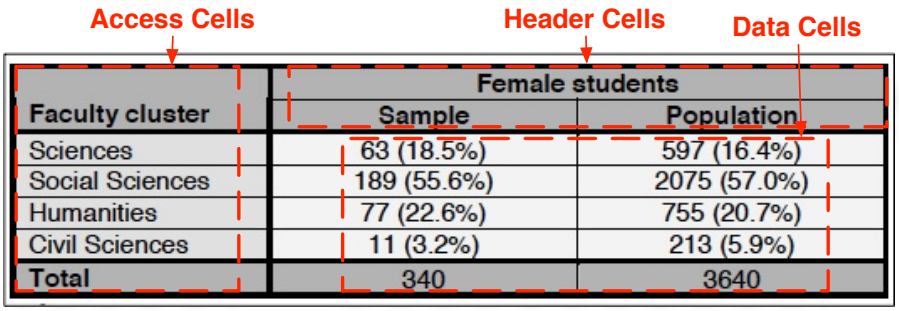


Figure 6.3: Table Function model

The functional aspects of cells in a table is described using a `Logical Table Function` model.

Logical Table Function (T^{Log}_{Func}): T^{Log}_{Func} specifies the role of each cell in a table, and identifies the four functional regions: $BoxHead$, $Stub$, $StubHead$ and $TableBody$. Roles are represented by a set of pairs $(Cell_i, Func_i)$, where $Cell_i$ is an identifier for a cell and $Func_i \in \{Data, Access, Attribute\}$. Each region is defined by the set of cells contained in it.

The following conditions hold on cells in the table:
if $Cell_i \in \{BoxHead, stub, stubHead\}$, $Func(Cell_i) \in \{Attribute, Access\}$
otherwise $Func(Cell_i) = Data$.

6.3.2 Structural Model

Structural Analysis defines the logical relationships between cells, which determines access paths to the data. A table is a two-dimensional representation of a multi-dimensional space. Labels are arranged in a hierarchy that maps the multi-dimensional space onto the table's two-dimensional data grid.

In Wang's work, a table dimension is referred to as a *category*. We can view the category hierarchy as a set of trees, with one tree for each category, and with each node labelled (see e.g. Figure 6.2). The roots of the trees in the hierarchy are called *top-level categories*, intermediate nodes are *sub-categories*, and the leaf nodes are simply *labels*. Reaching a data value in this multi-dimensional space requires following paths which intersect at the data cell. An *access path* is a sequence of labels that leads from a top-level category to a row or column of entries. An entry can be *uniquely accessed* via a combination of the access paths created from *each* of the 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.

This final output from our end-to-end processing is represented using a `Logical Table Structure` model (or Wang abstract table model). Figure 6.4 shows examples of the components in such a model.

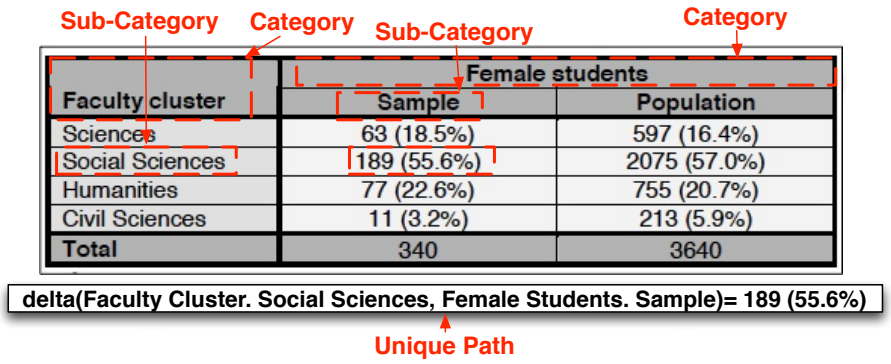


Figure 6.4: Table Structure model

Logical Table Structure (T^{Log}_{Struct}): A T^{Log}_{Struct} is a pair $(T^{Struct}_{Access}, F^{Struct}_{Map})$. A T^{Struct}_{Access} is a non-empty set of `Access Path` values, where each `AccessPath` is a hierarchical relationship between an `Attribute` or `Access` cell and their subordinate cells. An `AccessPath` $Access_i$ contains an `Attribute` cell `AttrCell` or `AccCell` cell as `Category` and an associated non-empty list of `SubCategory` values $\langle SC_1 ... SC_n \rangle$. A `SubCategory` SC_i contains an `AttrCell` or `AccCell` cell (and a possibly empty set of `SubCategory` values $\langle SC_{i,1} .. SC_{i,n} \rangle$. A `LeafPathCell` is any `AccCell` or `AttrCell` in a `SubCategory` which has an empty set of `SubCategory` values. F^{Struct}_{Map} is a function that takes a set of access paths and determines the `DataCell` corresponding to those access paths.

Logical Table Structure models satisfy the following properties: the table must have two or more categories; each category must have a single root header; a category tree cannot contain identical root-to-leaf paths; if a table has n categories, then data cells are accessed via a tuple of n access paths, one through each category tree,

Note that the Wang model does not handle tables where the top row in the $BoxHead$ contains more than one cell. We consider a *virtual header* above the top row as a starting point for the access path from the header. Similarly, if a table has an empty $StubHead$, we add a *virtual access* cell to act as the root for the access path from the stub. The names of the virtual cells could either be simple unique identifiers or could be derived based on the names in the header/stub respectively.

Note also that the final data model for our end-to-end processing consists of a set of T^{Log}_{Struct} models, one for each table, along with a set of table metadata entries (discussed in the next section).

6.4 Stub Analysis

The *stub* region consists of one or more table columns and extends from the leftmost column to the start of the body region. The labels in the stub define categories which are used to build access paths to individual table rows. The aim of stub analysis is to identify the category hierarchy implied by the arrangement of labels in the stub, and we achieve this by analysing patterns in these labels. In defining the patterns, we draw on from the work of Fang [FMTG12] and Wang [Wan96]. Wang identified layout styling rules for the stub, and Fang investigated different structural organisations commonly applied in arranging labels. We extend their work based on observations from a variety of tables in public table datasets.

We use the following kinds of features:

- Formatting Features: including font face, font size and colour
- Layout Features: including indentation, bullets, numbering and spanned cells
- Content-based Features: including repetition of labels and empty cells

In each pattern, there is an implied hierarchy among the labels in that pattern. We use the terms "enclosing label" to indicate that one label is the 'parent' of some other label and "enclosed label" to indicate that one label is subordinate to another. If A is the enclosing label of B, then B is an enclosed label of A.

6.4.1 Downward Expansion Patterns

This kind of pattern occurs when a single column is used to encode the entire category hierarchy. A column containing this pattern will be the last column in

the stub (i.e. we do not need to consider more columns to discover the category hierarchy after finding this pattern).

Downward expansion patterns can be recognised by the occurrence of the following features:

Indented Stub: Uses indentation (e.g. white spaces, tabs, bullets, or numbering) to indicate the hierarchy. Since various document rending environments have different default space threshold for indentation, we consider an initial Threshold of 10 whitespaces experimentally. If the difference between the current label left attribute and the previous label is more than the initial threshold, we assume that the current label is enclosed by the nearest non-indented label above it. Figure 6.5 shows examples of whitespaces and bullets. Taking the first table in Figure 6.5 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)

Figure 6.5: Indented Stub

Leading Label Stub: Uses special characters like ':', '=', ':-' at the end of the enclosing label ("leading label") to indicate that subsequent labels without this character are enclosed. Figure 6.6 shows an example where `Costs, Expenses and Other` and `Basic Earnings` are terminated by ':'.

Formatted Font Stub: Uses font formatting features (e.g., modifications of font

	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

Figure 6.6: Leading label Stub

appearances - bold, italic, larger size, underlines, colour) on the enclosing label. The formatted label encloses all the following plain format labels. Figure 6.7 shows an example of this case.

	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

Figure 6.7: Formatted Font Stub

Repeated Label Stub: Indicated by an identical set of labels being repeated consistently in the column. The set of repeating labels are enclosed by the nearest non-repeated label above them. In order to be more generic we consider a similarity ratio to cover the exact set of repetition or partial set of labels repeating. The threshold can be defined by user and by default it is set to 100% similar set of labels for repetition. Figure 6.8 shows an example of this case.

	2006	
	St. John's (N.L.) number	Halifax (N.S.) number
Total households	70,665	155,135
Total persons in households	178,710	368,005
Average number of persons in household	2.5	2.4
Single-detached house	38,360	80,000
Total persons in households	107,845	221,255
Average number of persons in household	2.8	2.8

Figure 6.8: Repeated Label Stub

6.4.2 Forward Expansion Patterns

This kind of pattern occurs when the category hierarchy is encoded across multiple columns. In terms of the category hierarchy, a label in the first column is a parent of one or more labels in the second column, a label in the second column is a parent of one or more labels in the third column, and so on, until the first data column is reached.

Spanned Cell Stub: Uses spanned cells in column j to indicate the association with several cells in column $j+1$. The label in column j encloses all labels in column $j+1$ whose cells are adjacent to the spanned cell. Figure 6.9 shows on sample of this type of stub.

Empty Cell Stub: Uses empty label cells in column j to indicate that the adjacent cell in column $j+1$ is enclosed by the closest non-empty label cell above the empty cell in column j. The non-empty cell and its following empty cells are treated like a spanned cell. The table presented in Figure 6.10 follow the empty cell stub pattern.

Duplicate Label Stub: Uses duplicate labels in column j to indicate that the label is associated with multiple labels in column $j+1$. A duplicated label in column j encloses all of the associated labels in column $j+1$. The set of

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

Figure 6.9: Spanned Cell 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

Figure 6.10: Empty Cell Stub

duplicated label cells could be treated like a single spanned cell. Figure 6.11 shows a table with duplicate label columns.

Cross-Product Stub: In this pattern, a set of labels in column $j+1$ are consistently repeated for each spanned cell in column j. The labels in column $j+1$ are enclosed by the corresponding spanned cell label in column j. Table presented in Figure 6.12 cross-products over two first columns in the stub.

State	Company Name	Plant I.D.	Plant Name	County
VA	MeadWestvaco Corp	50900	Covington Facility	Covington
WA	Tacoma City of	3920	Steam plant	Pierce
WI	Manitowoc Public Utilities	4125	Manitowoc	Manitowoc
WI	Mosinee Paper Corp	50614	Mosinee Paper	Marathon
WI	NewPage Corporation	10234	Biron Mill	Wood
WI	NewPage Corporation	10476	Whiting Mill	Portage

Figure 6.11: Duplicate Label Stub

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%

Figure 6.12: Cross-Product Stub

6.5 Forming a Temp Tree for Category Hierarchy from Patterns

After pattern identification, 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 forward expansion patterns since they

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

Figure 6.13: Normalised Table for Table in Fig. 6.5

are more common and an obvious way of conveying some hierarchical information. And from downward expansion patterns, the one relying on the table content are in the second level. Formatting feature based 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.

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 6.5, is shown in Figure 6.13. `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`.

Then the normalised tables interpreted to form a temporary tree of labels in the stub region, which will be used later on to determine the top-level categories and the category hierarchy.

First, we take the stub head as the root of the tree. If the stub head is empty, we create a virtual root. In a downward expansion pattern, the enclosing labels form the first level of the tree (children of root). Any corresponding enclosed labels become the children of the enclosing label. In a forward expanding pattern, starting

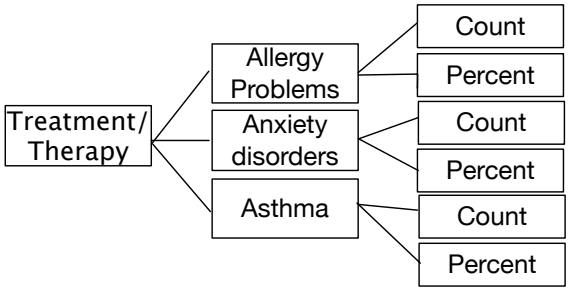


Figure 6.14: Temporary Trees from Table in Figure 6.12

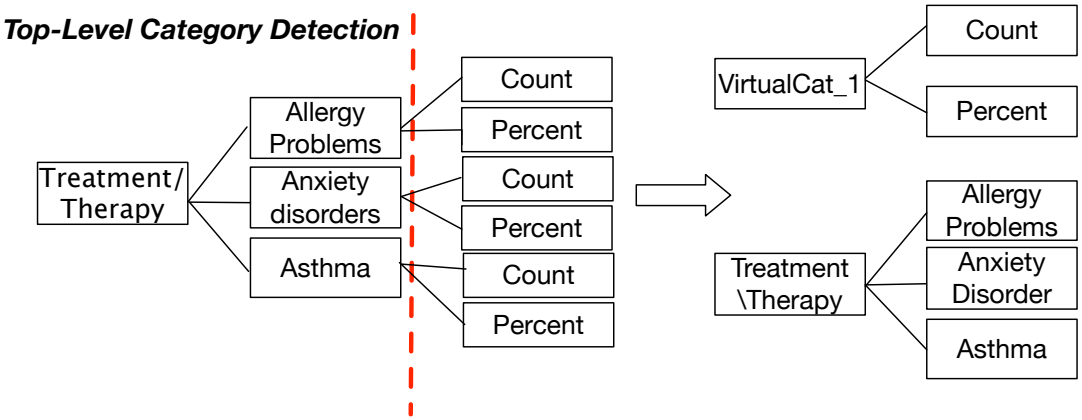


Figure 6.15: Top-Level Category Detection fro Table in Figure 6.12

from the left, the first column becomes the first level of the tree, the second column becomes the second level of the tree and so on. Figure 6.14 shows the temporary trees derived from the table in Figure 6.12. Figure 6.15 demonstrates the top-level category detection for the same table.

6.6 Top-Level Categories and Access Paths

After building the temporary trees, we analyse them to detect the top-level categories, which become the roots of the trees in the category hierarchy. The paths in the category heierarchy, from root to leaf nodes (labels) form the access paths to

the rows of data cells.

The top level of the category hierarchy is generally provided by the leftmost stub head. However, repeated sub-trees can be an indication of the existence of another top-level category. Therefore, we analyse the tree level-by-level checking for a repeated label set on one level. If found, a separate tree is created to represent the new top-level category. The repeated labels are attached as children of this category. The right side of the Figure 6.14 shows this type of category detection for Table Ex8 (which has {count, percent} consistently repeated).

If there is an appropriate label in the stub head for the new top-level category, it will be considered as the label for the top-level category, otherwise we use $\texttt{VirtualCat}_i$ (virtual category) as the label (see e.g. Figure 6.14).

We have focussed on category trees formed in the stub region in detail, but a similar process can be used to determine the category hierarchy in the boxhead region. One difference between the boxhead and stub is that the layout patterns in the boxhead are mostly forward expanding patterns. The same logic for detecting enclosing and enclosed labels and temporary tree formation is applied, except that in the boxhead, we expand the patterns from top-to-bottom rather than left-to-right (as in the stub).

The category hierarchy determined in this way forms the basis for the Abstract Table representation of tables. Each data entry in the table is associated with a set of access paths, where an access path originates from one of the top-level categories identified through the table understanding process.

6.7 Implementation

In this section, we focus on the implementation of the Table Understanding sub-system of TEXUS. We receive a PDF document as input to Table Extraction sub-system. The Document Conversion component converts it to an XML document according to our document model. Executing the processing pipeline up to the Segmenting component produces an XML file which contains 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* which consists of the Functional and Structural Analysis components.

Note that the algorithms below give a high-level view of the implementation. The procedures for finding patterns are described above, and so we simply mention them by name in the algorithms.

6.7.1 Functional Analysis

As for assigning a function/role of each cell in the table (i.e., data, attribute or access), various features can be considered such as cell content type and character features [PMWC03], similarity and coherence measures [PCS05], comparison with surrounding text [Hur00a], space distribution and number of non-empty cells per line [OR08] and ontological knowledge[PCS+07].

We consider the bottom right most cell in the table boundary as a *data cell*. Then neighbour cells are compared considering spatial and type pattern to find cells with the same functionality. At the same time, the top-most row is assumed as a *potential* header line and hierarchical structure along with spatial and type pattern help to detect the similarity of neighbour cells. The reaching point of the two algorithms is considered as the boxhead separation.

The left-most column is always considered as the *access* column. If we encounter a vertical spanned cell in the left-most column, we consider the following column for the access functionality class as well.

The large volume of unstructured tabular information is presented in high-level document formats, such as Excel, Word,PDf and HTML. The possibilities and constraints of the table presentations in these formats are similar. They allow to present some common information about a table. Therefore, regardless of the source of the input (i.e., by our own Table Extraction sub-system or an external system), we assume that the following information is accessible from the input: (i) coordinates of the cells, rows and columns, (ii) content of the individual cells in the table, (iii) spanned and merged cells, (iv) formatting styles of cells (e.g., alignment, font face, font color).

The goal of the Functional Analysis component is to assign one of three different functional roles (Data, Access, Header) to each table cell by determining the scope of stub and boxhead. We consider the first row and Left Most column of the table as Header and Access respectively. Then all possible combination of header rows and Access columns are considered to find the minimum rows and columns which should be considered in stub and boxhead. All cells in those regions are assigned the role of *Access* and *Header* respectively. Algorithm 11 shows how we determine the boundaries of these two regions. The remaining cells outside of these regions are considered *Data* cells (i.e., the entries).

To ensure the accuracy of Functional Analysis we have an extra procedure to check if the data region is recognised correctly. First, we assume the bottom right corner cell in the table boundary is a data cell. Then its neighbour cells are analysed based on the coordinates of the cell and alignment of the cell content. The content type is one of: numeric, alphabetic, alphanumeric, percent, date, currency, plain text and blank cell. A *type attribute*, which is the concatenation of all types found in the

cells of a row, is assigned to each row. The algorithm scans table lines one by one towards the top of the table. The final meeting point of this bottom-up scanning process and top-down pattern-based detection is considered the boxhead separator.

Algorithm 11 FunctionalAnalysing (Segmented Table *ST*)

```
1: stub [columns]= ST.getFirstColumn()
2: boxHead [rows]= ST.getFirstRow()
3: recursiveLevel=0
4: row=0
5: column=0
6: headerPattern [Pattern]=null
7: stubPattern [Pattern, type]=null
8: while row ≤ ST.getRowsNumber & column ≤ ST.getColumnsNumber) do
9:    hPattern= findForwardExpansionPattern(row)
10:   if hPattern!=null then
11:      row+=1
12:      boxHead.add(row)
13:      headerPattern.add(pattern)
14:   end if
15:   aPattern= findForwardExpansionPattern(column)
16:   if aPattern!=null then
17:      column+=1
18:      stub.add(column)
19:      stubPattern.add(pattern, forward)
20:   end if
21: end while
```

As described in Algorithm 11 we initially consider the first row and the first column as header and access. Then in a recursive function,function we try to find the minimum threshold of rows and columns which should be contained in the boxhead

and stub.

6.7.2 Structural Analysis

The structural analysis task in tables has been defined with various purposes and goals in the literature. Some of them are only limited to detecting hierarchies in the header cells [FMTG12], others would detect "super" headers, columns headers and row headers [PMWC03]. Different features such as external knowledge [PCS05], content types [TE09] and character feature are utilised for performing the structural analysis task. Very little research has focused on finding access paths for every data cell [JN08]. Some works consider table interpretation as part of structural analysis with different definitions and goals in the literature; e.g., from mapping table data to an ontology [VHM+11] or creating an ontology out of the table data [TEL+05] to category and relationship identification within the table [Hur00a]. Despite these varieties in the component definition, some similar features like keywords, dictionaries [LE08], context ontologies [ETL05] and content types [OR08] are considered for implementing the interpretation component.

The purpose of Structural Analysis is to detect the top-level categories and the category hierarchies. The access paths to each data entry are automatically mapped from these categories to form the Abstract Table representation.

First, we scan the stub and boxhead patterns from the output of functional analysis and then prefix the enclosing and enclosed labels to create the categories and subcategories by forming a tree. We finally look for repetition in trees and split them to from a tree for each top-level category to have unique access path from each category for every data cell. (see Algorithm 12).

our document converting module provide the information regarding that whether a

Algorithm 12 StructuralAnalysing(Functioned Table *FT*)

```
1: for (i=1, i ≤ FT.getStubSize(), i++) do
2:    stubdownPattern [Pattern] = findDownwardExpansionPatterns (i)
3: end for
4: prefixEnclosingCells(stubdownPattern,stubforwardPattern,headerPattern,FT)
5: findStubCategories(FT)
6: findHeaderCategories(FT)
7: for (i=boxHead.size(), i ≤ FT.getRowSize(), i++) do
8:    for (j=stubRegion.size, j ≤ FT.getColumnSize(), j++) do
9:       findAccessPaths(FT(i,j))
10:   end for
11: end for
```

text chunk is bulleted or not. Details of different type of bullets and their detection is described in Chapter 4. To detect the leading label existence in the stub we check if the text string of the related cell ends with a pre-defined list of characters. Algorithm 13 shows the heuristic function to detect the indentation.

Algorithm 13 Indentation Detection (Functioned Table *FT*)

```
1: if (previousCell != null & previousCell.getAttributeValue("left") != null &
   firstColumn.getAttributeValue("left") != null) then
2:    previousLeft = previousCell.getAttributeValue("left")
3:    currentLeft = firstColumn.getAttributeValue("left")
4:    if Math.abs(currentLeft - previousLeft) ≥ indentationThreshold then
5:       pattern.type = INDENTED
6:    end if
7: end if
```

For every pattern, a set of enclosing and enclosed cells will be detected and each enclosed cell will be prefixed by an enclosing one. After that there will be one path

from the boxhead and one from the stub for every data cell. At this point, each data cell is indexed by an access cell in the same row and a header cell in the same column. In order to detect the exact number of top-level categories, we analyse the paths, searching for repeated sub-paths which indicate the existence of another top-level category, after which the paths will be updated to include an access path for each top-level category.

6.8 Evaluation

We evaluate our system in two ways: first, we benchmark our results against the latest work by [SN13] and [NSE14] using the same dataset. Second, we analyse the overall performance of our system against two other well-known datasets.

6.8.1 Benchmarking with the DocLab Dataset

In the first part of the evaluation, we compare our results to the latest work in table understanding by Seth [SN13] and Nagy [NSE14] whose evaluations were based on the DocLab dataset[1]. We used the same dataset as used in [SN13, NSE14][2] which contained 200 HTML tables, pre-processed and converted to CSV formats, and ground truths.

Tables 6.1 and 6.2 show the characteristics of the tables in the dataset in terms of the number of top-level categories, the number of columns in the stub regions and the stub patterns. We note that the pre-processing of the HTML to CSV has removed much of the layout and formatting features. We manually traced some of

[1]http://www.iapr-tc11.org/mediawiki/index.php/The_DocLab_Dataset_for_Evaluating_Table_Interpretation_Methods

[2]We are grateful to the authors of work for sharing the dataset.

the features such as indentation and bullets, but font formatting was not traceable.

Table 6.1: The DocLab Dataset: Table Types

By Table Type	**# of Tables**
2-Category	177
Muti-Category	21
One-column Stub	194
Multi-Column Stub	4

Table 6.2: The DocLab Dataset: Stub Patterns

By Stub Patterns	**# of Tables**
Downward Expansion	**61**
Layout and Formatting	43
Content-based	18
Forward Expansion	**4**
Layout and Formatting	0
Content-based	4

In [SN13], the ground truth of the 200 tables was provided in the form of *4 critical cells* which determines the boundary of the stub and entries (i.e., body region that contains the data). CC1 and CC2 represents the the top-left and bottom-right cells of the stub head and CC3 and CC4 are indicators of the top-left and bottom-right cells of the data-cell region. To be able to compare our results with these, we transformed the XML output of our functional analysis component to highlight the locations of the cells that correspond to these 4 cells.

To compare the structural analysis part, we count the number of top-level categories detected by our system both from the stub and boxhead regions, and compare the numbers with the comparable work presented in [NSE14]. Since we provide a more detailed hierarchy for each category, we can only compare the top-level categories across the different methods.

Results

As shown in Table 6.3, Seth et al. [SN13] reported 100% accuracy in their system for functional analysis (excluding two erroneous tables). We also achieve the same accuracy, in that our system correctly segments and functionally analyses all tables (except for the same two erroneous tables).

Seth et al. [SN13] also reports 3 seconds processing time for detecting critical cells for all 200 tables. In our case, the pipeline processing takes one table at a time. So, the whole process on average, took 4 seconds per table.

In terms of the structural analysis, Nagy et al. [NSE14] reported that their system correctly detected all 21 cases in the dataset that contain tables with more than two top-level categories (multi-categories). We also detected all 21 cases correctly and the number of top-level categories in each table was also 100% correct.

Table 6.3: The benchmark results on DocLab Dataset

System	**Functional Analysis**	**Structural Analysis**
Seth and Nagy [3]	1 *	1**
TEXUS	1	1
*: 2 tables are non-indexable, therefore it is segmenting 198 table **: They just detect top-level categories		

However, by recognising patterns on the stub region, we have detected more top-level categories than other approaches. In particular, we detect the hierarchy completely in one-column stubs. Figure 6.16 shows an example of this. From the layout structure, we can understand that there are relationships between "Renewable Total" and all its enclosed labels, and also between "Waste" and its enclosed labels. By correctly detecting this through our analysis, further understanding of the relationship (e.g. 'kind of' , 'is member of') through semantic interpretation becomes easier.

7	Source	2003	2004	2005	2006	2007
8	Total	948,446	962,942	978,020	986,215	994,888
9	Renewable Total	96,847	96,357	98,746	101,934	107,954
10	Biomass	9,628	9,711	9,802	10,100	10,839
11	Waste	3,758	3,529	3,609	3,727	4,134
12	Landfill Gas	863	859	887	978	1,319
13	MSW1	2,442	2,196	2,167	2,188	2,218
14	Other Biomass2	453	474	554	561	598
15	Wood and Derived Fuels3	5,871	6,182	6,193	6,372	6,704
16	Geothermal	2,133	2,152	2,285	2,274	2,214
17	Hydroelectric Conventional	78,694	77,641	77,541	77,821	77,885
18	Solar/PV	397	398	411	411	502
19	Wind	5,995	6,456	8,706	11,329	16,515
20	Nonrenewable Total	851,599	866,585	879,274	884,281	886,934

Figure 6.16: Example of more detailed sub-category

6.8.2 Performance on ICDAR and PDF-Trex Datasets

With the first part of the evaluation, we have demonstrated that our system can perform just as effectively as the best known work in table understanding, with the advantage of producing more detailed category hierarchies from the stub.

In the second part of the evaluation, we investigate the performance of our system on a wider range of table types. We have compared our performance against two public datasets well-known to the table processing research community: *the ICDAR competition dataset* [4] containing 67 PDF documents with 156 tables and *PDF-TREX* [5] containing 100 PDF documents with 164 tables, in Italian and English.

As these datasets do not have ground truth for table understanding[6], we have manually created ground-truthed datasets based on two human judges. The human judges were asked to nominate the top-level categories and complete categories with their hierarchical structure for each table.

[4]http://www.tamirhassan.com/dataset/

[5]http://staff.icar.cnr.it/ruffolo/files/PDF-TREX-Dataset.zip

[6]The ICDAR dataset only has ground truth for table extraction (locating and segmenting).

Tables 6.4 and 6.5 show the characteristics of the tables in the dataset in terms of the number of categories, the number of columns in the stub regions and the stub patterns. As shown, most of the tables have only one column in the stub which strengthens our argument that a detailed analysis on the stub region, such as layout formatting in a single column, should provide more accurate table understanding. The summary also highlights that the layout and formatting in download expansion patterns is the most common pattern in the datasets, in which the indented stub (using whitespace, bullets and numbering) is the most dominant.

Table 6.4: The PDF-Trex and ICDAR Dataset: Table Types

By Table Type	**PDF-Trex**	**ICDAR**	**Total**
2-Category	136	124	260
Muti-Category	28	32	60
One-col Stub	143	139	282
Multi-Col Stub	21	17	38

Table 6.5: The PDF-Trex and ICDAR Datasets: Stub Patterns

By Stub Patterns	**PDF-Trex**	**ICDAR**	**Total**
Downwardd Expansion	**104**	**44**	**148**
Layout/Format	59	25	84
Content	45	19	64
Forward Expansion	**21**	**17**	**38**
Layout/Format	12	13	25
Content	9	4	13

For the analysis, we used our table extraction sub-system in TEXUS, to produce the correct segmenting output of the tables to be fed into our table understanding sub-system.

Results

We have correctly performed functional analysis on 96% of the tables in the dataset (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,
- one long table extending across two pages,
- one table with vertical text direction in the header rows (we only process horizontal text direction)

The result of structural analysis is show in two parts: First, the top part of Table 6.6 shows the performance of detecting stub patterns by type using the precision and recall measures. Second, on the bottom part of the table shows the performance on detecting the top-level categories and category hierarchy on 309 valid tables using the $\texttt{Total}_{\texttt{sim}}$ measure defined below.

In order to measure the effectiveness of our approach to detecting the top-level categories and the category hierarchies, we created an XML representation of the *category hierarchies* for each table in the ground-truth datasets and compared this against the output of our structural analysis module. There were 309 valid tables considered: 249 two-category, and 60 multi-category.

We measured the similarity of hierarchy trees (in XML files) based on the tree edit distance [RMS06]. We defined three edit operations, *Insert*, *Delete* and *Rename* at node level with the same cost function of 1. Then the overall similarity for the two trees were calculated as follow:

$$\texttt{Total}_{\texttt{Sim}}(\texttt{T}_\texttt{O}, \texttt{T}_\texttt{G}) = \frac{\texttt{match}(\texttt{T}_\texttt{O}, \texttt{T}_\texttt{G})}{\texttt{diff}(\texttt{T}_\texttt{O}, \texttt{T}_\texttt{G}) + \texttt{match}(\texttt{T}_\texttt{O}, \texttt{T}_\texttt{G})} \tag{6.1}$$

Table 6.6: Results of Table Understanding Performance on ICDAR and PDF-Trex

Pattern		Recall	Precision	F-Measure
Downward Expansion	Layout/Format	.97	.91	.94
	Content	.94	.93	.94
Forward Expansion	Layout/Format	1	1	1
	Content	1	1	1

	Top-Lvl Category	Category Hierarchy
2-Category	**1**	**.87**
Multi-Category	**1**	**.92**

where $\mathtt{T_G}$ is the ground-truth, $\mathtt{T_O}$ is the output of the system, $\mathtt{diff}(\mathtt{T_O}, \mathtt{T_G})$ is the number of nodes to be edited to map the $\mathtt{T_O}$ to $\mathtt{T_G}$, and $\mathtt{match}(\mathtt{T_O}, \mathtt{T_G})$ is the number of nodes that remain unchanged.

We report the average performance over all valid tables in the dataset. As can be seen in Table 6.6, we detected all top-level categories correctly in both table types. The differences in category hierarchies were mainly due to false detection of layout features, in particular changes in font colours and sizes.

6.9 Conclusion

In this chapter, we presented a novel approach to automated table understanding, based on an analysis of commonly-occurring patterns in the stub region of tables. An important aspect of our approach is that the use of patterns means that we do not rely on the use of external knowledge. Our approach is different to prior work on discovering category hierarchies, which focussed on the boxhead region, by also considering layout patterns in the *stub* region.

We incorporated the stub layout patterns defined above into the functional and

structural analysis components of our TEXUS system, and analysed its performance over widely-used public datasets.

Our system performs at least as well as the latest work in the area, and surpasses it by extracting more accurate category hierarchies for certain kinds of tables. Accurate category hierarchies are important in allowing the output of the system to be more effectively use for further table analysis such as semantic interpretation applications and table similarity detection.

Chapter 7

Applications of TEXUS

In this chapter, we present two more topics that are worth mentioning for further discussion. Firstly, to demonstrate applicability of the final output produced by TEXUS to practical scenarios, we show how the final XML content, which is in the form of Abstract Table, is straightforwardly translated to a common data format such as CSV (Comma Separate Values). Secondly, we discuss some of the existing issues in evaluating the performance of table processing systems, considering the view points of both the literature and our own experiences. For some, the TEXUS framework may provide a basis for creating common grounds, but many issues remain open and future research topics in the area.

7.1 Annotated Tabular Data

Although Abstract Tables are considered to be the "ultimate" table model by the table processing community, different representations of the model could make it more practical when the content needs to be used in some other applications.

One of the trends in this regard is to consider suitable Web-based representations for tabular data so that it can be easily distributed, processed (and understood) over the Web. Just recently, a W3C working group, *CSV on the Web Working Group*[1] has released a set of CSV-based schema, to which tables and their related metadata can be mapped. The *Model for Tabular Data and Metadata on the Web* is now a W3C recommendation [TK15]. In the rest of the section, we would like to show that our XML content can work well with this model.

CSV files can be generated and manipulated easily, and there is a significant body of software available to handle them. They make a simple and easy-to-adopt standard for tabular data presentation. In particular, it is a format readable by both machines and humans.

From a tabular data representation point of view, CSV is a two dimensional structure consisting of rows of data, each row containing multiple cells. Rows are usually separated by line terminators so each row corresponds to one line. Cells within a row are separated by commas. However, it should be noted that there are some drawbacks such as the lack of ways to specify type information or support for describing of relationships between tables.

The CSV model by the working group contains access methods for CSV metadata, metadata vocabulary for CSV data and transformation mechanism between CSV and other representations such as JSON, RDF and XML. The metadata can be located externally or embedded within the tabular data file itself. The metadata can be explained in different type and depth from atomic properties (e.g. boolean, string, integer) to natural language and semantic properties.

The metadata in the model provides information which can be utilised differently by many types of applications:

[1] `https://www.w3.org/2013/csvw/wiki/Main_Page`

- Viewers can use the indicated metadata to provide a more user-friendly or human-readable view of the CSV file, which might include displaying it in a table, or as graphs, or charts.
- Data entry tools can use the metadata to prompt people to supply information that is added to a CSV file.
- Validators can check that the labels of the columns in the metadata file match those in the CSV file, that the values in the columns are of the right type and in the right format, and that values in the GID column are all present and unique.
- Converters can use the metadata to map the CSV data into other formats such as JSON, RDF, and XML, or into databases or statistical applications, in intelligent ways.
- Data Aggregators can use the indicated metadata, such as descriptions, titles, modification dates, and licences, to enable more intelligent retrieval of relevant data on the web.

In the following, we explain an extension to our structural analysis component to provide the abstract table in CSV format along with the initial XML output. This extension enables the addition of the semantic metadata which can be utilised further by the CSV model.

7.1.1 Mapping XML Abstract Table to CSV Metadata

The CSV model [TK15] focuses on a simple header structure for tables. To fit our final output, which represents a multi-dimensional structure, into the model, we add our own interpretation of “multi-dimensional” headers as follows.

First, we divide our final table data into two relational entities[2]: the categories hierarchy (i.e., dimensions) and the access path for each data cell. We then represent each entity in a relational table, assigning a primary key value for each row in each table. These tables are then related by the keys, effectively joining/combing them.

Therefore, we present every Abstract Table in two CSV tables: (i) One table named *dimension* to represent the categories information and (ii) *data* table to show the table data based on the access paths. The metadata information is embedded in the *dimension* table according to the W3C recommendation [TK15]. Every top-category, sub-category and label in the tables along with the available metadata will be represented as a row in the *dimension* table. Table 7.1 shows the schema for *dimension* table. Then two tables are related to each other through the foreign keys.

Table 7.1: Dimension Table Schema

CREATE TABLE dimension{		
id	Integer	Primary Key
attribute	VARCHAR	not Null
parent	Integer	Ref dimension (id)
type	String	
format	String	
description	String	

Top-level categories are the main attributes of the table with no parent. The leaves are parents of no other attribute. For every table, there is one entry in the attribute table per each label in the header hierarchy. Figure 7.2 shows the category hierarchy for the Wang table (presented in Figure 6.2). Every node in the access path receives a unique ID, a name and also the parent of the node in the path and the data type presented in the node is specified. Figure 7.1 shows a visual of the IDs assigned to the nodes in the access path for each category from the dimension table.

[2]entities in the sense of relational database

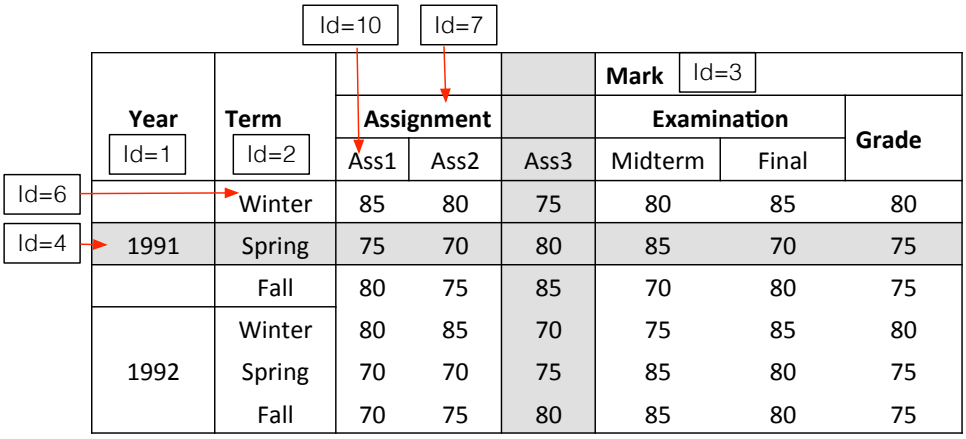


Figure 7.1: Sample of ID Assigned to Nodes in Categories for the Wang Table

Every Top-level category forms a header in the *data* table and the corresponding *id* from *dimension* table is considered to form the access path for each data cell. Figure 7.3 shows the data table for the first row of the Wang Table (presented in Figure 6.2). As it is mentioned in the schema, all columns except the *Data* in the table are a reference ID to an attribute in the related dimension table.

Generally, the necessary metadata is extracted directly from the tables, relying on the information such as structural analysis output and the data types of cell content. The algorithm for creating a dimension table from an Abstract Table is explained in Algorithm 14. As can be seen (Lines 1 to 9) we parse the abstract table and detect all top-categories. Then for each path, starting from the top-categories, we add the sub-categories to reach to the data nodes (Lines 13 to 27). Every node in the path will be added to the dimension table with a unique ID.

After creating a dimension table, a data table will be generated by referring to the category information in the dimension table. The steps are shown in Algorithm 15. For every data cell, we parse the access path and relate the subcategories and categories on the path to the corresponding record (Line 12-16) in the dimension table.

Algorithm 14 Generating Dimension Table (Structured Table *ST*)

```
1: paths = ST.getChild(Category).getChildren()
2: categories = new ArrayList()
3: for (category : paths) do
4:    if (category.getName().equals(Access or Header)) then
5:       for (c : (List)category.getChildren()) do
6:          categories.add(c)
7:       end for
8:    end if
9: end for
10: resultTable = new Element(table, header)
11: int seq = 0
12: subs = new ArrayList()
13: for (c : categories) do
14:    if (c.getName().startsWith(SubCategory)) then
15:       subs.add(c)
16:    end if
17: end for
18: while (subs.size() >0) do
19:    category = subs.remove(0)
20:    for (c : (List)category.getChildren()) do
21:       if (c.getName().startsWith(SubCategory)) then
22:          subs.add(c)
23:       end if
24:    end for
25:    seq++
26:    h = new Element(tr, category, seq)
27:    resultTable.addContent(h)
28: end while
29: return resultTable
```

Algorithm 15 Generating Data Table (Structured Table *ST*), Dimension Table *DT*

```
1: paths = ST.getChild(Category).getChildren()
2: header= new Element (tr)
3: header.addcontent(findTopLevelCategories(paths))
4: resultTable = new Element(table, header)
5: for (i = numHeaderLines; i <lines.size(); i++) do
6:    line = lines.get(i)
7:    for (j = numAccessColumns; j <columnSize; j++) do
8:       tr = new Element(tr);
9:       cell = getColumnofALine(line, j);
10:      accessPath = cell.getAttributeValue(AccessPath)
11:      for  (category : ST.getallaccessnodes()) do
12:         for (ap : accessPaths) do
13:            if (ap.startsWith(category.getAttributeValue(CName)) then
14:               td = new Element(td)
15:               td.setText(FindAccessOrHeaderID(DT, ap))
16:               tr.addContent(td)
17:            end if
18:         end for
19:      end for
20:      tddata = new Element(td)
21:      tddata.setText(cell.getText())
22:      tr.addContent(tddata)
23:      resultTable.addContent(tr)
24:   end for
25: end for
26: return resultTable
```

```
# This schema describes the hierarchical structures of categories in a table.,,,,
# URI, http://example.org/Wang-dimension.csv,,,
# name,ID,Attribute,Parent,Type
# description,Unique ID for each attribute in the hierarchy,The string content of the attribute,A reference to the ID of the attribute parent in the hierarchy,Type of the attribute
,id, attribute, parent, type
,1,Year,,String
,2,Term,,String
,3,Mark,,String
,4,1991,1,Date
,5,1992,1,Date
,6,Winter,2,String
,7,Spring,2,String
,8,Fall,2,String
,9,Assignment,3,String
,10,Ass1,9,Integer
,11,Ass2,9,Integer
,12,Ass3,9,Integer
,13,Examination,3,String
,14,Midterm,13,Integer
,15,Final,13,Integer
,16,Grade,3,Integer
```

Figure 7.2: CSV Dimension Table for the Wang Table Example

7.2 Table Processing Systems Evaluation

In this section, broadly explore the issues of evaluating the performance of table processing systems. We believe the TEXUS framework could provide a basis for creating some common grounds in some of the issues (e.g., table processing terminology), but many issues noted here remain open and future research topics in the area.

In all academic literature on the topic of evaluation, the goal of exercise is to measure the differences between the experimental results produced by a system on some test data against the ground-truth of the test data. The measurements are taken in

```
# This schema describes the table data and their related access paths,,,,
# All columns except the Data in the table are reference ID to an attribute in the
related dimension table,,,,
# URI http://example.org/Wang-data.csv,,,,
# foreignKeys:reference: resource,"http://example.org/Wang-dimension.csv",,,
# name, Year ,Term,Mark,Data
#     foreignKeys:reference:          resource,http://example.org/Wang-dimension.csv,
http://example.org/Wang-dimension.csv,http://example.org/Wang-dimension.csv,,
# foreignKeys:reference:columnReference,ID,ID,ID,,
,Year,Term,Mark,Data
,4,6,10,85
,4,6,11,80
,4,6,12,75
,4,6,14,80
,4,6,15,85
,4,6,16,80
```

Figure 7.3: CSV Data Table for the First Row of the Wang Table

terms of the number of errors and the types of errors.

In determining the measurements, there should be a standard evaluation methodology that defines 'what' will be compared as a matter of dataset and ground-truth data, and 'how' to compare them in the form of evaluation steps and metrics. However, the whole process of table processing performance evaluation is not straightforward, and there are some outstanding issues for the said questions "what" and "how".

We list some of the issues mentioned regarding the table processing evaluation which are noted in the literature.

7.2.1 Lack of Standard Dataset

In order to evaluate the performance of a table processing system, there should be a standard collection of documents selected by considering the following measures:

1. Size: Most of the existing table processing systems are evaluated on rather small-size datasets. For instance, the dataset used by [HKLW02] was composed of 26 Wall Street Journal articles in text format and 25 email messages. Silva [eSJT06] gathered 22 PDF financial statements as the testing dataset.

   The UW-III dataset [Phi96] is most widely used for evaluating image documents understanding and segmentation tasks. It is also used for table detection because there are 215 marked table zones distributed over 110 pages. However, it does not provide detailed structure information (only a bounding box of table region), is still small in size, and only applicable to scanned images.

   Wang [WPH04] gathered a total of 1125 document images, and generated ground-truths with 565 table entities and 10298 cell entities. However, the dataset and ground-truth are not publicly available. Fang et al. [FTT+12] have created a collection of 2000 PDF pages in which there are 1000 pages containing at least one table.

2. Public-Accessibility: In order to standardize table processing systems evaluation, it is necessary to make general datasets publicly available. Most of the current research work test their systems on in-house datasets, which make it difficult for other systems to compare their performance with them.

3. Diversity: A high-quality dataset should maintain good variety in both document formats and layouts as well as table styles. Some systems do not rely on specific language features for tables detection and processing, so there should be variety in the languages of the collected documents. The proposed dataset in [FMTG12] is composed of Chinese and English pages at the proportion of about 1:1. The dataset collected for an assessment of table understanding systems [GHOO12] contains a variety of single-column and multi-column/complex layout documents. The lack of a dataset composed of different formats of documents (e.g., HTML, digital PDFs, images) is another issue, which should be

considered [GHOO13].

4. Domain: In order to evaluate a domain-independent table processing system, the dataset should contains documents collected from different domains reflecting various table styles and content. [GHOO12] builds an objective dataset of freely distributable PDF documents from several domains by performing a Google search in government Web sites which tend to be in the public domain and inspected each returned document in sequence.

5. Different Table Types: A comprehensive dataset should contain different table types mentioned as "anomalous cases" in the literature. [HKL+01] has analysed the 135 tables in the UW1 dataset and found that 83 of them were abnormal for one reason or another. The irregularities come from both table layout structure and contents.

6. Covering Errors: In gathering documents for evaluation purpose, the common errors in different steps of table processing should be taken into consideration. Fang [FTT+12] has build their test dataset with both 'positive' and 'negative' cases at the proportion of around 1:1. From all 2000 pages within the dataset, there are 1000 pages containing at least one table, while the other 1000 pages do not contain tables, but have complex layout where some page components may be mistakenly recognized as tables, such as matrices and figures. [HKLW02] had common page segmenting error cases in mind when collecting samples for evaluating their table processing algorithm for PDF documents.

7. Different Applications: Performance evaluation is needed in order to compare and select the best-suited methods for a given application and this kind of application- oriented approach should be considered in test dataset collecting, too. [FTT+12] proposed a dataset having different penalty scores and coefficients for every error regarding the table processing application. They took mobile reading application as an example in their evaluation prototype

and divided errors based on 5 application oriented categorization rules and considered every one of them in collecting sample dataset.

Considering all these measures, there are some currently available datasets that can be used for table processing (specifically table locating and segmenting):

- PDF-TREX Dataset: It is a publicly-available dataset composes of 100 PDF documents (written in different languages) coming from different domains and containing 164 tables. [OR09].
- ICDAR 2013 competition Dataset: It is generated by collecting PDFs from a Google search with these two limitation `site:europa.eu` and `site:*.gov` to cover different domains and documents. From the collected documents, they selected tables that meet some criteria (unambiguous bounding box and cell structure; no super- or subscript) and extracted excerpts containing approximately two pages before and after the table, in order to give the algorithms ample opportunity to find false positives. They provided two datasets, one containing 59 excerpts with a total of 117 tables and the other one includes 77 excerpts with a total of 156 tables [GHOO13].
- Marmot Dataset: In total, 2000 pages in PDF format is collected and the corresponding ground-truths are extracted utilizing a semi-automatic ground-truthing tool "Marmot". The dataset is composed of Chinese and English pages. The Chinese pages are selected from over 120 e-Books with diverse subject areas provided by Founder Apabi library, and no more than 15 pages were selected from each book. The English pages were crawled from Citeseer website. The pages show a great variety in language type, page layout, and table styles. Among them, over 1500 conference and journal papers were crawled, covering various fields, spanning from the year 1970, to latest 2011 publica-

tions. The e-Book pages are mostly in one-column layout, while the English pages are mixed with both one-column and two-column layouts [FTT+12].

- T-Truth: UNLV and UW-3 datasets are encoded as color-code representation in image format by using different color channels for tables, rows and columns to enable uniquely identification of each cell in a given table. The input documents should be in image format too [SSKD10].

7.2.2 Lack of Ground-truth

Provided that we have a dataset to evaluate the performance of a table processing system, how the collected data should be ground-truthed is another challenge. Here is some fundamental consideration for ground-truthing discussed in the literature:

- Necessity of formal specification: First of all the compatible format of the ground truth makes evaluation independent of document medium. Also, standardising the ground-truth output make it easy for researchers to understand and use it for evaluating algorithms or experiments with new metrics. Most of the works in this era utilised XML format for the formalisation purpose. [JN08]

- Essence of evaluation metrics: In order to validate the system results regarding the ground-truthed data structures, there should be a metric to show the discrepancy between different specifications of the same object. Hu et.al. [HKLW02] proposed a system teval stands for table evaluation which calculates the distance between tables resulted from recognition system and table presented in the ground-trtuh as a combination of cost functions of *insertion, deletion, substitution(split), substitution(match), substitution(merge).*

- Ground-truth metadata: In both cases of automated ground-truthing and human-based one, there should be some information describing the ground-truth structure and its within elements or surrounding environment. Such information can also be formally codified. It represents both the skill-level of the truther and the knowledge-base of the recognizer. Fang [FTT+12] used an XML schema to describe the metadata, while [JN08] utilised descriptive ontologies. Gobel et.al. [GHOO12] mentioned presenting just the tabular region in ground-truthed documents as a shortcoming of some of current evaluation approaches.

There are some problems that may arise during ground-truthing tables and their related metadata:

- Non-expert ground-truther: It is needed that the ground-truther be either a part of table processing system development team or an expert of technical domain the documents and tables are extracted from. Lack of subject-matter specialist will reduce the quality and reliability of the ground-truthed documents [HKL+01].
- Lack of comprehensive table definition: Considering the ambiguity of table structure definition, it is difficult to differentiate between tables and similar structures. Although, there should be a unified formal model for tables before the evaluation, some measures during the ground-truth can improve the evaluation performance. For instance, there should be more than one ground-truther and some rules should be set to rely on in the case of disagreement. in [FTT+12] 15 people participated in ground-truthing task and to minimize subjectivity, unified standard for labelling were set, and each ground-truth file is double-checked.
- Inadequate ground-truthing tools: Any interactive system for creating ground-

truth introduces some obstacles, even when the interpretation of the table is clear [WPH01]. Shahab [SSKD10] proposed a interactive system called "T-Truth" for image documents ground-truthing.

- Complex table structures: Most of the automated table processing systems have problem in detecting and processing complex table structures. Most of the systems eliminate such tables from consideration beforehand.

- Difficulties related to table context: These are caused by errors in syntax or semantics, or fault in the table generation phase or printing/scanning process.

7.2.3 Lack of Performance Metrics

Over the years, many table processing systems were developed , and their performance usually published in terms of recalls and precision. Then the false positive and false negative instances are judged manually. Existence of partially recognition cases, makes it hard to decide whether an extracted table is totally correct or wrong.

The precision and recall metrics originally developed for information retrieval and later adapted for classification purposes. *Precision* is the proportion of elements correctly identified regarding the number of identified elements and *Recall* again considers the correctly detected elements but regarding to the number of target elements. The main point is that both metrics have this assumption that the granularity of items at output is the same as at input. However, this assumption is not valid for most of table processing tasks and authors must choose between measuring them in relation to input or output units if they want to use these metrics.

Fortunately, there are already some research work highlighting the insufficiency of precision and recall for table processing. Here are some reasons mentioned in such literature regarding the inadequacy of these metrics:

- Table processing usually involves merging and splitting elements to/from other elements, rather than classifying known elements into different types. Therefore, precision and recall as metrics dedicated for classification type of tasks are not entirely appropriate [eS10].

- Precision and recall measures are usually useful in cases evaluating performance based on human evaluation [FTT+12].

- High recalls and precisions for specific table processing tasks do not necessary imply good overall performance, as the results might come at the expense of making mistakes in other parts of the whole process [LCD06].

- The biggest problem that arises with the approach of using precision and recall for evaluating table related tasks is the ambiguous interpretation of the term 'correctly retrieved' in this context [Has10a].

- Recall and precision alone do not provide feedback on the types of errors and the sizes of errors in table processing task. For example, missing one line of an expected answer and almost all lines of that would result in the same number of errors [Lon09].

Considering the said drawbacks of precision and recall for table processing evaluation, some work proposed changes both in definition and application of these two metrics to make them suitable for the context:

- In order to cover the shortcoming of precision and recall in handling different level of abstraction in the input and output of a table processing system, Silve [eS11] suggested that what is needed is a measure, able to generate the granularity required at output from the input granularity. According to that, in any aggregation or division tasks, two main types of errors can occur: either different elements are merged together so the detected element is impure, or

one element gets split in two (or more) so the detected elements are incomplete. In fact, in order to be completely identified, a column must contain all of its cells; and to be pure, it must contain only its own cells. Based on these concepts, they propose the following absolute metrics:

- Completeness – proportion of completely identified elements with respect to the total number of real elements; for example, in order to be completely identified, a column must contain all of its cells.
- Purity – proportion of purely detected elements with respect to the total number of detected elements; a pure element is one whose components belong to only one original element.

Measuring completeness and purity typically involves choosing two levels of macro and micro granularity which the metrics will compare and when these two levels converge, completeness and purity converge towards recall and precision.

- Hurst [Hur00a] proposed a new metric. In the ground-truthing phase, each cell is linked to its vertical and horizontal adjacent cells and these cell-to-cell links are the basic entities considered for recall and precision calculation. This is called Proto-link evaluation.

- Tomas extended the measures recall and precision in order to deal with recognition probabilities of objects rather than just with boolean values [KD05].

Apart from precision and recall, some other researches suggested different approaches for evaluation. This should be mentioned that most of the suggested metric are only applicable for evaluating one or two specific task of table processing and can not be proposed as a comprehensive measure for all table processing tasks.

- Edit-distance: For the purposes of this approach, a table is considered to be a region consisting of one or more contiguous text lines and a document contains some number of non-overlapping (but possibly adjacent) tables listed consecutively in order of occurrence. In considering the output from table detection, it becomes evident that certain classes of errors may arise such as including non-table regions and improperly labelling them as tables (insertion errors), tables missed completely (deletion errors), larger tables broken into a number of smaller ones (splitting errors), and groups of smaller tables combined to form larger ones (merging errors). This leads naturally to the use of an edit distance model for assessing the results of table detection [HKLW02]. It is obvious that this approach can be applied just for locating and segmenting tables.

- Error-based: Kboubi et. al. [KCA05] identified and classified the error types may occur in table recognition process (specifically table locating and segmenting). These errors referring to splitting or fusion of cells, lines, columns or tables and table non-detection. They examined their evaluation system and related performance measures on five categories of OCR software. Also, Shafait et.al. [SS10] considered main page segmentation errors and categorized all the failed scenarios in table locating in six classes presenting both ground-truth tables and tables detected by their bounding boxes.

- Colour-encoding: Shahab eta. al [SSKD10] proposed a color-coding scheme for representing table segmentation. The main idea is to use the red color channel for representing the index of the table, i.e. the table number in the current document image. The green channel represents the row number within the table, and the blue channel represents the column number. In this way, each cell in the table gets a unique color since it has a unique row, column, and table number. By comparing the pixel colors in each segment and based on

six page segmentation errors[SS10], the result of table segmented is evaluated.

- Graph-probing: It has also been used in table structure recognition to compare graphs describing the logical structure of tables. Hu et. al [HKL+01] given the table recognition result and its corresponding ground-truth, they considered comparing the two as the way of determining how well the table detection has be done.

- Table-agreement:The idea presented in [HKLW02] is to present the result table and the ground truth one as a graph and then place each of the two graphs under study inside a 'black box' capable of evaluating a set of graph-oriented operations. Agreement is computed by automatically generating queries about the number, content, topology, and indexing structure of cells in logical table structure encodings for recognizer output and ground truth. Queries are verified or contradicted by searching the other table encoding; agreement is defined as the percentage of verified queries.

7.2.4 Lack of Unified Evaluation Strategy

Considering table processing tasks from an end-to-end perspective, there is not one unified strategy for performance evaluation. This strategy can be described from two perspectives:

- Proper Moment for Evaluation: When, at which point in the processing, we should measure the performance of a table processing system depends on different factors, such as the aim of the system, the approach taken for the processing technique and so on. Some researchers suggest that the appropriate moment to evaluate a table processing system is after the final result is produced. For example, if the endpoint is to extract particular information from tables, eval-

uation is based on how closely the extracted information matches what was expected [OR08]. However, others claim that, since a table processing system involves many steps, the overall performance of the system often relies on how well the intermediate steps interact, and measuring the performance of intermediate steps is important [Lon10, eSJT06]. Furthermore, it can be used as a means of improving each of the steps independently [eSJT06]. Considering the atomic tasks involved in table processing, some researchers are trying to define some unified "moments" for evaluation. For instance, Long [LCD06] describes three levels of region extraction for evaluation: table, row and cell levels.

- How to Aggregate the Results: Another issue is how to combine results over the whole dataset. Should evaluation be done document-based or corpus-based? How the different metrics should be aggregated to be a meaningful performance indicator for whole system? Gobel [GHOO13] suggested to first calculate these performance scores (relying on recall and precision) for each document separately and then calculate the average based on the document scores. Following this approach, documents will have an equal weighting and the result is not skewed by the few documents containing tables with hundreds or thousands of cells. Yildiz et al. [YKM05], first calculated the average table area and cell recall and precision for each document, and averaged these figures throughout the complete dataset. Sometimes neither approach is suitable. For instance, Tamir [Has10a] was unable to calculate precision values for documents where no tables or cells were detected. They calculate totals by averaging the total numbers of detected cells directly over the complete dataset. And as the result of this method, documents containing more information (more table areas/cells) are also given more weighting in the final result.

TEXUS Name	Synonym Terms
Locating	table identification [LMG08b], table detection [FGB+11], table area selection [eSJT03], table boundary detection [LBMG09], table spotting [FGB+11]
Segmenting	table recognition [Lon10], table decomposition[LMG08b], table text blocks discovery [Hur01], table internal structure detection[SSKD10]
Functional Analysis	table header detection [SJKN10], table format verification [NT12], table augmentation[NPJ+10]
Structural Analysis	table factoring [Jin13], table abstraction [Wan96], detecting table read-wise pattern [Yan02]. table access structure detection[KD01]

Table 7.2: TEXUS Tasks and the Synonyms

7.2.5 Appropriate Terms/Names for Evaluation

It is difficult to adequately compare the quality of results from different table processing systems even when they *seem* to work on the same task [eS10, FTT+12, HKL+01]. We discuss this issue from two perspectives. First, different terms are used to describe systems that perform table extraction tasks belonging in the same scope. For example, there is much work focused on locating tables in a document. In these works, there is a wide variation in terminology to describe their goals, e.g. *table identification*, *table detection*, *table spotting*, etc. In the terms defined in TEXUS, we can describe all of these as *Table Locating*. Since TEXUS defines the atomic tasks involved in a table processing system, the goal of each system can either be mapped to one task or a composition of several tasks. Table 7.2 gives examples of terms that appear in the table processing literature and their mapping to TEXUS atomic tasks.

In fact, when there is some common understanding about these tasks, it would be possible to describe the goals of many table processing systems in terms of logically separated steps and evaluate the outputs at each "step". We believe TEXUS,

through the well-defined tasks along with concrete data models, can contribute to providing a useful framework to objectively evaluate a processing system as a whole, as well as individual sub-systems.

7.2.6 Unit of Measurements

The characteristics of the appropriate metrics which satisfy the needs of table processing tasks is another discussion in the community. One of the considerations is that the granularity level of the elements at input is not the same as that of output in a table processing task. Different suggestions were made regarding this issue. Silva [eS10] proposed two new metrics: completeness and purity. Long [LCD06] suggested multi-level evaluation methodology.

However, there is not yet any set unit of measurements in task-based evaluation. Take the locating task as an example, Liu et. al [LMG08b] evaluated their system in terms of lines, Wang [WH02a] used cells and Chen [CTT00] considered full tables. In the ICDAR'13 table competition [GHOO13], they set the measurement unit for locating at an individual character level. However, the results showed that it was not a proper choice for some cases. This approach gave more weight to the parts of tables that have more characters (e.g., there was a system having a very good precision at the character level, but did not manage to locate even one table completely). Given the main goal of the locating task, the performance on recognising table boundaries should be valued more, rather than say, recognising text areas within a table.

We believe that TEXUS could provide a basis for task-based evaluation measurement metrics, because the table elements (e.g., table boundaries, rows, columns and cells) which may be relevant for measuring performance are explicitly defined in the tasks and their data models. Following the locating task example, we may consider the "text chunks" as a measuring unit, since cells are the smallest meaningful unit of

data in a table structure.

7.3 Conclusion

In this chapter, we presented two more topics on TEXUS for further discussion. We showed that our final output in XML can be mapped to another format, using the recent W3C recommended tabular data standards. We explained how we map our multi-dimension model into the simple CSV model.

We then discussed the issues and difficulties involved in the performance evaluation of table processing systems. We explored a range of issues in detail and suggested that, in some cases, our framework – with its modular definitions and explicit output models – could provide a basis for creating some common grounds.

Chapter 8

Conclusion

This chapter highlights the contributions of the thesis and presents some future work directions before drawing final conclusions.

Recent decades have witnessed vast growth in the volume of data and information in a variety of formats. While much of this data is in structured form (such as in databases), there is a large amount of useful information in tabular form in unstructured documents (e.g. online news, government documents, corporate reports, legal acts, medical alerts and records). Converting this tabular data (with its inherent structure and relationships between data items) is critical to maximising the utility of these documents.

Tables in documents are a presentation medium for capturing the underlying semantics of a data set and conveying this to human readers. They accomplish this using layout features and spatial arrangements, which can be effectively decoded by humans but not so easily by computer systems. Enabling automated systems to extract not only the data values, but also the relationships between data items and, as much as possible, their meaning, is a challenging research problem, and

is complicated by considerable diversity in (a) table layouts (e.g. row vs column), (b) document formats (e.g. HTML, PDF, plain text), (c) data types (e.g. text, numbers, formulae, images).

Table processing has gained attention from a number of different research communities: document engineering, artificial intelligence, information extraction and retrieval, database and knowledge management. These diverse communities describe and model table from different perspectives and at different levels of abstraction, which makes it difficult to compare their work, even when the end-goals look similar. It would clearly be desirable to have a general model of table processing within which future work could be carried out and evaluated; a major goal of this thesis is to provide such a model.

The thesis began with a review of the various approaches to the *table processing* problem. Using the 4W+1H methodology, we summarised the research communities, their motivation and objectives for table processing, their different research ideas and issues, the problems each had encountered, and the methods and techniques each had applied. The review revealed that tables are discussed at three different levels: physical, logical or abstract.

The review also found that many of the systems that had been developed to address table processing either focused on specific problems (e.g. locating tables) or specific domains or types of tables. Systems that claimed to be more general were typically semi-automatic, and accompanied by a discussion on the necessity of more automatic solutions. In addition, most existing systems are implemented as "blackbox" solutions, making it impossible to conduct a detailed analysis of their effectiveness. Opaque, domain-specific systems have the additional problem that shifting to other kinds of documents requires rewriting most, if not all, of the system. The above problems indicate a clear gap in the research on general table processing.

There have been limited efforts to incorporate the concept of reusability and modular design into table processing processing, under the name "End-to-End table processing". We argue that, looking at the table processing from a wider perspective, having end-to-end table processing is not enough. A general model for tabular data management is also needed, to incorporate the storage, indexing, querying, and maintenance of both the original raw data and the extracted information. With this background and motivation, the thesis tried to answer the following questions:

- What are the characteristics of automatic end-to-end table processing systems?
- What are the atomic tasks involved in generic, domain-independent table processing?
- How can table processing systems be designed to improve reusability and extensibility?
- How can tables be understood and interpreted in a more domain-independent way?

The thesis makes the following contributions to address these questions:

- We propose a general model for table processing systems. The model identifies discrete steps in table processing and associates a well-defined data model with each step. The model does not contain any domain-specific aspects and so should be applicable in a wide range of table processing applications.
- We propose TEXUS, a framework for end-to-end task-based table extraction and understanding. Having identified the individual tasks and their input and output as part of our data model, TEXUS realises the tasks as a series of modules with well-defined interfaces. Having partitioned the task into defined

steps allows us to investigate in detail the techniques applicable to effective realisation of each step. The ultimate goal here is to promote the investigation of alternative strategies for the individual steps within a framework that provides a basis for systematic and standard evaluation of table processing systems.

- We have designed a service-based architecture for the realisation of TEXUS to show the feasibility of the framework in practice. Services can be consumed by end-users via directly programming the access to the services or through a well-established service composition tool.

- We have implemented six components as REST-based services to create a complete end-to-end table processing pipeline. The pipeline receives PDF documents as input and provides a set of Abstract Tables as output. The modular structure allows alternative implementations of the components to be developed and compared far more easily than in existing systems.

In the detailed discussion of the components, we highlight the following features that are unique to our system:

- We developed a PDF Wrapper system purposely designed for table processing. The input to the first module of our system is a table-oriented document model, which captures the essential elements required by a table processing system, and can be used in composition with any table processing system. The main elements of the document model are formally defined, based on the notion of a *Text Chunk*, a <C,S,F>tuple which defines the basic attributes of document items.

- We developed an automatic table extraction sub-system. This consists of two components: one to perform table location and the other table segmentation.

The input to table extraction is our table-oriented document model, and the output is a set of segmented tables based on the TEXUS model. The effectiveness of each component and the table extraction as whole has been evaluated using several corpuses which are well-used in the table processing community. The results show that our system is significantly better than existing academic systems.

- We developed an automatic table understanding sub-system. This consists of two components: functional analysis and structural analysis. We proposed a number of novel techniques for improving the recognition of several important table structures, by analysing stub patterns, layout features and content features. The proposed system is application and domain independent and was also evaluated against the same corpuses. Based on the evaluation results, it improves the understanding of multi-dimensional tables.

Our end-to-end table processing system produces Abstract Tables as output. These are expressed in XML format, and capture the semantics of the table and its data, but are not immediately suitable for use in applications. In order to demonstrate that Abstract Tables can provide the basis for further use of the extracted data, we show how they can be mapped to the widely-used CSV format. In doing so, we followed the W3C recommendation for modelling tabular data and provided schemas and mapping algorithms between Abstract Tables and the W3C CSV model.

We also provided a summary of the challenges involved in table processing systems evaluation and explained how TEXUS, as a task-based framework, can address some of the current issues.

8.1 Future Work

The general model for table processing developed in this thesis provides new opportunities for further research on the general problem of table extraction and manipulation. For example:

- Operations on abstract tables: We have shown that the output of the system can be mapped to standard data formats (e.g. CSV) and processed via those. However, abstract tables provide the opportunity to manipulate whole tables as "first-class objects" which can be compared and queried using a formal language. Different types of similarity functions can be defined based on the categories, data, metadata and access paths. This could provide a basis for a range of new table processing applications, including more efficient information retrieval.

- Manipulation of table extraction intermediate results: Our task-based framework and modular design provides the opportunity to manipulate the intermediate results in the table processing pipeline. Each task in the pipeline reads and writes data using a well-defined format (specified in XML). This could allow us the extract data from a specific point in the pipeline, make changes, and feed it back into subsequent stages of the pipeline. For example, we could apply dynamic runtime corrections to the output of the table location module to produce overall more accurate table processing results.

- Extending the range of input document formats: We currently extract and map the PDF document elements to our table-oriented document model. This covers a wide range of documents, but there are other important document formats which often contain tables which could be usefully extracted. Providing wrappers for formats such as HTML and PPT would extend the range of

documents that can be processed. Note that our modular processing structures means that we simply need to develop a mapping from each input format to our document model, and the rest of the pipeline is unaffected.

- Formalising and extending table processing evaluation: As explained in Chapter 7, standard evaluation methodologies for table processing are not yet well-developed. We believe that our task-based approach could provide the basis on which to develop a set of standard metrics for table processing evaluation. This, in turn, could lead to the development of standard benchmarks which could be used to evaluate future table processing systems.

8.2 Summary

We have developed a modular architecture for table processing systems to extract and structure tabular data in a wide range of documents. For each of the modules in the system, we have investigated techniques for efficient and effective mapping of its input elements to a set of well-defined output elements. This architecture provides a number of benefits in solving the table processing problem, as discussed through the thesis, and opens up new avenues for investigation as outlined above.

Bibliography

[AA03] Akira Amano and Naoki Asada. Graph grammar based analysis system of complex table form document. In *null*, page 916. IEEE, 2003.

[ACPP11] Apostolos Antonacopoulos, Christian Clausner, Christos Papadopoulos, and Stefan Pletschacher. Historical document layout analysis competition. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1516–1520. IEEE, 2011.

[Act] Activiti. Activiti BPM Platform. http://activiti.org.

[AG02] Saleh Alrashed and WA Gray. Detection approaches for table semantics in text. In *International Workshop on Document Analysis Systems*, pages 287–290. Springer, 2002.

[AL12] Norah Alrayes and Wo-Shun Luk. Automatic transformation of multi-dimensional web tables into data cubes. In *International Conference on Data Warehousing and Knowledge Discovery*, pages 81–92. Springer, 2012.

[Anj01] Anjo Anjewierden. Aidas: Incremental logical structure discovery in pdf documents. In *icdar*, page 0374. IEEE, 2001.

[ATV13] Nikita Astrakhantsev, Denis Turdakov, and Natalia Vassilieva. Semi-automatic data extraction from tables. In *RCDL*, pages 14–20, 2013.

[BDBIS13] Patrice Buche, Juliette Dibie-Barthelemy, Liliana Ibanescu, and Luciana Soler. Fuzzy web data tables integration guided by an ontological and terminological resource. *Knowledge and Data Engineering, IEEE Transactions on*, 25(4):805–819, 2013.

[Bel08] Michael Bell. Service-oriented modeling. *John Willey & Sons, Inc*, 2008.

[BOO15] Florence Folake Babatunde, Bolanle Adefowoke Ojokoh, and Samuel Adebayo Oluwadare. Automatic table recognition and extraction from heterogeneous documents. *Journal of Computer and Communications*, 3(12):100, 2015.

[BTEL15] Katrin Braunschweig, Maik Thiele, Julian Eberius, and Wolfgang Lehner. Column-specific context extraction for web tables. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 1072–1077. ACM, 2015.

[Bur07] Radek Burget. Automatic document structure detection for data integration. In *Business information systems*, pages 391–397. Springer, 2007.

[Caf09] Michael J Cafarella. Extracting and querying a comprehensive web database. In *CIDR*, 2009.

[CBC+07] Eric Chu, Akanksha Baid, Ting Chen, AnHai Doan, and Jeffrey Naughton. A relational approach to incrementally extracting and querying structure in unstructured data. In *Proceedings of the 33rd international conference on Very large data bases*, pages 1045–1056. VLDB Endowment, 2007.

[CCF+06] Julien Carme, Michal Ceresna, Oliver Frölich, Georg Gottlob, Tamir Hassan, Marcus Herzog, Wolfgang Holzinger, and Bernhard Krüpl. The lixto project: Exploring new frontiers of web data extraction. In *British National Conference on Databases*, pages 1–15. Springer, 2006.

[CDKL07] Francisco Curbera, Matthew Duftler, Rania Khalaf, and Douglas Lovell. *Bite: Workflow composition for the web*. Springer, 2007.

[CHCG15] Xu Chu, Yeye He, Kaushik Chakrabarti, and Kris Ganjam. Tegra: Table extraction by global record alignment. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1713–1728. ACM, 2015.

[CHJ02] William W Cohen, Matthew Hurst, and Lee S Jensen. A flexible learning system for wrapping tables and lists in html documents. In *Proceedings of the 11th international conference on World Wide Web*, pages 232–241. ACM, 2002.

[CL12] Jin Chen and Daniel Lopresti. Model-based tabular structure detection and recognition in noisy handwritten documents. In *Frontiers in Handwriting Recognition (ICFHR), 2012 International Conference on*, pages 75–80. IEEE, 2012.

[CL14] Bertrand Coüasnon and Aurélie Lemaitre. Recognition of tables and forms. In *Handbook of Document Image Processing and Recognition*, pages 647–677. Springer, 2014.

[Coü06] Bertrand Coüasnon. Dmos, a generic document recognition method: Application to table structure analysis in a general and in a specific way. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):111–122, 2006.

[CP10] Eric Crestan and Patrick Pantel. Web-scale knowledge extraction from semi-structured tables. In *Proceedings of the 19th international conference on World wide web*, pages 1081–1082. ACM, 2010.

[CP11] Eric Crestan and Patrick Pantel. Web-scale table census and classification. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 545–554. ACM, 2011.

[CPV13] Alexandru Constantin, Steve Pettifer, and Andrei Voronkov. Pdfx: fully-automated pdf-to-xml conversion of scientific literature. In *Proceedings of the 2013 ACM symposium on Document engineering*, pages 177–180. ACM, 2013.

[CTT00] Hsin-Hsi Chen, Shih-Chung Tsai, and Jin-He Tsai. Mining tables from large scale html texts. In *Proceedings of the 18th conference on Computational linguistics-Volume 1*, pages 166–172. Association for Computational Linguistics, 2000.

[Cun02] Hamish Cunningham. Gate, a general architecture for text engineering. *Computers and the Humanities*, 36(2):223–254, 2002.

[DMFE13] Nicola Di Mauro, Stefano Ferilli, and Floriana Esposito. Learning to recognize critical cells in document tables. In *Digital Libraries and Archives*, pages 105–116. Springer, 2013.

[DNR+09] AnHai Doan, Jeffrey F Naughton, Raghu Ramakrishnan, Akanksha Baid, Xiaoyong Chai, Fei Chen, Ting Chen, Eric Chu, Pedro DeRose, Byron Gao, et al. Information extraction challenges in managing unstructured data. *ACM SIGMOD Record*, 37(4):14–20, 2009.

[DSEG11] Florian Deckert, Benjamin Seidler, Markus Ebbecke, and Michael Gillmann. Table content understanding in smartfix. In *2011 International Conference on Document Analysis and Recognition*, pages 488–492. IEEE, 2011.

[DSFG+12] Anish Das Sarma, Lujun Fang, Nitin Gupta, Alon Halevy, Hongrae Lee, Fei Wu, Reynold Xin, and Cong Yu. Finding related tables. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, pages 817–828. ACM, 2012.

[DT+14] David Doermann, Karl Tombre, et al. *Handbook of Document Image Processing and Recognition*. Springer, 2014.

[DV11] Jan Dedek and P Vojtas. Semantic annotation semantically: Using a shareable extraction ontology and a reasoner. *Proceedings of SEMAPRO*, pages 29–34, 2011.

[EAS13] Ivan Ermilov, Sören Auer, and Claus Stadler. User-driven semantic mapping of tabular data. In *Proceedings of the 9th International Conference on Semantic Systems*, pages 105–112. ACM, 2013.

[EFBDM08] Floriana Esposito, Stefano Ferilli, Teresa MA Basile, and Nicola Di Mauro. Machine learning for digital document processing: from layout analysis to metadata extraction. In *Machine learning in document analysis and recognition*, pages 105–138. Springer, 2008.

[EHLN06] 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.

[EKNS11] David W Embley, Mukkai Krishnamoorthy, George Nagy, and Sharad Seth. Factoring web tables. In *International Conference on Industrial, Engineering and Other Applications of Applied Intelligent Systems*, pages 253–263. Springer, 2011.

[EKNS16] David W Embley, Mukkai S Krishnamoorthy, George Nagy, and Sharad Seth. Converting heterogeneous statistical tables on the web to searchable databases. *International Journal on Document Analysis and Recognition (IJDAR)*, 19(2):119–138, 2016.

[ELN06] David W Embley, Daniel Lopresti, and George Nagy. Notes on contemporary table recognition. In *Document Analysis Systems VII*, pages 164–175. Springer, 2006.

[Emb04] David W Embley. Toward semantic understanding: an approach based on information extraction ontologies. In *Proceedings of the 15th Australasian database conference-Volume 27*, pages 3–12. Australian Computer Society, Inc., 2004.

[Erl05] Thomas Erl. *Service-oriented architecture: concepts, technology, and design*. Pearson Education India, 2005.

[eS10] Ana Costa e Silva. *Parts that add up to a whole: a framework for the analysis of tables*. PhD thesis, The University of Edinburgh, 2010.

[eS11] Ana Costa e Silva. Metrics for evaluating performance in document analysis: application to tables. *International Journal on Document Analysis and Recognition (IJDAR)*, 14(1):101–109, 2011.

[eSJT03] Ana Costa e Silva, Alípio Jorge, and Luís Torgo. Automatic selection of table areas in documents for information extraction. In *Progress in Artificial Intelligence*, pages 460–465. Springer, 2003.

[eSJT06] 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.

[ESN14] David W Embley, Sharad Seth, and George Nagy. Transforming web tables to a relational database. In *2014 22nd International Conference on Pattern Recognition (ICPR)*, pages 2781–2786. IEEE, 2014.

[ETL05] David W Embley, Cui Tao, and Stephen W Liddle. Automating the extraction of data from html tables with unknown structure. *Data & Knowledge Engineering*, 54(1):3–28, 2005.

[FFT+08] Bettina Fazzinga, Sergio Flesca, Andrea Tagarelli, Salvatore Garruzzo, and Elio Masciari. A wrapper generation system for pdf documents. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 442–446. ACM, 2008.

[FGB+11] Jing Fang, Liangcai Gao, Kun Bai, Ruiheng Qiu, Xin Tao, and Zhi Tang. A table detection method for multipage pdf documents via visual seperators and tabular structures. In *Document Analysis and Recognition (ICDAR)*, pages 779–783. IEEE, 2011.

[FMTG12] Jing Fang, Prasenjit Mitra, Zhi Tang, and C Lee Giles. Table header detection and classification. In *AAAI*, 2012.

[FSCG03] Robert P Futrelle, Mingyan Shao, Chris Cieslik, and Andrea Elaina Grimes. Extraction, layout analysis and classification of diagrams in pdf documents. In *null*, page 1007. IEEE, 2003.

[FTT+12] Jing Fang, Xin Tao, Zhi Tang, Ruiheng Qiu, and Ying Liu. Dataset, ground-truth and performance metrics for table detection evaluation. In *Document Analysis Systems (DAS), 2012 10th IAPR International Workshop on*, pages 445–449. IEEE, 2012.

[GBH+07] Wolfgang Gatterbauer, Paul Bohunsky, Marcus Herzog, Bernhard Krüpl, and Bernhard Pollak. Towards domain-independent information extraction from web tables. In *Proceedings of the 16th international conference on World Wide Web*, pages 71–80. ACM, 2007.

[GDPP05] Basilios Gatos, Dimitrios Danatsas, Ioannis Pratikakis, and Stavros J Perantonis. Automatic table detection in document images. In *Pattern Recognition and Data Mining*, pages 609–618. Springer, 2005.

[GH13] Kyle Goslin and Martin Hofmann. Cross domain assessment of document to html conversion tools to quantify text and structural loss during document analysis. In *Intelligence and Security Informatics Conference (EISIC), 2013 European*, pages 100–105. IEEE, 2013.

[GHJ+10] Hector Gonzalez, Alon Halevy, Christian S Jensen, Anno Langen, Jayant Madhavan, Rebecca Shapley, and Warren Shen. Google fusion tables: data management, integration and collaboration in the cloud. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 175–180. ACM, 2010.

[GHOO12] Max Göbel, Tamir Hassan, Ermelinda Oro, and Giorgio Orsi. A methodology for evaluating algorithms for table understanding in pdf documents. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 45–48. ACM, 2012.

[GHOO13] Max Gobel, Tamir Hassan, Ermelinda Oro, and Giorgio Orsi. ICDAR 2013 table competition. In *12th International Conference on Document Analysis and Recognition (ICDAR'13)*, pages 1449–1453. IEEE, 2013.

[GHPS05] Hélene Gagliardi, Ollivier Haemmerlé, Nathalie Pernelle, and Fatiha Saïs. An automatic ontology-based approach to enrich tables semantically. In *AAAI Context and Ontologies Workshop*, 2005.

[GJJH12] Yingqin Gu, Lei Ji, Ziheng Jiang, and Jun He. Endless and scalable knowledge table extraction from semi-structured websites. In *Data Mining Workshops (ICDMW), 2012 IEEE 12th International Conference on*, pages 835–842. IEEE, 2012.

[GLM03] Yassine Gargouri, Bernard Lefebvre, and Jean-guy Meunier. Ontology maintenance using textual analysis. In *Proc. 7TH World Multi Conference on Systemics, Cybernetics and Informatics, USA. List of Figures Figure*, volume 1. Citeseer, 2003.

[GTL+11] Liangcai Gao, Zhi Tang, Xiaofan Lin, Ying Liu, Ruiheng Qiu, and Yongtao Wang. Structure extraction from pdf-based book documents. In *Proceedings of the 11th annual international ACM/IEEE joint conference on Digital libraries*, pages 11–20. ACM, 2011.

[GW06] Chuck Geschke and John Warnock. Pdf reference. Technical Report Version 1.7, Adobe Systems Incorporated, November 2006.

[GWWS01] Ning Gu, Guowen Wu, Xiaoyuan Wu, and Baile Shi. Extracting web table information in cooperative learning activities based on abstract semantic model. In *Computer Supported Cooperative Work in Design, The Sixth International Conference on, 2001*, pages 492–497. IEEE, 2001.

[Han00] John C Handley. Table analysis for multiline cell identification. In *Photonics West 2001-Electronic Imaging*, pages 34–43. International Society for Optics and Photonics, 2000.

[Has09] Tamir Hassan. User-guided wrapping of pdf documents using graph matching techniques. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 631–635. IEEE, 2009.

[Has10a] Tamir Hassan. Towards a common evaluation strategy for table structure recognition algorithms. In *Proceedings of the 10th ACM symposium on Document engineering*, pages 255–258. ACM, 2010.

[Has10b] Tamir Hassan. *User-Guided Information Extraction from Print-Oriented Documents*. PhD thesis, Institut für Informationssysteme, 2010.

[HB05] Tamir Hassan and Robert Baumgartner. Intelligent text extraction from pdf documents. In *Computational Intelligence for Modelling, Control and Automation, 2005 and International Conference on Intelligent Agents, Web Technologies and Internet Commerce, International Conference on*, volume 2, pages 2–6. IEEE, 2005.

[HB07] Tamir Hassan and Robert Baumgartner. Table recognition and understanding from pdf files. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, pages 1143–1147. IEEE, 2007.

[HCHL07] Chiung-Wei Huang, Chih-Yuan Chien, Chun-Nan Hsu, and Hahn-Ming Lee. Automatic hypertext table understanding by using logical structure description algorithm. In *Innovative Computing, Information and Control, 2007. ICICIC'07. Second International Conference on*, pages 247–247. IEEE, 2007.

[HFP+08] Lushan Han, Tim Finin, Cynthia Parr, Joel Sachs, and Anupam Joshi. *RDF123: from Spreadsheets to RDF*. Springer, 2008.

[HKL+01] Jianying Hu, Ramanujan Kashi, Daniel Lopresti, George Nagy, and Gordon Wilfong. Why table ground-truthing is hard. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 129–133. IEEE, 2001.

[HKLW02] Jianying Hu, Ramanujan S Kashi, Daniel Lopresti, and Gordon T Wilfong. Evaluating the performance of table processing algorithms. *International Journal on Document Analysis and Recognition*, 4(3):140–153, 2002.

[HL14] Jianying Hu and Ying Liu. Analysis of documents born digital. *Handbook of Document Image Processing and Recognition*, pages 775–804, 2014.

[HS10] Nattapon Harnsamut and Naiyana Sahavechaphan. Mining for attributes and values in tables. In *Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 53–60. ACM, 2010.

[Hur00a] Matthew Hurst. *The interpretation of tables in texts*. PhD thesis, The University of Edinburgh, 2000.

[Hur00b] Matthew Hurst. Layout and language: An efficient algorithm for detecting text blocks based on spatial and linguistic evidence. In *Photonics West 2001-Electronic Imaging*, pages 56–67. International Society for Optics and Photonics, 2000.

[Hur01] Matthew Hurst. Layout and language: Exploring text block discovery in tables using linguistic resources. In *International Conference on Document Analysis and Recognition*, pages 523–527, 2001.

[Hur06] Matthew Hurst. Towards a theory of tables. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):123–131, 2006.

[IFS05] Yasuto Ishitani, Kosei Fume, and Kazuo Sumita. Table structure analysis based on cell classification and cell modification for XML document transformation. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1247–1252. IEEE, 2005.

[Inca] Bonitasoft Inc. Bonita BPM. http://www.bonitasoft.com.

[Incb] Adobe Systems Incorporated. *PDF Reference*.

[JCYT15] Changjiang Jia, Yan Cai, Yuen Tak Yu, and TH Tse. 5w+ 1h pattern: A perspective of systematic mapping studies and a case study on cloud software testing. *Journal of Systems and Software*, 2015.

[Jin13] Dongpu Jin. An algebraic approach to building category parse trees for web tables. *2012 NCUR*, 2013.

[JKN+09] 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.

[JN08] Piyushee Jha and George Nagy. Wang notation tool: Layout independent representation of tables. In *ICPR 2008*, pages 1–4. IEEE, 2008.

[JOp] JOpera. JOpera for Eclipse. http://www.jopera.org.

[JY09] Deliang Jiang and Xiaohu Yang. Converting pdf to html approach based on text detection. In *Proceedings of the 2nd International Conference on Interaction Sciences: Information Technology, Culture and Human*, pages 982–985. ACM, 2009.

[Kah03] Wolfram Kahl. Compositional syntax and semantics of tables. *SQRL Report*, 15, 2003.

[KCA05] Ferihane Kboubi, Anja Habacha Chabi, and Mohamed Ben Ahmed. Table recognition evaluation and combination methods. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1237–1241. IEEE, 2005.

[KD01] Thomas Kieninger and Andreas Dengel. Applying the t-recs table recognition system to the business letter domain. In *Document Analysis and Recognition, 2001. Proceedings. Sixth International Conference on*, pages 518–522. IEEE, 2001.

[KD05] Thomas Kieninger and Andreas Dengel. An approach towards benchmarking of table structure recognition results. In *Document Analysis and Recognition, 2005. Proceedings. Eighth International Conference on*, pages 1232–1236. IEEE, 2005.

[KGK+07] Sunil Kumar, Rajat Gupta, Nitin Khanna, Santanu Chaudhury, and Shiv Dutt Joshi. Text extraction and document image segmentation using matched wavelets and mrf model. *Image Processing, IEEE Transactions on*, 16(8):2117–2128, 2007.

[KH06] Bernhard Krüpl and Marcus Herzog. Visually guided bottom-up table detection and segmentation in web documents. In *Proceedings of the 15th international conference on World Wide Web*, pages 933–934. ACM, 2006.

[KHG05] Bernhard Krüpl, Marcus Herzog, and Wolfgang Gatterbauer. Using visual cues for extraction of tabular data from arbitrary html documents. In *Special interest tracks and posters of the 14th international conference on World Wide Web*, pages 1000–1001. ACM, 2005.

[KHKL12] Seongchan Kim, Keejun Han, Soon Young Kim, and Ying Liu. Scientific table type classification in digital library. In *Proceedings of the 2012 ACM symposium on Document engineering*, pages 133–136. ACM, 2012.

[KL06] Yeon-Seok Kim and Kyong-Ho Lee. Generating structured documents from html tables. In *2006 International Conference on Hybrid Information Technology*, volume 2, pages 605–610. IEEE, 2006.

[KL08] Yeon-Seok Kim and Kyong-Ho Lee. Extracting logical structures from html tables. *Computer Standards & Interfaces*, 30(5):296–308, 2008.

[KL11] Seongchan Kim and Ying Liu. Functional-based table category identification in digital library. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 1364–1368. IEEE, 2011.

[KLU14] 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.

[LBG11] Ying Liu, Kun Bai, and Liangcai Gao. An efficient pre-processing method to identify logical components from pdf documents. In *Advances in Knowledge Discovery and Data Mining*, pages 500–511. Springer, 2011.

[LBMG07a] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Automatic searching of tables in digital libraries. In *Proceedings of the 16th international conference on World Wide Web*, pages 1135–1136. ACM, 2007.

[LBMG07b] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Tablerank: A ranking algorithm for table search and retrieval. In *PROCEEDINGS OF THE NATIONAL CONFERENCE ON ARTIFICIAL INTELLIGENCE*, volume 22, page 317. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999, 2007.

[LBMG07c] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, pages 91–100. ACM, 2007.

[LBMG09] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines. In *10th International Conference on Document Analysis and Recognition (ICDAR'09)*, pages 1006–1010. IEEE, 2009.

[LCD06] Vanessa Long, Steve Cassidy, and Robert Dale. A multi-level table evaluation method for plain text documents. In *Extended Abstracts of the 7th International Association for Pattern Recognition Workshop on Document Analysis Systems (DAS 2006)*, pages 21–24, 2006.

[LCZW10] Huilin Liu, Chen Chen, Liwei Zhang, and Guoren Wang. The research of label-mapping-based entity attribute extraction. In *Progress in Informatics and Computing (PIC), 2010 IEEE International Conference on*, volume 1, pages 635–639. IEEE, 2010.

[LDC05] Vanessa Long, Robert Dale, and Steve Cassidy. A model for detecting and merging vertically spanned table cells in plain text documents. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1242–1246. IEEE, 2005.

[LE08] Stephen Lynn and David W Embley. Semantically conceptualizing and annotating tables. In *The Semantic Web*, pages 345–359. Springer, 2008.

[Liu09] Ying Liu. *Tableseer: automatic table extraction, search, and understanding*. PhD thesis, The Pennsylvania State University, 2009.

[LMG08a] Ying Liu, Prasenjit Mitra, and C Lee Giles. A fast preprocessing method for table boundary detection: Narrowing down the sparse lines using solely coordinate information. In *The Eighth IAPR International Workshop on Document Analysis Systems*, pages 431–438. IEEE, 2008.

[LMG08b] Ying Liu, Prasenjit Mitra, and C Lee Giles. Identifying table boundaries in digital documents via sparse line detection. In *Proceedings of the 17th ACM Conference on Information and Knowledge Management*, pages 1311–1320. ACM, 2008.

[LMGB06] Ying Liu, Prasenjit Mitra, C Lee Giles, and Kun Bai. Automatic extraction of table metadata from digital documents. In *Proceedings of the 6th ACM/IEEE-CS joint conference on Digital libraries*, pages 339–340. ACM, 2006.

[LN99] Daniel Lopresti and George Nagy. A tabular survey of automated table processing. In *Graphics Recognition Recent Advances*, pages 93–120. Springer, 1999.

[LN00] Daniel Lopresti and George Nagy. A tabular survey of automated table processing. In *Graphics Recognition Recent Advances*, pages 93–120. Springer, 2000.

[Lon09] Vanessa Long. An rdf-based blackboard architecture for improving table analysis. In *Document Analysis and Recognition, 2009. ICDAR'09. 10th International Conference on*, pages 916–920. IEEE, 2009.

[Lon10] Vanessa Long. *An Agent-Based Approach to Table Recognition and Interpretation*. PhD thesis, Macquarie University Sydney, Australia, 2010.

[LRGMVG+11] Eduardo Lupiani-Ruiz, Ignacio GarcíA-Manotas, Rafael Valencia-GarcíA, Francisco GarcíA-SáNchez, Dagoberto Castellanos-Nieves, Jesualdo TomáS FernáNdez-Breis, and Juan Bosco CamóN-Herrero. Financial news semantic search engine. *Expert systems with applications*, 38(12):15565–15572, 2011.

[LSC10] Girija Limaye, Sunita Sarawagi, and Soumen Chakrabarti. Annotating and searching web tables using entities, types and relationships. *Proceedings of the VLDB Endowment*, 3(1-2):1338–1347, 2010.

[LTSX06] Juanzi Li, Jie Tang, Qiang Song, and Peng Xu. Table detection from plain text using machine learning and document structure. In *Frontiers of WWW Research and Development-APWeb 2006*, pages 818–823. Springer, 2006.

[LY10] Xu Lei and Ren Yong. Ontology generation from web tables: A 1+ 1+ n approach. In *Information Technology and Applications (IFITA), 2010 International Forum on*, volume 1, pages 234–239. IEEE, 2010.

[Mar08] Simone Marinai. Introduction to document analysis and recognition. In *Machine learning in document analysis and recognition*, pages 1–20. Springer, 2008.

[MCDC06] Sekhar Mandal, SP Chowdhury, Amit K Das, and Bhabatosh Chanda. A simple and effective table detection system from document images. *International Journal of Document Analysis and Recognition (IJDAR)*, 8(2-3):172–182, 2006.

[MHON11] Rosmayati Mohemad, Abdul Razak Hamdan, Zulaiha Ali Othman, and Noor MaizuraMohamad Noor. Automatic document structure analysis of structured pdf files. *International Journal of New Computer Architectures and their Applications (IJNCAA)*, 1(2):404–411, 2011.

[Mul10] Varish Mulwad. *T2LD-An automatic framework for extracting, interpreting and representing tables as Linked Data*. PhD thesis, University of Maryland, 2010.

[Mul15] Varish Vyankatesh Mulwad. *TABEL–A Domain Independent and Extensible Framework for Inferring the Semantics of Tables*. PhD thesis, University of Maryland, 2015.

[Nag12] George Nagy. Learning the characteristics of critical cells from web tables. In *ICPR 2012*, pages 1554–1557. IEEE, 2012.

[NEKS15] 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.

[NJ07] Anoop M Namboodiri and Anil K Jain. Document structure and layout analysis. In *Digital Document Processing*, pages 29–48. Springer, 2007.

[NPJ+10] George Nagy, Raghav Padmanabhan, RC Jandhyala, W Silversmith, and MS Krishnamoorthy. Table metadata: Headers, augmentations and aggregates. In *Ninth IAPR International Workshop on Document Analysis Systems*, 2010.

[NSE14] George Nagy, Sharad Seth, and David W Embley. End-to-end conversion of html tables for populating a relational database. In *DAS 2014*, pages 222–226. IEEE, 2014.

[NSJ+11] George Nagy, Sachin Seth, Dongpu Jin, David W Embley, Spencer Machado, and Mohan Krishnamoorthy. Data extraction from web tables: The devil is in the details. In *Document Analysis and Recognition (ICDAR), 2011 International Conference on*, pages 242–246. IEEE, 2011.

[NT12] George Nagy and Mangesh Tamhankar. Vericlick: an efficient tool for table format verification. In *IS&T/SPIE Electronic Imaging*, pages 1–9, 2012.

[Nur13] Anssi Nurminen. *Algorithmic extraction of data in tables in PDF documents*. PhD thesis, TAMPERE University Of Technology, 2013.

[OR08] Ermelinda Oro and Massimo Ruffolo. Xonto: An ontology-based system for semantic information extraction from pdf documents. In *20th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'08)*, volume 1, pages 118–125. IEEE, 2008.

[OR09] Ermelinda Oro and Massimo Ruffolo. PDF-TREX:an approach for recognizing and extracting tables from pdf documents. In *ICDAR*, pages 906–910. IEEE, 2009.

[OSO+14] Isaac Okada, Minoru Saito, Yoshiaki Oida, Hiroyuki Yamato, Kazuo Hiekata, Satoru Nakamura, and Naoto Fukada. Technique for searching tabular form documents using metadata harvested by table structure analysis. *Artificial Intelligence Research*, 3(1):p46, 2014.

[Pad09] Raghav Krishna Padmanabhan. *Table abstraction tool*. PhD thesis, Rensselaer Polytechnic Institute, 2009.

[Pan02] Ashwini Pande. *Table understanding for information retrieval*. PhD thesis, Virginia Polytechnic Institute and State University, 2002.

[PBC+02] David Pinto, Michael Branstein, Ryan Coleman, W Bruce Croft, Matthew King, Wei Li, and Xing Wei. Quasm: a system for question answering using semi-structured data. In *Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 46–55. ACM, 2002.

[PCC12] Pranit C Patil, Pramila M Chawan, and Prithviraj M Chauhan. Extracting information from tables of html document. *International Journal of Computer Science & Applications (TIJCSA)*, 1(4), 2012.

[PCS05] Aleksander Pivk, Philipp Cimiano, and York Sure. From tables to frames. *Web Semantics: Science, Services and Agents on the World Wide Web*, 3(2):132–146, 2005.

[PCS+07] Aleksander Pivk, Philipp Cimiano, York Sure, Matjaz Gams, Vladislav Rajkovič, and Rudi Studer. Transforming arbitrary tables into logical form with tartar. *Data & Knowledge Engineering*, 60(3):567–595, 2007.

[PGL06] Aleksander Pivk, Matjaz Gams, and Mitja Luštrek. Semantic search in tabular structures. *Informatica*, 30(2), 2006.

[Phi96] IT Phillips. Users reference manual for the uw english/technical document image database iii. *UW-III English/Technical Document Image Database Manual*, 1996.

[Piv06] Aleksander Pivk. Automatic ontology generation from web tabular structures. *AI Communications*, 19(1):83–85, 2006.

[PJK+10] 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.

[PMWC03] David Pinto, Andrew McCallum, Xing Wei, and W Bruce Croft. Table extraction using conditional random fields. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, pages 235–242. ACM, 2003.

[PN08] Raghav Padmanabhan and George Nagy. Query by table. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. IEEE, 2008.

[PS11] Sarang Pitale and Tripti Sharma. Information extraction tools for portable document format. *International Journal of Computer Technology and Applications*, 8:2047–2051, 2011.

[QR13] Gianluca Quercini and Chantal Reynaud. Entity discovery and annotation in tables. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 693–704. ACM, 2013.

[RBH+05] Maurizio Rigamonti, Jean-Luc Bloechle, Karim Hadjar, Denis Lalanne, and Rolf Ingold. Towards a canonical and structured representation of pdf documents through reverse engineering. In *Eighth International Conference on Document Analysis and Recognition (ICDAR'05)*, pages 1050–1054. IEEE, 2005.

[RCVF03] J-Y Ramel, Michel Crucianu, Nicole Vincent, and Claudie Faure. Detection, extraction and representation of tables. In *Document Analysis and Recognition, 2003. Proceedings. Seventh International Conference on*, pages 374–378. IEEE, 2003.

[RMS06] Davood Rafiei, Daniel L Moise, and Dabo Sun. Finding syntactic similarities between xml documents. In *17th International Workshop on Database and Expert Systems Applications (DEXA'06)*, pages 512–516. IEEE, 2006.

[RPHB12] Cartic Ramakrishnan, Abhishek Patnia, Eduard Hovy, and Gully APC Burns. Layout-aware text extraction from full-text pdf of scientific articles. *Source code for biology and medicine*, 7(1):1, 2012.

[RPS15] Roya Rastan, Hye-Young Paik, and John Shepherd. Texus: A task-based approach for table extraction and understanding. Technical Report TR-201504, School of Computer Science and Engineering, UNSW, May 2015.

[RPSH16] Roya Rastan, Hye-young Paik, John Shepherd, and Armin Haller. Automated table understanding using stub patterns. In *The 21st International Conference on Database Systems for Advanced Applications, 16-19 April 2016, Dallas, TX*, pages 533–548. Springer International Publishing, 2016.

[RYG12] Fethi A Rabhi, Lawrence Yao, and Adnene Guabtni. Adage: a framework for supporting user-driven ad-hoc data analysis processes. *Computing*, 94(6):489–519, 2012.

[Shi15] Alexey O Shigarov. Table understanding using a rule engine. *Expert Systems with Applications*, 42(2):929–937, 2015.

[Sil07] Ana Costa E Silva. New metrics for evaluating performance in document analysis tasks_application to the table case. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 1, pages 481–485. IEEE, 2007.

[SJKN10] Sharad Seth, Ramana Jandhyala, Mukkai Krishnamoorthy, and George Nagy. Analysis and taxonomy of column header categories for web tables. In *IAPR 2010*, pages 81–88. ACM, 2010.

[SKB08] Faisal Shafait, Daniel Keysers, and Thomas M Breuel. Performance evaluation and benchmarking of six-page segmentation algorithms. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(6):941–954, 2008.

[SN13] Sharad Seth and George Nagy. Segmenting tables via indexing of value cells by table headers. In *ICDAR 2013*, pages 887–891. IEEE, 2013.

[Som04] Ralph Sommerer. Presentable document format: Improved on-demand pdf to html conversion. Technical report, Technical Report MSR-TR-2004-119, Microsoft Research (MSR), 2004.

[SS10] Faisal Shafait and Ray Smith. Table detection in heterogeneous documents. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 65–72. ACM, 2010.

[SSKD10] Asif Shahab, Faisal Shafait, Thomas Kieninger, and Andreas Dengel. An open approach towards the benchmarking of table structure recognition systems. In *Proceedings of the 9th IAPR International Workshop on Document Analysis Systems*, pages 113–120. ACM, 2010.

[TAN+14] D Yu Turdakov, NA Astrakhantsev, Ya R Nedumov, AA Sysoev, IA Andrianov, VD Mayorov, DG Fedorenko, AV Korshunov, and Sergei D Kuznetsov. Texterra: A framework for text analysis. *Programming and Computer Software*, 40(5):288–295, 2014.

[Tav] Taverna. Taverna workflow management system. http://www.taverna.org.uk.

[tBC04] Richard *Zanibbi*, Dorothea Blostein, and James R Cordy. A survey of table recognition. *Document Analysis and Recognition*, 7(1):1–16, 2004.

[TE09] Cui Tao and David W Embley. Automatic hidden-web table interpretation, conceptualization, and semantic annotation. *Data & Knowledge Engineering*, 68(7):683–703, 2009.

[TEL+05] Yuri A Tijerino, David W Embley, Deryle W Lonsdale, Yihong Ding, and George Nagy. Towards ontology generation from tables. *World Wide Web*, 8(3):261–285, 2005.

[THYX02] Chew Lim Tan, Weihua Huang, Zhaohui Yu, and Yi Xu. Imaged document text retrieval without ocr. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(6):838–844, 2002.

[TK15] Jeni Tennison and Gregg Kellogg". Model for tabular data and metadata on the web. W3C working draft, W3C, Dec 2015.

[VHM+11] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Paşca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proceedings of the VLDB Endowment*, 4(9):528–538, 2011.

[Wan96] Xinxin Wang. *Tabular abstraction, editing, and formatting*. PhD thesis, University of Waterloo, 1996.

[WCC08] Chung-Chih Wu, Chien-Hsing Chou, and Fu Chang. A machine-learning approach for analyzing document layout structures with two reading orders. *Pattern Recognition*, 41(10):3200–3213, 2008.

[WCM06] Xing Wei, Bruce Croft, and Andrew McCallum. Table extraction for answer retrieval. *Information Retrieval*, 9(5):589–611, 2006.

[WCP04] Xing Wei, Bruce Croft, and David Pinto. Question answering performance on table data. In *Proceedings of the 2004 annual national conference on Digital government research*, page 106. Digital Government Society of North America, 2004.

[WH02a] Yalin Wang and Jianying Hu. Detecting tables in html documents. In *Document Analysis Systems V*, pages 249–260. Springer, 2002.

[WH02b] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th international conference on World Wide Web*, pages 242–250. ACM, 2002.

[WHF+13] Katherine Wolstencroft, Robert Haines, Donal Fellows, Alan Williams, David Withers, Stuart Owen, Stian Soiland-Reyes, Ian Dunlop, Aleksandra Nenadic, Paul Fisher, Jiten Bhagat, Khalid

Belhajjame, Finn Bacall, Alex Hardisty, Abraham Nieva de la Hidalga, Maria P. Balcazar Vargas, Shoaib Sufi, , and Carole Goble. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. *Nucleic Acids Research*, 41(W1):W557–W561, 2013.

[WHHP02] Yalin Wang, M Haralick, Robert M Haralick, and Ihsin T Phillips. Document analysis: table structure understanding and zone content classification. Technical report, Citeseer, 2002.

[WL06] Dekai Wu and Ken Wing Kuen Lee. A grammatical approach to understanding textual tables using two-dimensional scfgs. In *Proceedings of the COLING/ACL on Main conference poster sessions*, pages 905–912. Association for Computational Linguistics, 2006.

[WMC09] Wern Wong, David Martinez, and Lawrence Cavedon. Extraction of named entities from tables in gene mutation literature. In *Proceedings of the Workshop on Current Trends in Biomedical Natural Language Processing*, pages 46–54. Association for Computational Linguistics, 2009.

[WMD10] Aleš Wojnar, Irena Mlỳnková, and Jiří Dokulil. Structural and semantic aspects of similarity of document type definitions and xml schemas. *Information sciences*, 180(10):1817–1836, 2010.

[WPH01] Yalin Wangt, IT Phillipst, and Robert Haralick. Automatic table ground truth generation and a background-analysis-based table structure extraction method. In *Proceedings Sixth International Conference on Document Analysis and Recognition*, pages 528–532. IEEE, 2001.

[WPH04] Yalin Wang, Ihsin T Phillips, and Robert M Haralick. Table structure understanding and its performance evaluation. *Pattern Recognition*, 37(7):1479–1497, 2004.

[WWLN09] Yuk Wah Wong, Dominic Widdows, Tom Lokovic, and Kamal Nigam. Scalable attribute-value extraction from semi-structured text. In *Data Mining Workshops, 2009. ICDMW'09. IEEE International Conference on*, pages 302–307. IEEE, 2009.

[WWW+00] Huei-Long Wang, Shih-Hung Wu, IC Wang, Cheng-Lung Sung, Wen-Lian Hsu, and Wei-Kuan Shih. Semantic search on internet tabular information extraction for answering queries. In *Proceedings of the ninth international conference on Information and knowledge management*, pages 243–249. ACM, 2000.

[Yan02] Yingchen Yang. *Web table mining and database discovery.* PhD thesis, Simon Fraser University, 2002.

[YKM05] Burcu Yildiz, Katharina Kaiser, and Silvia Miksch. pdf2table: A method to extract table information from pdf files. In *IICAI*, pages 1773–1785, 2005.

[YL02] Yingchen Yang and Wo-Shun Luk. A framework for web table mining. In *Proceedings of the 4th international workshop on Web information and data management*, pages 36–42. ACM, 2002.

[YSGH04] Yeliz Yesilada, Robert Stevens, Carole Goble, and Shazad Hussein. Rendering tables in audio: the interaction of structure and reading styles. In *Proceedings of the 6th international ACM SIGACCESS conference on Computers and accessibility*, pages 16–23. ACM, 2004.

[Zha14a] Ziqi Zhang. Learning with partial data for semantic table interpretation. In *International Conference on Knowledge Engineering and Knowledge Management*, pages 607–618. Springer, 2014.

[Zha14b] Ziqi Zhang. Start small, build complete: Effective and efficient semantic table interpretation using tableminer. *Under transparent review: The Semantic Web Journal*, 2014.