The Pennsylvania State University

The Graduate School

Department of Information Sciences and Technology

TABLESEER: AUTOMATIC TABLE EXTRACTION, SEARCH, AND UNDERSTANDING

A Dissertation in

Information Sciences and Technology

by

Ying Liu

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

December 2009

The dissertation of Ying Liu was reviewed and approved[1] by the following:

C. Lee Giles
Professor of Information Sciences and Technology
Dissertation Co-Adviser
Co-Chair of Committee

Prasenjit Mitra
Assistant Professor of Information Sciences and Technology
Dissertation Co-Adviser
Co-Chair of Committee

Dean R. Snow
Professor of Anthropology

Tracy Mullen
Assistant Professor of Information Sciences and Technology

Henry C. Foley
Professor of Chemical Engineering
Dean, College of Information Sciences and Technology

[1]Signatures on file in the Graduate School.

Abstract

Tables are ubiquitous with a history that pre-dates that of sentential text. Authors often report a summary of their most important findings using tabular structure in documents. For example, scientists widely use tables to present the latest experimental results or statistical data in a condensed fashion. Along with the explosive development of the digital library and internet, tables have become a valuable information source for information seeking and data analysis.

Interest in and use of table data necessitates table indexing and search. However, current search engines do not support table search. The difficulty of automatically extracting tables from un-tagged documents, the lack of a universal table metadata specification, and the limitation of the existing ranking schemes make the table search problem challenging. Effectively and efficiently searching table data becomes an urgent demand.

In this dissertation, we present an automatic table extraction and search engine, *TableSeer*. *TableSeer* crawls the web and digital libraries, detects tables from documents using heuristic-based and machine-learning based methods, represents tables using an extensive set of medium-independent table metadata that other people can reuse, indexes table metadata files, ranks tables, and provides a user-friendly search interface. To improve the performance of the table boundary detection, a novel page-box-cutting method and a sparse-line detection method are proposed. Given a keyword-based table search query, TableSeer ranks the matched tables and returns the most relevant tables

with a novel table ranking algorithm – TableRank. TableRank tailors the classic vector space model and adopts an innovative term weighting scheme by aggregating multiple features from three levels: the term, table and document levels.

Although tables are widely used, there is no standard on the table structure designing. Many issues that go into the design of tables and will impair the table data readability, accessibility, and reusability are ignored. In order to have a deep understanding on the table characterization and to improve the table extraction and search performance, we also implement the first large-scale table quantitative study on table natures in digital libraries.

We demonstrate the value of *TableSeer* with empirical studies on scientific documents. The experimental results show that our table search engine outperforms existing search engines on table search. Overall, TableSeer eliminates the burden of manually extracting table data from digital libraries and enables users to automatically examine tables. TableSeer is successfully deployed and in current use in several scientific digital libraries, for example $CiteSeer^x$.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Most prior work on information extraction has focused on extracting information from texts in digital documents. Often, however most important information being reported in an article is presented in tabular form. Tables are ubiquitous in scientific publications, web pages, financial reports, news papers, magazine articles, etc. More and more researchers in chemistry, biology, and finance fields adopt tables to display the latest experimental results as well as the statistical information in a concise layout. In the traditional media, tables have a history that pre-dates that of sentential text [40]. Tables present structural data and relational information in a two-dimensional format and in a condensed fashion. Researchers always use tables to concisely display their latest experimental results, to list the statistical data, and to show the financial activities of a business, etc. Other researchers, who are conducting the empirical studies in the same topic, can quickly obtain valuable insights by examining these tables. For example, a bio-chemist may want to search tables containing experimental results about "mutant genes" or an economist may look for the tables about "the GDP growth of USA in 2000-2007." In the table processing literatures, tables are usually analyzed from the following five perspectives: graphical, physical, structural, functional, and semantic [70]. The works in this dissertation mainly focus on the first four perspectives.

With the rise of the World Wide Web to the biggest pool of information, tables have gradually accumulated a significant amount of valuable information. Along with the rapid expansion of digital libraries, tables become an important data source for information retrieval. The demand for searching table is increasing. In order to unlock the information in tables, table search is a highly desirable feature for the Web search. If the data reported in tables can be extracted and stored in a database, the data can be queried and joined with other data using database management systems. However, due to the huge volume, seeking a relevant table is more difficult than finding a needle in the haystack. Unfortunately, current search engines are deficit in helping the end-users to find satisfactory table results. Existing search engines do not support table search. When applying a table search query to the popular search engines, we observe that a flood of unwanted and sometimes unsolicited results will be returned. Moreover, we also notice that within the top n returned results, the ranking order does not precisely reflect the relevance to the queries. The ranking schemes embedded in the existing search engines are inadequate and are not designed for table search.

Automatic table extraction is critical to the success of table search. However, current search engines do not support it. The difficulty of automatic table extraction, especially from un-tagged documents, makes automatic table extraction and table search problems challenging. Automatic table extraction is important because there exists no table markup language that scientists and other users have adopted for representing table information in documents. There is not yet a universal metadata specification for tables. Scientists and scholars who now want to extract data from tables in published papers have to do it manually, establishing there on metadata formats. In order to eliminate the

burden and enable users to automatically examine these tables, we design *TableSeer*, a system to automatically extract table metadata and search tables in digital libraries.

To get and share table data, we have to finish the following steps: 1) table boundary detection: identifying tables in documents; 2) table structure decomposition and table data extraction; 3) table data storage and representation for better accessibility and reusability; 4) Table analysis and understanding. The table structure/layout nature is very important for the performance of these steps.

Automatic table extraction and searching in digital libraries is a challenging problem for several reasons. First, although considerable research has been done to extract tables from HTML or Image documents, extracting un-tagged tables (e.g. in PDF format) in digital libraries is difficult. Limited attention has focused on this problem. Second, tables present unique challenges to information retrieval (IR) systems because of diverse media, different press layouts, cell types, affiliate table elements, etc. Along with the increasing demands of information sharing, an extensive and universal table metadata specification is needed to facilitate the table information processing, extracting, and reusing. Additionally, such a standard representation can heavily reduce the workload of table information extraction and increase the accuracy of the table indexing and searching. Third, the ranking schemes of current search engines are inadequate and not designed for table search.

The performance of a ranking scheme is also crucial to the success of a search engine. In the literature, most approaches primarily focus on the similarity of a query and a page, as well as the overall page quality using the vector space mechanism to calculate the relevant score and use the ordinary *TFIDF* [27] technique to determine the

weight of each term in the vector space. There are several other methods to calculate the similarity score between a document and the query. For example, the vector space model with pivoted document length normalization [5], the language modeling approach (LM) [71], the Okapi BM25 formula [77], etc. PageRank is another popularly-used ranking technique designed by Brin and Page [81]. With increasing popularity of search engines, many works focus on how to improve the rankings by tailoring the PageRank algorithm, e.g., incorporating implicit feedback (i.e., the actions users take when interacting with the search engine) [6], using historical data [25], overcoming the drawback of the random walk model [24], optimizing scoring functions and indexes for proximity search in type-annotated corpora [15], re-ranking using links induced by language models [53], using annotations in enterprise search [22], etc.

In this thesis, we also explore the issue about how tables should be ranked. We propose *TableRank*, a table ranking algorithm to facilitate our table search engine *TableSeer* to bring orders to the relevant tables. *TableRank* considers multiple features of a table and the document it appears in, and aggregates these features to determine the final ranking of the table with respect to a query. In order to evaluate the performance of *TableRank*, we compare *TableSeer* with several popular web search engines. Because the popular web search engines make no effort at treating tables specially, we propose two methods to set up the common test-beds. Our experiments on scientific documents show that *TableRank* can significantly improve the ordering of the top results in table search by incorporating features from different levels.

Our thesis has the following contributions: a table search engine *TableSeer*, a set of universal medium-independent metadata specification for tables, an automatic table

detector and table metadata extractor with a novel page box-cutting method, and an innovative table ranking algorithm *TableRank*, an improved table boundary detection method with the innovative sparse-line definition using machine learning concepts, the first large-scales table quantitative study for better table understanding, the first PDF table groundtruth dataset.

TableSeer crawls scientific documents from digital libraries, identifies documents with tables, extracts tables from documents, represents each table with a unique table metadata file, indexes tables and provides an interface to enable the end-users to search for tables. *TableRank* considers multiple features of a table and the document it appears in, and aggregates these features to determine the final ranking of each table with respect to an user query. Overall, the work in my thesis tries to achieve the following goal: detecting tables, extracting tables, enabling search function on tables, and providing a novel ranking algorithm to improve the displaying of the matched tables for a given search keyword. In this thesis, we focus on PDF documents for two reasons. First, PDF gains popularity in digital libraries due to the compatibility of output on a variety of devices. Second, PDF documents are overlooked in table extraction field. Please note that *TableSeer* can easily be extended to deal with documents in other media, e.g., PPT, HTML, Word, PDF, Image, etc.

Extensive experimental studies have been conducted to provide support for the table metadata extraction and the table ranking function. We evaluate the performance of our *TableRank* schema by comparing it with popular web search engines because no table search engine and table ranking algorithm exist. We propose two methods to set up

the common test-beds (see Section 4.4). Empirical results show that *TableSeer* achieves encouraging results compared to *Google Scholar*[1] and *CiteSeer*[2].

The remainder of the thesis proposal is organized as follows. Chapter 2 is the related work. Chapter 3, 4, 5, 6, 7 present five main proposal topics respectively: the architecture of *TableSeer*, the set of medium-independent table metadata, the table ranking algorithm *TableRank*, table boundary detection, and table structure decomposition. Chapter 8 shows the interface of the implemented TableSeer system. Chapter 9 elaborates the table quantitative study we implemented. The detailed parameter settings and the experimental results are discussed in Chapter 10. Chapter 11 presents an error analysis on the table metadata extraction algorithm. The final section lists all the references.

[1] http://www.google.com/
[2] http://citeseer.ist.psu.edu/

Chapter 2

Related work

2.1 Related Work on Search Engines

A search engine is an information retrieval system designed to help finding information [2]. Most commonly search engines are Web search engine, which searches for information on the public Web. For example, *Google*, *Yahoo! search*, *Microsoft MSN Search*, *ASK.com*, *Bing.com* etc. Other kinds of search engine are enterprize search engines, which search on intranets, personal search engines, and mobile search engines. Different selection and relevance criteria may apply in different environments, or for different uses. More recently, more and more lights are shed on specialty search engines. Some of them support search on various kinds of documents (e.g., map search [1], video search [2], image search [3] [4], patent search [5] etc.), as well as on document components (e.g., citation search [6], acknowledgement search [7], caption search [8] etc).

With the development of the internet and digital library, the accumulated digital documents become a valuable data repository . Table search has not gained enough

[1] http://maps.google.com/maps
[2] http://www.youtube.com/
[3] http://images.google.com/
[4] http://photo.net/gallery/caption-search/
[5] http://www.google.com/patents
[6] http://citeseerx.ist.psu.edu/
[7] http://citeseer.ist.psu.edu/
[8] http://biosearch.berkeley.edu/

attention in the information retrieval field. There is no specialized search engine and ranking algorithm to retrieve and rank tables in response to an user query.

Although table-related research has recently received considerable attention, most of the research focuses on the table extraction. Some researchers try to associate the table extraction with question answering (QA) or information retrieval. For example, Pyreddy and Croft [73] design a character alignment graph (CAG) to extract tables for information retrieval. TINTIN [73] is a system that incorporates indexing and searching concepts into the table detection field. However, it only roughly divided the information of a table into two parts: table captions or table entries. Hu [88] designs a system that extracts table-related information, stores them in databases, and generates a man-machine dialog to access the table data via a spoken language interface. None of above provide a real web search engine to tables. Recently, several online systems – Illustrara [80], BioText [63], and Telemarkus [9] – have provided either direct access to additional document components or incorporated these components into the information extraction and processing systems. Among these tools, only BioText provides the similar table search function as the TableSeer system provides. Although the performance of figure extraction and figure search in BioText system is very good, BioText can not handle tables very well. For example, BioText always displays a single table multiple times instead of returning all the tables in a documents because of unknown reasons. In addition, we checked all the published papers about BioText, unfortunately we can not find any information about table detection and table metadata extraction.

[9] http://www.telemakus.net/

To the best of our knowledge, *TableSeer* is the first table search engine, which not only supports the automatic table metadata extraction and table search, but also detects tables and extracts table metadata from PDF documents directly. Given a search query, the matched tables can be ordered by date, relevance, and citation.

2.2 Related Work on Table Definition

People are very familiar with tables. Everyone can design tables and read tables easily. Although there are numerous papers about the table processing, most of them start with the detailed analysis without providing the table definition. It seems that the authors assume the audiences holding the same concept on the tables in their papers. However, careful study reveals that the question "what constitutes a table?" is indeed difficult to answer [61].

Free online dictionary[10] defines a table as *an orderly arrangement of data, especially one in which the data are arranged in columns and rows in an essentially rectangular form.* The Oxford English Dictionary defines a table as "An arrangement of numbers, word or items of any kind, in a definite and compact form, so as to exhibit some set of facts or relations in a distinct and comprehensive way, for convenience of study, reference, or calculation." This definition summarizes the characteristics of a table using three different aspects: content, form and function. In Edward Tufte's book "Envisioning Information"[**?**], he refers to tables as a form of "small multiples", in which the same design structure is applied across a set of information. Because of the diverse

[10]http://www.thefreedictionary.com/table

table structure layouts and the people backgrounds, different people have different intuitive understanding of what a table is. For example, chemistry researchers will view the Periodic Table of the Elements as a table.

Among those papers that give out the definition, the definitions usually are far from general enough. The authors usually only define a type of tables or only emphasize the properties that are important in their specific domains. For example, the papers in the document image understanding area describe the table in terms of lines and non-analyzed text blocks [30]. In the lower level, a table is a set of cells located on a two-dimensional grid, or a structure laid out in an X-Y tree. In the higher level, a table is modeled as a set of categories (of strings) which are combined in some way to provide a mapping to a data domain. These definitions do not provide a theoretical basis from which to work.

Several researchers have provided definitions. Hurst defines a theory of tables in [40] that views the table as a presentation of a set of relations holding between organized hierarchical concepts (categories).

Long et al. define the tables in plain text document. In such documents, a table is a superstructure imposed on a character-level grid [60].

Peterman et al. [67] state that "table have a regular repetitive structure along one axis so that the data type is determined either by the horizontal or vertical indices".

Some papers do not separate the tables with other common representations for structure data (e.g., a form, a tabular structure, or a list). Some tables are similar to ruled forms. Many papers were published for form processing [94]. The main difference between the table processing and the form processing is that tables usually have an

unknown format and forms usually have a pre-defined template. The authors construct templates from empty forms and correlate with filled-in forms.

In HTML pages, people use the <table> tag not only for relational information display, but also to create multiple-column layout to facilitate easy viewing. Wang et al. define *genuine* table and *non-genuine* table to separate them from each other [92]. The *genuine* table refers to the document entities where a two dimensional grid is semantically significant in conveying the logical relations among the cells. In my thesis, all the tables we are processing are *genuine* tables.

In this dissertation, we focus on the tables in scientific documents. Comparing with the tables in the web, we call them as scientific tables and web tables. Figure 2.1 shows the examples. We add an additional restriction on the definition of tables in scientific documents: each table has a caption that starts with a keyword (e.g., "Table" or "TABLE"). With this restriction, the tables can be understood as genuine tables [96]. The table boundary includes all the parts in the Wang's model [89]. Both table caption and footnote are not included.

2.3 Related Work on Table Extraction

In the last 15 years, over two hundreds papers have been published on table recognition field [23]. Zanibbi [75] provides a survey with detailed description of existing approaches and systems. A detailed bibliography on table segmentation and extraction can be found here[11]. To extract the data from tables, we should fulfill two separate tasks: detecting the table boundary from the document and extracting the table cells as well

[11] http://www.visionbib.com/bibliography/char969.html

Shopping Categories

Baby
Apparel, Cribs, Strollers

Books, CDs & DVDs
Books, Music, Movies, Magazines

Clothing & Accessories
Men, Women, Shoes

Collectibles & Art
China, Figurines, Posters

Health & Beauty
Fragrances, Nutrition

Home & Garden
Bed & Bath, Garden, Kitchen

Jewelry & Watches
Rings, Watches, Necklaces

Mortgages
Refinance, Home Purchase, Loans

(a)

Table 1. EXAFS Single-Scattering Fits[a] of a (≡SiO)Cr(=O)$_2$ Model with Refined and Fixed Coordination Numbers against Data of Cr(VI) Xerogel Sample, Recorded at 15 K

	refined N[b]				fixed N[c]			
path	N	R/Å	σ^2/Å^2	ΔE_0/eV	N	R/Å	σ^2/Å^2	ΔE_0/eV
Cr−O	2.14	1.60	0.00293	−2.13	2	1.60	0.00456	−0.51
Cr−O	1.83	1.79	0.00742	−2.13	2	1.80	0.00785	−0.51

[a] Errors for first-shell scattering fits calculated against EXAFS data, in the absence of systematic fitting uncertainties, are generally to be accepted as follows: bond lengths ±0.02 Å, σ^2 ± 20%, E_0 ±20%.[42] [b] $S_0^2 = 0.84$, residual = 16.37 for this model optimization. [c] $S_0^2 = 0.83$, residual = 18.18 for this model optimization.

(b)

Fig. 2.1. A web table example (a) and a scientific table example (b)

as other table-related information. Researchers in the automatic table extraction field largely focus on analyzing the table in a specific document media. Most works focus on three main document types: *image-oriented*, *HTML-oriented*, and the *plain text-oriented*. Although many approaches and systems are available for these document types, there are few methods that are proposed to deal with PDF tables. Different media needs different table boundary detection and table decomposition methods. Although they are not new research topics, very few published papers provide algorithms to extract tables from PDF documents. Because of the numerous literatures and the similarity among some of them, we discuss those representative papers in the follow.

2.3.1 Table detection from image documents

Tables have been the focus of a large volume of research in the document image analysis field. Many approaches try to extract tables from image documents. They mainly focus on layout components (table boundaries, cell boundaries, etc.) to detect and decompose tables. Usually, they detect the potential table areas from the whole documents and isolate the table cells by analyzing the space information. Most research in this area focuses on analyzing the raster image. Some later works adopt the optical character recognition (OCR) technique to extract the text from the segmented parts. However, the performance of the OCR tool is not satisfying. The X-Y cut [43] algorithm is widely used by many papers in this field to cut the whole page into small pieces, then use bottom-up methods to group the pieces according to the pure geometric features. Most papers follow three basic steps: image pre-processing (e.g., the skew correction and

the noisy border removal), horizontal and vertical line detection, and table detection. The performance of the line detection is very crucial to the quality of the table detection.

Tupaj and colleagues [87] follow these steps and use OCR software to preserve the layout of tabular data by means of white space. They evaluate their method using two kinds of tables: technical tables and financial tables. The results show that they gain different results for these two kinds (the results on the scientific journals are much better than the results on the financial tables). However, their test size is too small (only 120 tables totally).

Goto et. al [28] is also a typical paper in this field. They focus on the detection of both vertical and horizontal lines as well as their intersection. No table structure decomposition is mentioned.

Shin and Guerette [47] proposed a table recognition algorithm based on Kieninger's work [51]. They use region growing to locate bounding boxes around text, and cluster them into columns by examining spatial relationships between bounding boxes and their vertical neighbors. Then, post-processing steps are needed to form tables. They only test the algorithm on 20 document images. The error types are large and the distribution frequencies of the error cases are high.

Mandal et al. [78] [62] rely on the spatial characteristics to separate tables and math-zones from image documents. Based on their observation, tables have distinct columns which imply that gaps between the fields are substantially larger than the gaps between the words in text lines. The main limitation of their algorithm is the simplicity.

In the scanned image documents, most researches detect tables based on some ruling lines. For example, Hirayama [33] uses ruling lines and the analysis of the presence

of characters as initial evidence of a table. However, many printed tables do not have such lines. Such method will miss many tables. There is a brief allusion in [34] to a system for semi-automatically identifying lines in tables as header, title, records, etc. No details of the algorithms are given. Other similar works are [55], [16], [42], etc.

Many of them believe that a key component of table structure recognition is column segmentation and the most common algorithms for table column segmentation without ruling lines are the vertical projection profile based method [76]. Cesarini et al. [14] label the table location in image documents where the presence of a table is hypothesized by searching parallel lines in the modified X-Y tree of the page. The problem with such vertical projection method is that when the columns are not perfectly aligned, or when the gaps between columns are slanted, the resulting histogram suffers from spurious peaks or flattened or completely smeared peaks, causing the results to be unreliable.

Zheng [103] proposed a frame line detection algorithm based on the Directional Single-Connected Chain (DSCC). Each extracted DSCC represents a line segment and multiple non-overlapped DSCCs are merged to compose a line based on rules.

Different from most systems that recognize tables by grouping layout information, Tersteegen [95] designs a system named SCANTAB to recognize tables using predefined information. However, such predefined domain information restricts the system usage. They only tested on scanned German business letters. Similar to our TableSeer, they predefined a table keyword dictionary to locate the potential tables quickly. An interesting idea of this paper is the graphical user interface that enables users to provide the table knowledge. These knowledge can facilitate the table detection on the same field. Some

predefined information is arbitrary, such as "table headers consist of a maximum of three text lines". This paper also contains a common shortcoming: the test bed is too small. They only trained on twenty-four documents and tested on fourty-four documents.

Shamilian et al. report the similar research in [83]. The read machine-printed documents in known predefined tabular-data layout styles (e.g., bills). In such documents, tables often do not rely on line-art to delimit fields. The main contributions of this paper contain the identifying and segmenting algorithm and graphical user interface (GUI) for defining new layouts. Until the published date, they collected more than 400 distinct tabular layouts. How to automatically verify the format of the incoming document is a big problem.

Zuyev [105] proposes an algorithm for table image segmentation. He introduces a concept of table grid that can assistant the further table structure analysis. A layer of terminal symbols for the table is provided. He tests the algorithm in FineReader OCR product. The author only publishes this paper and there is no any further research. Tsuruoka et al. [86] also describe a system of region segmentation and conversion into an HTML file for an unknown machine-printed table image. First, the system segments a table by means of the ruled lines into some regions. Secondly, these segmented regions are further segmented into cells by the omitted ruled lines that are indicators (such as numerals and characters). They convert a table of unknown complex structure into an HTML file.

2.3.2 Table detection from HTML documents

Detecting tables from HTML documents is not difficult. The tables rows and cells are delineated by the HTML markups such as $< TABLE >< /TABLE >$. Based on the tags, researchers usually convert the HTML table structure into a tree-shape, which the children of each node are the next lower level of table elements. Most HTML table detection rely on heuristic rules and their test bed is small. In addition, they are highly domain specific. For example, Chen et. al. [17] test their algorithm based on heuristic rules and cell similarities on 918 HTML tables from airline information web pages. Their F-measure is 86.50% .

Some researchers work on the classification of HTML tables because HTML tables are often used to format documents instead of presenting data. To decide whether a table is worthy of further process, we should combine language and layout information for a better content judgement. The drawback of the inconsistent HTML markups tags incurs the difficulty of the further identification on many table components, e.g., header lines.

Wang and Hu concentrate on the HTML table detection in [90] [92]. They classify HTML tables into two types: *genuine* tables and *non-genuine* tables, according to whether a two dimensional grid is semantically significant in conveying the logical relations among the cells. They introduce 16 features which reflect the layout as well as content characteristics of tables. These features are trained on thousands of examples, which are used to build a large table ground truth database. The experimental results show heavily improvement comparing with their previous rule-based system [26] (the

F-value is improved from 61.93% to 87.63%). No comparison is finished with other researchers' results. Both publications have a restriction that interesting tables can only be found in leaf table elements, which are $< TABLE >$ nodes that do not contain any other nested $< TABLE >$ nodes.

Yoshida et. al. [100] integrate HTML tables according to the object category. Their test data set contains 75 web pages and the F-measure is 88.05%. Shih and Karger [84] use table layout to improve the web-page classification task such as content recommendation and ad blocking.

Hurst talks about a Naive Bayes classifier algorithm but there is no detail about the algorithm and the experiment [39]. They firstly mention [20] the idea of rendering HTML code in order to use the results for detecting relational information in web tables. Although they do not actually render documents in a browser, they can infer relative positional information of table nodes in an abstract table model.

Different from the traditional method, Kuprl et. al. [9] use the visual rendition from the Open Source Mozilla browser rather than the HTML tags to detect the web tables. Their bottom-up clustering algorithm [52] groups visualized words into large groups and uses a set of heuristics to detect tables unsupervised. The reported performance is about 70%. In addition, there are several limitations: only those table types that are known in advance are supported by their algorithm. If there is a vertical gap in table cells, the algorithm currently does not deliver correct results. Also, the process is relatively slow, because it has to wait for Mozilla to render a page.

Gatterbauer and Bohunsky [29] provide an innovative approach to deal with the table extraction problem from web pages by analyzing CSS2 visual box model. CSS

represents textual HTML document elements by rectangular boxes, and a double topographical grid is adopted to manage all these visual boxes. A recursive spatial reasoning algorithm named VENTrec is proposed. VENTrec uses keywords as input, and tries to expand from any possible HyperBox on the grid into all directions until no possible expansion. The authors rely upon the positional and metadata information of visualized DOM element nodes in a browser. They believe that the individual steps of table understanding become easier by taking advantage of visual cues of the web.

WebTable [12] extracts tables from a web page file. The resulting tables can be imported into a spreadsheet or database program.

2.3.3 Table detection from plain text documents

Comparing with the HTML tables and image tables, there are few papers on the table detection from plain text documents. These documents may generated in ASCII form directly, or converted from a richer documents (e.g., save a HTML page into the pure text). Usually, there is no ruling lines in such documents. We can only analyze the cell contents based on the 2-D layout information. In plain text, writers often use special symbols, e.g., tabs, blanks, dashes, etc., to make tables.

Hu et. al [45] propose a dynamic programming table recognition algorithm to detect tables based on computing an optimal partitioning of a document into some number of tables. Because it is American Standard Code for Information Interchange (ASCII) text based, it cannot fully make use of document image information. They also propose a table structure recognition algorithm in [46]. Initially, hierarchical clustering

[12] http://windowsutilities.com/webtable.html

was used to identify columns. Then spatial and lexical criteria were used to classify headers. All of the existing algorithms were evaluated on their in-house data set. Its assumption of using single text column as input is relatively restrictive.

Kieninger [51] [85] proposed a bottom-up table recognition system, T-Rec System, where vertically overlapping words are grouping into blocks to identify tabular structure. He parses tables in ASCII or paper documents that do not rely on ruling lines. This method relies on complex heuristics which were based on local analysis to split or merge the blocks into proposed columns. However, he starts from the detected table regions and focus on the table structure decomposition, without considering the table detection. Because of its pure bottom-up nature and the fact that the heuristics are based on very local analysis as well, the method suffers when there is disruption to the normal structure of the columns (e.g., the present of a comment line spanning multiple columns, or two cells from different columns accidentally overlapping with each other), or when there is a gap within a column. The author has discarded this topic and the code package died for years.

2.3.4 Table detection from PDF documents

Comparing with our TableSeer, the most related work is pdf2table [72], which also extracts tables from PDF files. Instead of the tools PDF2TET [13], PDFBOX[14], and Text Extraction Toolkit (TET)[15] we tried, they use the pdf2html[16] developed by Gueorgui Ovtcharov and Rainer Dorsch to get all text elements (i.e., strings) with their absolute

[13]http://www.pdf2txt.com/
[14]http://www.pdfbox.org/
[15]http://www.pdflib.com/products/tet/
[16]http://pdftohtml.sourceforge.net

coordinates in the original file. All the returned texts are saved in a structured data format (XML file). Comparing with our methods, they have similar preprocessing steps: sorting texts by the coordinates to avoid the order uncertainty brought by the document creator (we detect the table boundary first, then only sort the small area), merging text elements into lines, and combing lines into blocks (in our thesis, we define them as *boxes*). Although they also use heuristic-based approach to detect and decompose tables, their methods are very limited because of the following reasons. First, they only process the single-column PDF documents and the performance is only good for the lucid tables. However, most scientific documents contain more than one column. Second, the adopted heuristics are also rough. Each of them will incur a lot of noisy results. For example, the definition of the multi-line will treat a lot of actual single lines as the multi-lines. Third, to merge the multi-line block, they use five self-defined threshold values. Based on our experience, these thresholds are very important to the final performance and it is impossible to deal with all the tables with a single fixed threshold value. However, they do not provide any explanation on the value setting. Fourth, treating each multi-line block as a table and allowing up to five single-line objects within a table will incur many false positive tables and table lines. Fifty, according to their table decomposition method, the detected table column number is usually larger than the actual value. Sixty, they do not work on the vertical tables. Seventy, only the table itself is processed without other important table-related data, such as the table footnote and the table-referenced text. In addition, the accuracy of pdf2html is also not good enough. We processed their paper [72] with their released package, both precision and recall value are less then 50%. To remedy the deficiency of the tool, the authors implemented a graphical user interface,

which enables the user to make adjustments on the extracted data. There is no further work on this topic after 2005.

Wasserman et al. [93] design an algorithm to detect potential table regions in the PDF document, and to analyze the table structure. First, they convert 100 scientific documents in PDF format into TIFF image format, then treat a set of word boxes as the input of the algorithm and use the word segmentation technique in [91]. They yield a set of potential table-region bounding boxes through the processing of a set of modules: word-box consolidation and text-line elimination module, vertical-range module, horizontal-range module, and table-region extraction module. The main shortcoming of this paper is the lack of the details. Although they realize the importance of the precise definition of table, there is no definition for all the terms. In addition, there is no experiment and evaluation.

Tamir Hassan and Robert Baumgratner [32] also use PDFBox to extract texts from PDF files and try to recognize tables. AIDAS system, which extracts the logical document structure from PDF documents [7], uses shallow grammars to detect the logical elements as the authors admitted, and no result and evaluation is presented.

2.3.5 The methodologies used in the table detection field

From the methodology perspective, all the methods can be divided into three categories: pre-defined layout based [49], heuristics based [36][48][51][85], and statistical based [98]. Pre-defined layout based algorithms usually work well for one domain, but is difficult to extend because of its reliance on hand-crafted rules. Heuristics based methods need a complex post-processing and the performance relies largely on

the choice of features and the quality of training data. Most approaches described so far utilize purely geometric features (e.g. pixel distribution, line-art, white streams) to determine the logical structure of the table, and different document mediums require different process methodologies: OCR [21], X-Y cut [43], tag classification and keyword searching [9][18][96] etc. Most of them use trial-and-error methods and no general table ground-truth data set is publicly available to train and test these algorithms [97].

[50] is a typical work based on predefined table layout structure while [51] is a typical work relying on complex heuristics that were based on local analysis. Almost all the table-structure-extraction algorithms were developed using trial-and-error methods. No general table ground truth data set is publicly available to train and test these algorithms [97].

Pinto and colleagues [69] think that the traditional language modeling techniques are not good enough to deal with the rich combination of formatting and content present in tables. They presented a model that utilizes both content and layout of tables by using conditional random fields (CRFs), and compares them with hidden Markov models (HMMs). CRFs are undirected graph models that can be used to calculate conditional probability of values on designated input nodes. At the beginning, they label each line of a document with a tag, such as non-extraction label, header label, data row label, caption label. They also describe features [68] to identify a narrower range of line types: white space features, text features, separator features. A threshold is set for each percentage feature. After a training phase, they test on documents from a crawl of www.FedStats.gov performed in June 2001. They focus on the text tables in the web pages (text tables are human formatted using white space and fixed font to

align columns). The experimental results show that tables are located with 92% F1, and their constituent lines are classified into 12 table-related categories with 94% accuracy. However, the current state of their model can only locate labels and tag lines but knows nothing about the columns and cannot distinguish between data cells and header cells. Moreover, their work starts from the detected table areas without elaborating how to detect the table boundary. Even they can label a line as the table line, if there are several continuous tables, it is impossible to separate them from each other.

Ng, et al. [36] proposed a machine learning based method for both table detection and column/row segmentation. They designed purely text features and trained classifiers on Wall Street Journal news articles. The performance of these classifiers relies largely on the choice of features and the quality of training data. The authors selected the features based on typical table layouts in their test files. It is not clear whether the proposed features can generalize to documents in other domains.

Hu, et al., proposed a hierarchical clustering to recognizing the structure of a detected table region by identifying columns and spatial and lexical criteria to classify headers [44]. They also experimented with Wall Street Journal (WSJ) pages. Wang formulated the table structure-understanding problem in the whole page segmentation framework using optimization methods [98]. No single algorithm works well on all types of documents.

For a table search query, the matched tables may be contained in diverse media, which require different methods to extract. It is inconvenient to divide the documents into different groups and apply the corresponding algorithms. Thus, good table representation schemes are needed.

2.4 Related Work on Table Representation

The well-known table representation schemes are designed by the World Wide Web Consortium (W3C) in the specification of XHTML (The Extensible HyperText Markup Language) and by the Organization for the Advancement of Structured Information Standards (OASIS). However, users have to have a long learning curve and may learn many useless attributes for their purposes.

Wang [89] proposed a table model to show the table layout. The model consists two components: a finite set of labeled domains (or categories), and a mapping from the tree paths labels to the possible values. This model is widely adopted by many researchers. As Wang admitted himself, the greatest shortcoming of the model is the neglect of the external table information, such as the table footnote. Embley et al. [23] illustrate the transition from a single table recognition method to a system with some web table examples. They show that the Wang model is adequate for some useful tasks. Another valuable point of this paper is the proposed components that should be included in any table ontology.

Akira Amano [4] proposes a representation of table form document based on XML. Wang [97] [98] develops a software tool to generate documents that include similar table elements based on the given table ground truth.

Matthew Hurst et. al. [37] design a model to break tabular text into blocks, then determines what the blocks representation using generative language models in both stages. His paper [40] provides a framework for representing tables at both the semantic and structural levels. He views the table as a presentation of a set of relations

holding between organized hierarchical concepts (*categories*). In order to provide a conceptualization couched in basic terms that will assist research into table processing system, he tries to indicate the framework to represent categories, the interaction between the syntactic structure of the string content of the table, the logic structures they present, and the structure of the table. The author wants to use the theory to classify tables into classes, to evaluate table processing methods, to understand the distribution of table classes, etc.

However, all of them seldom cover table structure and layout information, as well as the table-related information and the document background. Our table metadata tries to provide a universal metadata specification to represent the information and all the necessary facts of a table in any document medium for the table searching purpose. Comparing with the above representation schemes, *TableSeer* also reduces the extraneous metadata to improve the retrieval performance.

The Dublin Core is a set of metadata elements to describe the resources in the Internet. It is proposed by Online Computer Library Center (OCLC) [17] and National Center for Supercomputing Applications (NCSA) [18] on March 1995. The goal of DC is to provide standardization for cross-domain information resource description. With the Dublin core metadata, researchers aim to describe online resources in ways that are much easier to understand and search. The semantics of Dublin Core were established and are maintained by an organization - The Dublin Core Metadata Initiative (DCMI) [19]. The advantages of the Dublin Core include: easy to create, easy to understand,

[17] http://www.oclc.org/
[18] http://www.ncsa.uiuc.edu/
[19] http://dublincore.org/

across sections and subject domains, low cost, universally understandable descriptions common across disciplines, etc. The simple Dublin Core standard comprises only fifteen elements to describe many types of digital materials such as video, sound, image, text, and composite media such as web pages. To the best of our knowledge, no researchers incorporate the table data with the Dublin Core, which is our ongoing task.

2.5 Related Work on Table Ranking

Ranking search results is a fundamental problem in information retrieval. Most common approaches primarily focus on the similarity of a query and a page, as well as the overall page quality. There are several methods to calculate the similarity score between a document and the query. For example, the vector space model with pivoted document length normalization [5], the language modeling approach (LM) [71], the Okapi BM25 formula[77], etc. PageRank is a popularly-used ranking technique designed by Brin and Page [81]. With increasing popularity of search engines, many works focus on how to improve the rankings by tailoring the PageRank algorithm, e.g., incorporating implicit feedback (i.e., the actions users take when interacting with the search engine) [6], using historical data [25], overcoming the drawback of the random walk model [24], optimizing scoring functions and indexes for proximity search in type-annotated corpora [15], re-ranking using links induced by language models [53], Using annotations in enterprise search [22], etc. Most of these ranking approaches use the vector space mechanism to calculate the relevant score and use the ordinary *TFIDF* [27] technique to determine the weight of each term in the vector space.

More recently, many researchers shed light on designing novel ranking algorithms to satisfy their specific search purposes, for example, XRANK [31] to rank XML elements using the link structure of the database, ObjectRank [41] for authority-based search on databases, and PopRank [66], a domain-independent object-level link analysis model to rank the objects within a specific domain instead of web pages, SemRank [8] to rank complex relationship search results on the semantic web, etc.

Unfortunately, to search tables, all the above will introduce a high false positive rate and unwanted results in which the query keywords appear in other areas instead of within or related to the tables. After checking the results returned by these popular search engines, we find that current search engines as well as their ranking algorithms are very limited in dealing with the table searching problem. They introduce a high false positive rate and unwanted results in which the query keywords appear in other document sections instead of within or related to the tables. There are two main reasons. First, current ranking algorithm treat all the terms in a document equally without the capability of recognizing which terms are related to tables. Second, they overlook several important factors (e.g., the term location, the reference text of a table, the document quality, etc), which play important roles to determine the ranking score of each table.

We build on existing table search engine – *TableSeer* [56] – to develop a novel table ranking algorithm for the real web search. Instead of treating each document as a whole unit, we automatically extract tables and calculate their relevant scores by aggregating impact features from multiple levels. In order to detect tables in un-tagged documents (e.g., Portable Document Format (PDF)), We design a novel page box-cutting method

[56]. To the best of our knowledge, we are the first to do a comprehensive study on the research of table search and table ranking.

2.6 Related works on table analysis with machine learning approaches

Several machine learning approaches are applied in the table analysis field, e.g., decision tree [79], Naive Bayes classifier [104], Support Vector Machine (SVM) [12], Conditional random fields (CRF) [54] etc. Hurst mentioned in [38] that a Naive Bayes classifier algorithm produced adequate results but no detailed algorithm and experimental information was provided. Wang et. al. tried both the decision tree classifier and SVM to classify each given table entity as either *genuine* or *non-genuine* table based on features from layout, content type, and word group perspectives. Decision tree learning is one of the most widely used and practical methods for inductive inference. It is a method for approximating discrete-valued functions that is robust to noisy data. Comparing with our work, they started with the detected tables and all features are related to the table itself (e.g., the number of columns). How to detect these tables, the key problem of our paper, is missing in their work.

Conditional random fields (CRFs) is initially introduced by Lafferty et al. [54] in 2001 as a framework for building probabilistic models to segment and label sequence data. Afterwards, CRF is applied in bio-informatics, computational linguistics and speech recognition fields. Conditional Random Fields (CRF) have been shown to be useful in part-of-speech tagging [54], shallow parsing [82], named entity recognition for newswire data [64], as well as table detection [21]. To the best of our knowledge, Pinto et al. [21] did the most related work as we did. Comparing with our work, the difference

spans the following areas: 1) they extract table from a more specific document type – plain-text government statistical reports; 2) because of the specific document nature, they adopted several special labels and corresponding features (e.g., BLANKLINE label and SEPARATOR features), which are not applicable for other document types; 3) their features focus on white space, text, and separator instead of the coordinate features, which are important for the table structure decomposition; 4) although they claimed that their paper concentrated on locating the table and identifying the row positions and types, they did provide the detail about the table locating. In order to improve the performance table data extraction, we zoom in the table boundary detection problem and elaborate the feature selection in our paper. Moreover, we consider the coordinate features, which not only play a crucial role in the table boundary field, but also are unavoidable in the later cell segmentation phase. Different from most CRF applications, the input data is a document line instead of a word.

2.7 Related Works on Table Classification

Although there are more than two hundred of published papers on table analysis, almost all of them focus on how to locate table boundary and/or how to de-composite table structures in documents. Only a few researchers analyze tables further after the extraction [96] [35] [19] [1] [70] [102]. In order to have a better table understanding and usage, a decent table classification from different perspectives is needed. However, some do not state a clear definition for various tables and most of them only classify HTML tables according to the usage of table element in HTML files:

Wang and Hu [96] classified each given HTML table entity as either genuine or non-genuine. According to their definition, genuine tables are document entities where a two dimensional grid is semantically significant in conveying the logical relations among the cells. Conversely, non-genuine tables are "document entities where $< TABLE >$ tags are used as a mechanism for grouping contents into clusters for easy viewing only". Different from their work, we focus on genuine tables in this paper because almost all the tables in the scientific repository are genuine tables.

Similarly, Wang et. al. [19] also classify HTML tables into two categories: *data tables* and *layout tables*. In data tables, the content in cells is clearly correlated. oppositely, layout tables are only considered to be tables by virtue of their appearance once rendered by a suitable browser.

Hurst [1] distinguished two types of tables in HTML documents. type 1 has some clear relationship with an instance of the TABLE tag in the HTML document; type 2 tables are only considered to be tables by virtue of their appearance once rendered by a suitable browser.

Pivk et. al. [70] identified three major table classes according to the layout: 1-Dimensional (1D), 2-Dimensional (2D), and Complex tables. According to their definition, 1-Dimensional table has at least one row of A(ttribute)-cells above the rows of I(nstance)-cells. 2-Dimensional table has a rectangular area of I(nstance)-cells appearing within columns. Complex table might have the following features: partition data labels, over-expanded labels, and combination.

Yoshida et. al. [102] categories tables into nine types according to the table structures and the relative locations of attribute string and value strings: type 0, type

1-h, type 1-v, type 2-h, type 3-h, type 4-h, type 2-v, type 3-v, and type 4-v. Please refer the original papers for the details.

Chapter 3

Topic I: The architecture of the TableSeer

3.1 System Overview

Figure 3.1 shows the architecture of *TableSeer*, which consists of a number of important components: 1) a table crawler, 2) a table metadata extractor, 3) a table metadata indexer, 4) a table ranking algorithm, and 5) a table searching query interface. In summary, *TableSeer* crawls scientific documents from the digital libraries, identifies the documents with tables, detects each table using a novel document *page box-cutting* method, extracts the metadata for each identified table, ranks the matched tables against the end-user's *query* with the *TableRank* algorithm, and displays the ordered results in a user-friendly interface.

3.2 The Table Crawler

TableSeer harvests online scientific documents by crawling open-access digital libraries and scientists' webpages. The crawler supports a number of document media, such as PDF, HTML, WORD, PowerPoint, etc. *TableSeer* is designed to be able to handle all the document media listed above. In this paper, the table crawlers pay attention to the scientific documents in the PDF format because they gain more and more popularity in digital libraries. In addition, comparing to other document media that have been extensively studied in the table detection field, PDF documents have been

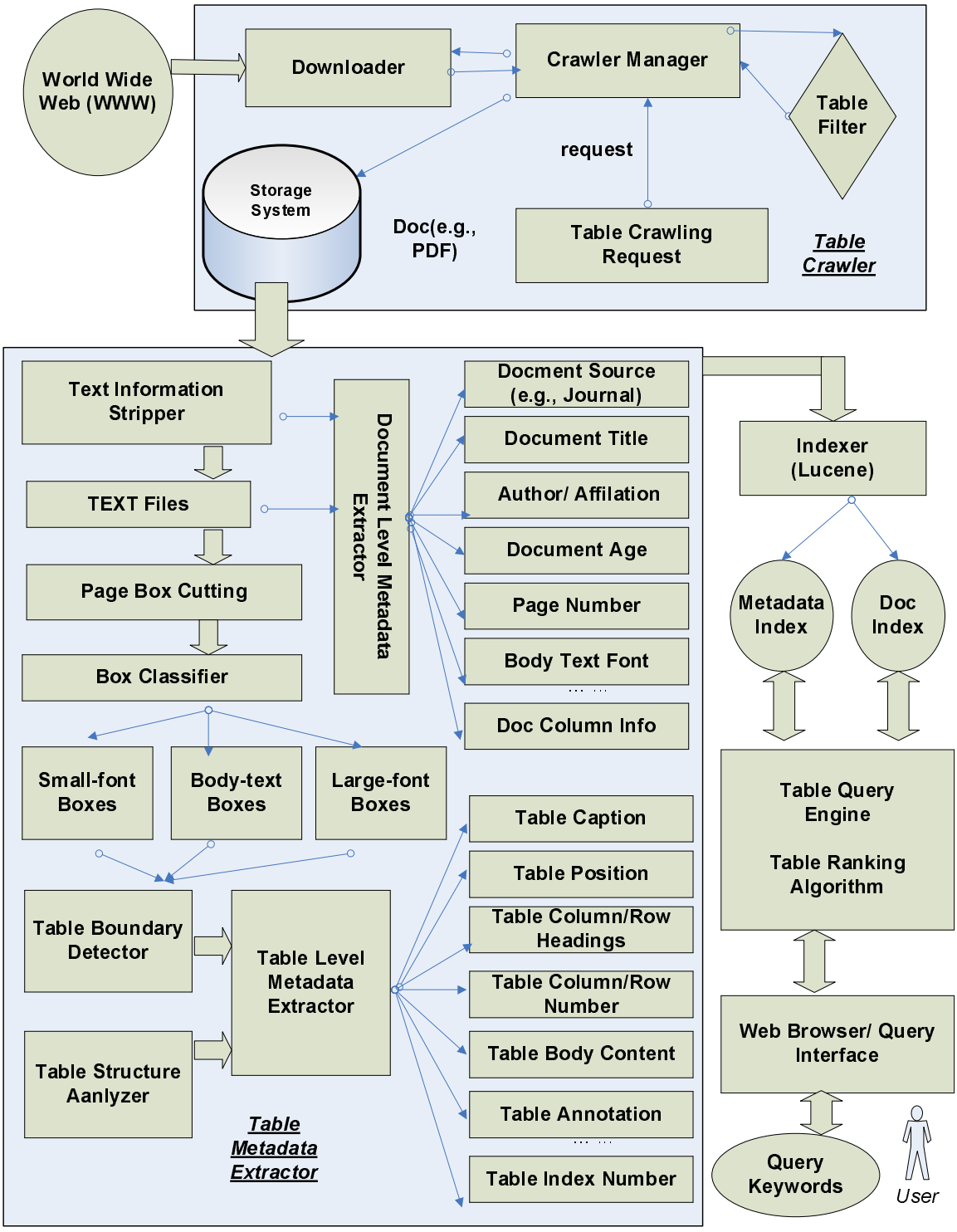


Fig. 3.1. The Architecture of TableSeer

overlooked. Our table crawlers use a depth-first crawling policy with a maximum depth of five (from the seed URLs).

3.3 The Table Metadata Extractor

The table metadata extractor comprises of three key parts: 1) a text information stripper (*TIS*), 2) a table boundary detector with the *page box-cutting* method, and 3) a table metadata extractor from both the document level and the table level. The details for the second and third parts can be found in Chapter 6 and 7.

Initially, for each PDF document, *TIS* strips out the text information from the original PDF source file word by word through analyzing the *text operators* and the related glyph information[1]. *TableSeer* reconstructs these words into lines with the aid of their position information and saves the lines into a *Document Content File* in the *TXT* format. For each document page, *TableSeer* analyzes the text information and merges them into different physical component levels (lines, paragraphs, boxes, pages) according to their font and position information. Figure 3.2 shows an example segment of a PDF document and its corresponding *Document Content File*. The text file also records the coordinates of the left-most word (X_0, Y_0), the line width W, the line height H, the font size F, as well as the text content of each line. All the lines are ordered in the same sequence as shown in the PDF document.

With the extracted table metadata, we can manipulate the table data freely.

[1]PDF Reference Fifth Edition, Version 1.6

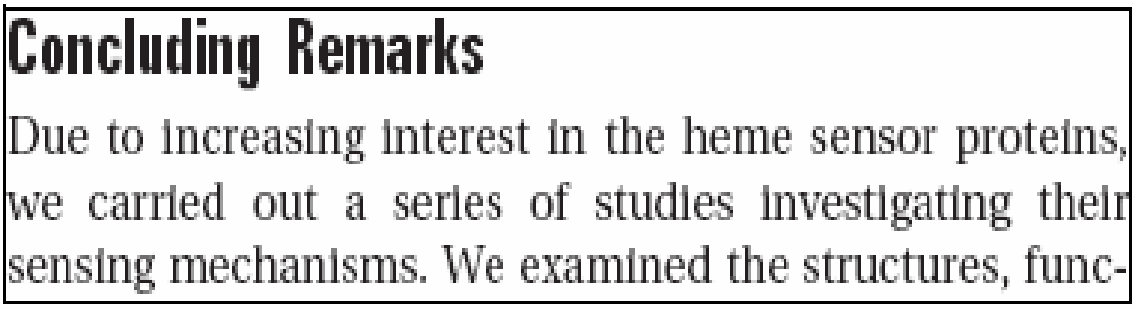


Fig. 3.2. An Example of a *PDF* Document Segment and the Corresponding *Document Content File*

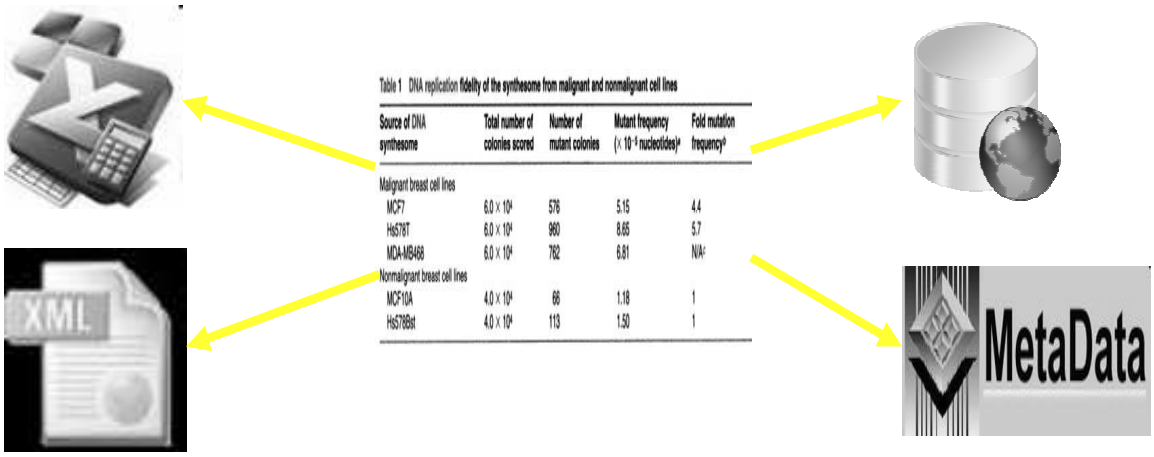


Fig. 3.3. Table Data Storage and Sharing ways

Chapter 4

Topic II: Table Metadata

Tables present a unique challenge to an information extraction system. Extracting sharable table metadata from documents no matter what document formats they are in is a challenging problem for several reasons: 1) Tables are shown in documents with diverse medium types, like HTML, PDF, etc; Different media use different schemes to save table information and present tables in different layouts; 2) Different press categories (e.g. journals or conference proceedings, scientific papers as well as financial reports) may have their own requirements for the table layout. For example, different from computer science fields (e.g. JCDL proceedings), most chemistry/biology journals only require vertical lines to separate table with paper context. Within the table body, some journals display rows one by one without any separators while some others alternatively change the background color for a better visual effect; 3) Table cells are so diverse that different content elements can be stored in one cell; 4) Some tables have affiliated elements; 5) No formal rules/standards on table designing; 6) Different applications and purposes have different requirements on the table metadata extraction;

Regarding the physical layout, the simplest tables are regular matrices of cells: all the cells of a line have the same height and all the cells of a column have the same width. The borders of all the cells are marked by graphic lines. However, very few tables follow such a strict prescription. Many tables can be much more complex. We often

notice some over-expanded tables that the presence of cells extend over several rows or columns, or only a few of the borders are marked by graphic lines. Not all the cells are filled and some partitioned tables can be treated as a combination of several tables as a multi-dimensional structure.

What metadata can be summarized from tables no matter what media they are located in? Which metadata is essential if we only want to search the contents of a table? If we want to exchange a table with others who have not seen it, which metadata are required to explicitly describe it, with/without layout information? Can a table reconstructed to their original visualization layout and structures as well as all the cells with the original contents if we get its metadata? If we create a database to enable users to search tables in a digital library, which metadata are indispensable? Given a table in a document in any medium, can the table metadata be extracted automatically? In other words, our research purposes are generating table metadata that can be used to facilitate the table indexing, searching, and exchanging, and designing an automatic metadata extractor on electronic documents. The metadata not only include the information related to the table itself, but also the accessorial information ignored by current researchers, such as table footnote, document background, referenced annotation etc. We propose a set of medium-independent table metadata and building up the metadata foundation for table representation and description, table searching and exchanging. Some characteristics of our table metadata extraction are: 1) The table metadata should have meaningful names. 2) They should be easily stored for indexing and searching. 3) They can be combined differently according to the users' purposes. For example, only sharing the text or displaying the original layout.

In order to be able to characterize tables occurring in very diverse composite documents, we have designed a rich and flexible representation scheme for table metadata that describes tables in digital documents. We classify the table metadata into six mutually exclusive categories: 1) table environment/geography (Document-level), 2) table-frame metadata, 3) table affiliated metadata, 4) table-layout metadata, 5) table cell-content metadata, 6) and table-type metadata. Figure 4.1 shows a segment of a table metadata file.

Table Environment/Geography Metadata. The *table environment/geography metadata* includes the information of the document where a table is located, such as *Document Medium Type* (HTML, PDF, image, PS, text, email, etc), *Document Page Number* of the table (the index number of the page where the table is located), *Document Title* (the paper title shown in the journal or conference proceedings, usually in a large font size), *Document Author*, *Document Origination* (the name of the journal or conference), *Document Age*, *Table Starting Position* (the X and Y-axis coordinates of the starting place of the table), etc. This metadata can facilitate the table searching if users only know pieces of the document information or wish to restrict the search to certain types of documents.

Table-Frame Metadata. The Table Frame metadata records whether there are frames in the four sides around a table. The values can be left, right, top, bottom, all, none, top and bottom, left and right.

Table Affiliated Metadata. A table has several affiliated elements. *Table Caption* is the caption sentence(s) that appears along with the table, e.g., *"Table 1. Molecular Properties of Tested Polymers"*. *Table Caption Position* is the position of the

caption with relation to the body of the table: above or below. *Table Footnote* is text that explains the information in the table and usually appears below the table body. *Table Reference Text* is the text in the document body that refers to the table and discusses the content of the table.

Table Layout Metadata helps to capture the visualization of the original table. It is composed of *Table Width* (the width of the table boundary), *Table Length* (the length of the table boundary), *Number of Columns*, *Number of Rows*, *Stub Separator* (vertical ruling), *Boxhead Separator* (horizontal ruling), and *Column Width*, *Row Length*, *Column Headers*, *Row Headers*, and *Horizontal Alignment* (the values can be left, right, and central).

Table Cell Content Data refers to the values in each cell of a table, and enables people to search tables based on the contents of their cells. Content in *Cell[i, j]* is the content in the cell that is located in the i^{th} row and the j^{th} column of a table.

Table Cell Type Metadata records the type of a table based on the type of its cells: *Numerical* and/or *Symbolic*. If the table contains cells with numeric information, it gets a type *Numerical*, if it has symbols, like, text, equations, etc., then it is marked *Symbolic*, Numerical tables can also be further divided into number tables, mathematical equation tables, percentage tables, and so on. Symbolical tables include character tables, image tables, formula tables, and so on. A table can have both numerical and symbolic cells.

```
<table-metadata>
<property>
  <name>Paper Title</name>
  <value>Dissolution of albite glass and crystal</value>
</property>

<property>
  <name>Table Caption</name>
  <value>Table 2. Comparison of crystalline and amorphous albite dissolution rates</value>
</property>

<property>
  <name>Table Column Head</name>
  <value>Type of experiment Initial pH Final pH Temperature</value>
  <description>……</description>
</property>
          ……
</table-metadata>
```

Fig. 4.1. An Example of the Table Metadata File

Chapter 5

Topic III:
The Table Ranking Algorithm – TableRank

One key question in information retrieval is how to rank matched results based on their relevance to a query. However, The existing ranking schemes are inadequate and are not designed for table search. Our *TableSeer* search engine has an innovative table ranking algorithm – *TableRank*. Given a user query, *TableRank* returns the matched tables in a descendant order according to their relevance scores. Different from the popular web search engines, our *TableRank* rates the <query, table> pairs instead of the <query, document> pairs.

TableRank tailors the traditional vector space model to rate the <query, table> pair by replacing the document vectors with the table vectors. As shown in Table 1, each row is a query or a table vector. To determine the weight for each term in the vector space, we design an innovative term weighting scheme: Table Term Frequency - Inverse Table Term Frequency (TTF-ITTF), a tailored *TF-IDF*[27] weighting scheme. Compared with *TF-IDF, TTF-ITTF* demonstrates two major advantages. First, it calculates the term frequency in the table metadata file instead of the whole document, which prevents the false positive results. Second, it calculates the weight of a term with a comprehensive view.

We divide the features we consider into two groups: *query-dependent* features and *query-independent* features. Query-dependent features include the traditional features

of a document (e.g., the document title, the author/affiliation, the abstract, etc) and the structural features of a table (e.g., the table caption, the table column header, etc). Query-independent features include the document citation, the document freshness, and the document origination, etc. We consider the query-independent features because when several similar tables match the end-user's query, intuitively, the end-user is most interested in the tables in newly published articles (document freshness) with the high quality, such as a top conference (document origination) or with a large number of citations (document citation). They are the tables that the user probably has not seen before and is seeking. Overall, *TableRank* considers features at three levels: 1) the term level, 2) the table level, and 3) the document level to determine the final ranking score of a table.

TableRank algorithm first applies each impact feature to weight the terms in the vector space, then aggregates all these features to determine the final weight values. Cosine measure is used to determine the similarity between the query vectors and the table vectors.

5.1 Problem Statement

We formalize the notations of the table ranking problem used throughout this paper. We represent the set of crawled and extracted tables as $T = \{\cup\ tb_j, j \in [1, b]\}$, where b is the total number of tables in a set of documents, $D = \{\cup d_\alpha, \alpha \in [1, N]\}$, where N is the total number of documents. $F(T, D)$ is the mapping function from $T \rightarrow D$. $\forall tb_j \in T, \exists d_\alpha \in D$ where the table tb_j comes from the document d_α. Also, $\forall tb_j \in T$, $\exists$

$TM_j = \{\cup\, m_k, k \in [1, d]\}$, where, m_k is the metadata proposed in [59] to represent table tb_j, and d is the number of metadata used to describe each table.

A term t_i may repeatedly appear in different table metadata m_k (e.g., table caption, table column/row header, table reference text, even the document title). In particular, terms reflecting the core idea usually appear more frequently and broadly. We believe that different m_k have different influences on the results and each m_k has an assigned weight MW_k (the details of weight setting is discussed in Section xxxx). The same terms appearing in different m_k should be treated separately. Likewise, the same terms appearing in the same m_k should be treated equally.

Let $sim(tb_j, Q)$ denote the similarity between the table tb_j and the query Q. Both of the table and the query can be represented as a vector. Each vector is composed of a set of alphabetically ordered terms. All the table vectors and query vectors construct a vector matrix as shown in *Table 5.1.* Each row in the matrix represents the vector of a table tb_j ($j \in [1, b]$) or the query Q. Each table tb_j has k metadata elements describing it. The size s of the vector matrix is defined as the total number of terms appearing in the table collection T and the given query Q. For example, in *Table 5.1*, $s = (x + y + ... + z)$.

$w_{i,j,k}$ is the term weight of the i^{th} term in the k^{th} metadata of the table tb_j and $w_{i,q,k}$ refers the term weight of the i^{th} query term in the k^{th} metadata. If the i^{th} term does not occur in the metadata m_k of the table $tb_j, w_{i,j,k} = 0$. Otherwise, $w_{i,j,k} > 0$. The term weight $w_{i,q,k}$ follows the same rule. As discussed, a term t_i may repeatedly appear in multiple metadata m_i (e.g., table caption, table column/row header, table reference text, etc). The importance of different table metadata varies. We assign each metadata m_i a metadata weight MW_i. For example, the metadata "*Table Column Header*" should

get more credits than the metadata "*Table Footnote*." The details can be found in section "*Parameter Settings*". In addition, if the user specifies the query term location, e.g. find the tables with term "*gene*" in the metadata *"Table Caption"*, the term "*gene*" should only be filled in the metadata *"Table Caption"* columns in *Table 5.1*, and the MW of "*Table Caption*" should be set higher. Otherwise, the term "*gene*" should also be filled in the metadata columns where the term occurs.

5.2 Similarity Measurement with a Tailored Vector Space Model

TableRank constructs the vector matrix and measures the similarity between the query and each table by computing the cosine of the angle between these two vectors (the equation is shown in Equation 5.1).

$$sim(tb_j, Q) = cos(tb_j, Q) = \frac{\sum_{i=1}^{s} w_{i,j,k} w_{i,q,k}}{|tb_j||Q|} \tag{5.1}$$

where, the final weight $w_{i,j,k}$ of the i^{th} term in the k^{th} metadata of the table tb_j is computed by Equation 5.2. The weighting scheme is also applicable to the query term $w_{i,q,k}$.

Most existing text retrieval techniques rely on indexing keywords. Unfortunately, keywords or index terms alone cannot adequately capture the document contents, resulting in poor retrieval performance. In *TableSeer*, we eliminates this risk by indexing table metadata files, which concentrates the table-related information into a small-size file. *TableSeer* uses the Lucene Index Toolbox [3] to index tables. A "document" is created where the table metadata fill the "fields". *TableSeer* rates the <query, table> pairs

instead of the <query, document> pairs by tailoring the classical vector space model [10] to index each table metadata file instead of the whole document.

Table 5.1. The Vector Space Model for rating <Query, Table> pairs

	$m_1(MW_1)$			$m_2(MW_2)$			...	$m_k(MW_k)$			TLB	D
	$t_{1,1}$	...	$t_{x,1}$	$t_{1,2}$	...	$t_{y,2}$	...	$t_{1,k}$	...	$t_{z,k}$	...	
$tb1$	$w_{1,1,1}$	...	$w_{x,1,1}$	$w_{1,1,2}$	...	$w_{y,1,2}$	...	$w_{1,1,k}$	...	$w_{z,1,k}$	...	
$tb2$	$w_{1,2,1}$	...	$w_{x,2,1}$	$w_{1,2,2}$	...	$w_{y,2,2}$	...	$w_{1,2,k}$	...	$w_{z,2,k}$	...	
$\vdots$	...	$\vdots$	...	...	$\vdots$	...	$\vdots$	...	$\vdots$	...	...	
tb_b	$w_{1,b,1}$	...	$w_{x,b,1}$	$w_{1,b,2}$	...	$w_{y,b,2}$	...	$w_{1,b,k}$	...	$w_{z,b,k}$	...	
Q	$w_{1,q,1}$			$w_{1,q,2}$			...	$w_{1,q,k}$			...	
$ITTF$	...	...	...		...	...	...	...	...	...	...	

5.3 Term Weighting

Equation 5.2 shows that the final weight of a term $w_{i,j,k}$ comes from three levels: the term-level weight $w_{i,j,k}^{TermLevel}$, the table-level weight boost $TLB_{i,j}$, and the document-level weight boost DLB_j, which are described in *Sections 3.2.1 – 3.2.3* respectively.

$$w_{i,j,k} = w_{i,j,k}^{TermLevel} * TLB_{i,j} * DLB_j \tag{5.2}$$

The term-level weight contributes a great proportion to the final weight of a term using an innovative weighting scheme: Table Term Frequency - Inverse Table Term Frequency (TTF-ITTF). $TTF - ITTF$ is adapted from the TF-IDF[27] weighting scheme, a widely used weighting method for free-text documents. However, it is not suitable for table ranking because it only considers the term frequency and inverse document frequency and ignores many other important impact factors, e.g, term position. In contrast

to TF-IDF, TTF-ITTF demonstrates two major differences. First, TTF-ITTF calculates term frequency in a table metadata file instead of the whole document. Second, the position of a term in the table and the document is considered. Therefore, we have

$$w_{i,j,k}^{TermLevel} = TTFITTF_{i,j,k} = TTF_{i,j,k} * ITTF_{i,j,k} \tag{5.3}$$

where $TTF_{i,j,k}$ is the term frequency of the table term t_i in the metadata m_k of the table tb_j and $ITTF_{i,j,k}$ is the inverse table term frequency. Like in the case of inverse document frequency, the idea behind using the $ITTF_{i,j,k}$ is that a term that occurs in a few tables is likely to be a better discriminator than a term that appears in most or all tables.

The importance of different table metadata elements varies. Each metadata m_k has an assigned metadata weight MW_k. For example, if a matched term comes from the metadata "*Table Column Header*," it should get more credits to the term weight than those that hit in the metadata "*Table Footnote*." In particular, if the user specifies the table metadata in the query by an advanced search (e.g. Find the tables with term "*gene*" in table caption), the MW of the metadata "*Table Caption*" should be set higher. The details of establishing metadata weight are addressed in Section *4.1*.

A longer document has a greater possibility of having more hits than a short document. Tables follow this rule as well. To avoid the bias of long documents, we normalize the term frequency by dividing the total term frequency $tf_{j,k}$. $tf_{j,k}$ refers to the total occurrence times of all the terms in the metadata m_k of the table tb_j. For free-text document retrieval, $\frac{f_{i,j,k}}{tf_{j,k}}$ is widely used to calculate the TF (Term Frequency),

where $f_{i,j,k}$ denotes the occurrence times of the term t_i in the metadata m_k of the table tb_j. However, this fraction is not suitable for the table texts because the table metadata has a different text distribution from the text distribution in the free texts. Based on the observation of hundreds of table metadata files, we notice that we should treat the texts of the table metadata files as semi-structured texts instead of the the free texts. Essentially, the semi-structured text is typically a short summary of an object, rarely more than a few words long. Table metadata files, search queries, and document abstract are, many times, similar in nature: all of them tend not to be complete sentences. They are a few words long and express a summary of a subject that the user is interested in. According to [27], which proposes the term weighting scheme for semi-structured texts, we have the following Equation 5.4 to compute the TTF.

$$TTF_{i,j,k} = (p + (1-p) * \frac{tf_{i,j,k}}{tf_{j,k}}) * MW_k \tag{5.4}$$

where p is a parameter between 0 and 1. To weight terms in the table and query, p is set to be 0.5 according to [27]. $ITTF_{i,j,k}$ is computed by Equation 5.5

$$ITTF_{i,j,k} = \log_2(\frac{b}{IDF_{i,j,k}}) + 1 \tag{5.5}$$

where b is the number of tables in T. $IDF_{i,j,k}$ denotes the number of tables that term t_i occurs in the metadata m_k.

5.3.1 Table Level Boosting (TLB)

Intuitively, a table itself is also important and influences the weight of terms in it. Besides the term-level features, *TableRank* also considers table-level features like: 1) the table frequency, 2) the length of the text that elaborates the table content in the document (namely the table reference text), and 3) the table position. These factors are embodied in the Table Level Boosting *(TLB)* factor as follows:

$$TLB_{i,j} = B_{tbf} + B_{trt} + (r * B_{tp}) \tag{5.6}$$

where B_{tbf} is the boost value of the table frequency, B_{trt} is the boost value of the table reference text, B_{tp} is the boost value of the table position, and r is a constant $\in$[0,1]. If users specify the table position in the query, $r = 1$, otherwise $r = 0$. The principle behind the Equation 5.6 is that if the query terms appear in several tables of the same document, then these tables from the document have a high relevance potential. Thus, more credits should go to the terms appearing in these tables. Section 4 displays some experimental results to show the effects with and without these boosts.

In order to calculate B_{tbf}, we propose another weighting scheme – TBF-ITBF (TaBle Frequency Inverse TaBle Frequency). Both TBF-ITBF and TTF-ITTF derive from TFIDF. The former works on the table level while the later works on the table term level.

$$B_{tbf} = TBF_{i,j} * ITBF_i = \frac{tbf_{i,j}}{tbf_j} * (log_2 \frac{b}{b_i} + 1) \tag{5.7}$$

where $tbf_{i,j}$ denotes the number of tables that include the term t_i in the document d_j, and tbf_j denotes the total number of tables in the document d_j. b is the total number of identified tables in the table collection T, and b_i denotes the number of tables in T in which the term i appears.

If the query terms appear in a table with a long table reference text or the size of the table itself is large, users may have a great possibility for achieving their desired results. nlr_j denotes the normalized total length of the table size and the reference text of tb_j over the entire document length.

$$B_{trt} = nlr_j \tag{5.8}$$

Tables usually appear in the document sections named "Experiment", "Evaluation", and "Result Analysis", etc. Sometimes, tables appear in "Related Work" or "Architecture" or "System Designing"' sections. This factor is not considered unless users specify the table position in the query because we can not make a conclusion about the priority of different sections. For those tables in the specified document section, terms should also have a table position boosting value B_{tp}. Combing these factors, we have:

$$TLB_{i,j} = \begin{cases} \frac{tbf_{i,j}}{tbf_j} * (log_2 \frac{b}{b_i} + 1) + nlr_j + B_{tp} & \text{if } r = 1 \\ \frac{tbf_{i,j}}{tbf_j} * (log_2 \frac{b}{b_i} + 1) + nlr_j & \text{if } r{=}0 \end{cases} \tag{5.9}$$

5.3.2 Document Level Boosting (DLB)

Unlike the *TTF-ITTF* and *TLB*, *Document Level Boosting* (*DLB*) is a query-independent (static) ranking. *DLB* indicates the overall importance of a document where a table appears. For a high quality document, its tables are also inclined to be important. The terms should receive a high document-level boosting. The *DLB* effectively complements the dynamic table ranking algorithm.

Given a document d_j, IV_j denotes its document Importance Value (IV). Setting the document importance value IV_j for the document d_j considers three factors: 1) the inherited citation value (IC_j) from the ones who cite the document, 2) the document origin value(DO_j), and 3) the document freshness(DF_j). Suppose there are x documents $(d_1, d_2, ..., d_x)$ that cite the document d_j, then IV_j is defined using the following recursive equation:

$$DLB_j = IV_j = IC_j * DO_j * DF_j = (\frac{\sum_{v=1}^{x} IV_v}{x}) * DO_j * DF_j \tag{5.10}$$

The inherited citation value (IC) relies on the nature of the scientific document itself by using its crucial citation link structure as a major indicator. In essence, a citation link from one document D_α to another document D_β can be seen as an endorsement of D_α. We represent it as $D_\alpha \rightarrow D_\beta$ and the document D_β is considered older than the document D_α. By this way, academic documents construct a citation network which is a *DAG (Directed Acyclic Graph)* as shown in *Figure 5.1.* The number of times D_α is cited and the quality of the ones that cite D_α are indicative of the importance of D_α. Our algorithm computes the document importance of each document using this citation network. Each node in the *DAG* highlights the fact that the importance of a node is

essentially obtained as a weighted sum of contribution coming from every path entering into the node.

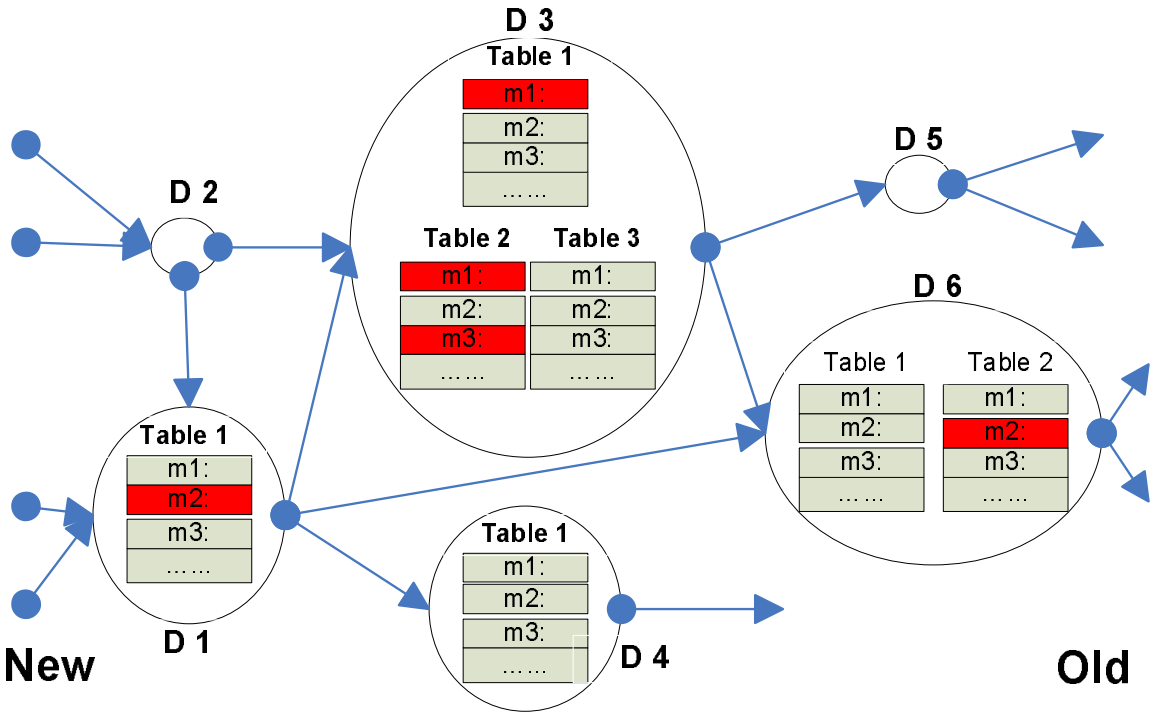


Fig. 5.1. An Example of the Citation Network

Similar to the *PageRank* approach [11], $\forall d_j \in D$, IC_j is defined recursively based on all the "incoming links" – the documents that cite d_j. The influence of a document may be repeatedly considered because of the nature of the citation structure. For example, d_6 in Figure 5 inherits IV from d_3 and d_1 but d_1 also affects d_3. An exponential decay can be useful to deal with the impact of the propagated importance. In this paper, we calculate the IV with a simplified way to avoid the computationally expensive recursions: for each node in the diagram, we calculate the IV from the nodes in the previous hierarchical level. Another important difference is that the inherited IV should

not divided by the number of the outbound links. We belief that IV of a document should not be decreased as the number of references increase. For example, suppose we have a small universe consisting of six documents $D = \{d_1, d_2, d_3, d_4, d_5, d_6\}$. For each document in the set D, the initial IC of these six documents is $1/6 = 0.167$. In Figure 5.1, both document d_1 and document d_2 link to the document d_3, each of them contributes $IC = 0.167$ to d_3 although d_1 also links to d_4 and d_6.

The document origination DO_j can be set based on the journal/conference ranking while DF_j is set according to the publication date of the document. More recent documents are preferred. The details of the parameter settings is addressed in Section *4*. The IC will be impacted by the publisher/Journal/Proceeding quality of the documents that cite it. Citation cast by documents that are themselves "important" weigh more heavily and help to make other document "important". Those terms in a document with a high IV should receive more credits.

We define the combination as the feature aggregation. Each table can have an accurate relevance measure because feature aggregation can prevent the tables from an intentional manipulation for a high rank. For example, by increasing the keyword frequency in a number of places or cutting a large tables into several smaller ones.

Chapter 6

Topic IV: Table Boundary Detection

Tables present structural data and relational information in a two-dimensional format and in a condensed fashion. Scientific researchers always use tables to concisely display their latest experimental results or statistical data. Other researchers can quickly obtain valuable insights by examining and citing tables. Tables have become an important information source for information retrieval. The demand for locating such information (table search) is increasing. Automatic table metadata extraction includes two tasks: table detection and metadata extraction. To successfully get the table data from a PDF document, detecting the boundary of the table is a crucial phase. To effectively detect tables, we design two methods: a novel *page box-cutting* method and a sparse-line detection method.

6.1 Table Boundary Detection with Box-Cutting Method

The input of the *box-cutting* method is a *Document Content File* shown in Figure 3.2. We define a *page box* as a rectangle of adjacently connected *lines* with a uniform font size in the same document page. Whether two lines merge into a same *page box* is decided by two factors: the font size and the position. We treat a line l_η in a *Document Content File* as the seed line l_{seed} of a *box* b_θ. Initially, $\eta = 1, \theta = 1$, and l_η is the only line in b_θ. If the next line $l_{\eta+1}$ in the same *Document Content File* satisfies the following

three conditions, we combine $l_{\eta+1}$ into b_θ and set $l_{\eta+1}$ as the new l_{seed}. Otherwise, $l_{\eta+1}$ will be the l_{seed} of a new *box* $b_{\theta+1}$. C1) $Font(l_{\eta+1}) = Font(l_{seed})$; C2) $l_{\eta+1}$ is adjacent to l_{seed}; C3) $l_{\eta+1}$ is close enough to l_{seed}.

TableSeer defines "*adjacent*" as no other lines exist between $l_{\eta+1}$ and l_{seed}. Different definitions of the "*close enough*" may generate different box cutting results. We define the "*close enough*" as: $Y_0(l_{\eta+1}) - (Y_0(l_{seed}) + H(l_{seed})) \leq \delta^l$. δ^l is the maximal space between two adjacent lines in a same paragraph. Figure 6.1 displays an example of a PDF document page together with the boxes segmented by our *page box-cutting* method.

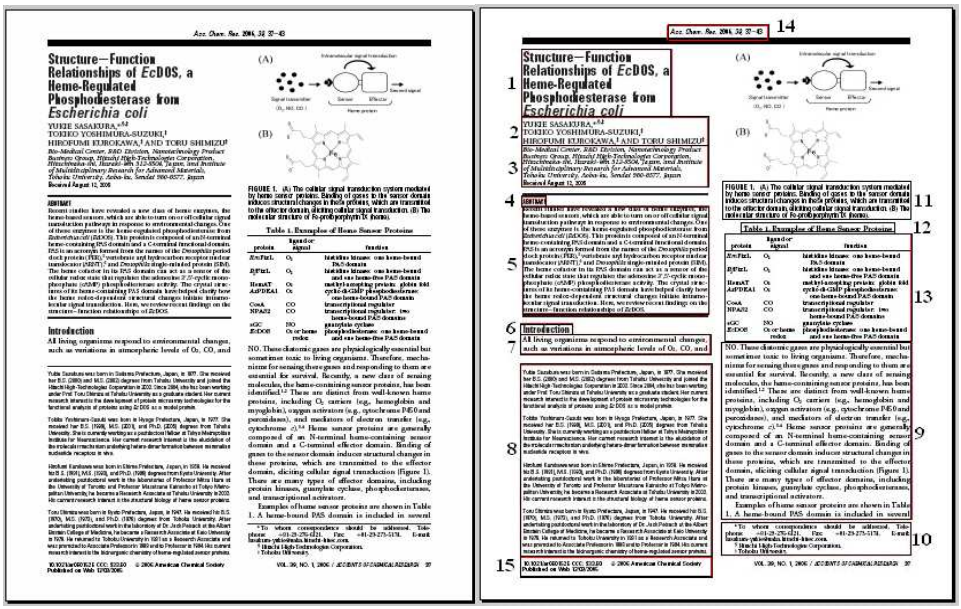


Fig. 6.1. An Example of the Page Box-Cutting Approach

For all the segmented boxes in a document, *TableSeer* classifies them into three categories: small-font boxes B^{SF}, large-font boxes B^{LF}, and regular-font boxes B^{RF},

whose font sizes are smaller than, larger than, or equal to the font size of the *document body text* F_b. Based on the observation and statistical study on the proceeding/journal templates, we summarize a set of heuristic rules (see *Table 1*), which are crucial for mapping boxes to different logical components (*titles, authors, affiliations, abstract, references, etc.*) and specific physical components (*tables, figures, etc.*).

Table 6.1. Heuristic Rules for Table Box Dictions

Rules	Content
1	• *Document title, author, affiliation, heading* $\in B^{LF}$.
2	• *Document title, author, affiliation, abstract* $\in$ Page 1, in a fixed order.
3	• *Figure caption, table caption, table body, footnote, Reference* usually $\in B^{SF}$.
5	• $\forall$ *Figure* and *table* have captions.
6	• $\forall$ *Figure* captions are beneath the figure; $\forall$ *table* captions are above the table.
7	• *Table captions* start with keywords "Table" or "TABLE" while *Figure captions* start with "Figure" or "FIGURE".

TableSeer detects tables in at most three iterations. In each iteration, we use a *keyword matching* method and a whitespace checking method to examine one box category. We create a predefined *keyword list* K^l that records all the possible starting keywords of the table captions in scientific documents, such as "Table", "TABLE", "table", "Form", "form", "FORM", "Figure", "FIGURE", etc. In the first iteration, we process all B^{SF} in the whole document page by page. If we find a B^{SF} that starts with a keyword listed in the K^l, we treat this box as a potential table candidate. Once we detect the tabular structure from its white space information, we confirm this B^{SF} as a

Algorithm 1: Table Detection using the Page Box-cutting Method

```
begin
    η ← 1, θ ← 1, l_s ← l_η, found ← 0;
    for l_η ∈ l do
        if l_{η+1} does not satisfy C_{1-3} then
            compare b_θ.fontsize() with F_b;
            classify b_θ into one of B^SF, B^LF, B^RF;
            θ ← θ + 1;
        l_{η+1} ∈ b_θ, l_seed ← l_{η+1}, l_η ← l_{η+1};
    while !found do
        for b ∈ B^(SF,RF,LF) do
            if ∃ b[StartWord] ∈ K^l then
                if there is tabular structure in b or its neighbor boxes then
                    get F_table; found ← 1; break;
        foreach b with b.fontsize() = F_table do
            if ∃ b[StartWord] ∈ K^l and has a tabular structure then
                b is a table;
end
```

real table. Once we get the font size of a table in a document, the font size of all tables in this document are fixed. We can ignore all the other boxes in different font sizes. If we do not find any table candidates, we start the second iteration on all B^{RF} because there still is a percentage of documents that contain tables displayed in F_b. If fails again, we start the third iteration to check all the B^{LF}. Usually we only have to do the first iteration.

Sometimes, the table caption and the table body may be displayed in two boxes because of the sparse layout or the trivial font difference. *TableSeer* deals with such cases as well. When we detects a box started with words in the K^l without tabular structures, we check the white space information in the neighbor boxes in both directions.

For example, F_b is *9.23* in Figure 3. Boxes *1, 2, 3, 4, 6, 11* are B^{LF}, boxes *5, 10, 8, 12, 13, 14, 15* are B^{SF}, and boxes *7, 9* are B^{RF}. In the first iteration, we detect the keyword "*Table*" in box *12* without tabular structures. We check its neighbor – Box *13*

with the same font size and confirm it as a real table. With the fixed table font size 8.76, to detect all the other tables in the same document, we only have to check the boxes with font size 8.76. With this method, *TableSeer* easily exclude more than (93.6%) document areas since the very beginning, which impressively increases the efficiency and accuracy of the table detection.

6.2 Heuristical-based Sparse-line Detection

Based on the observation, we notice that different lines in the same document page have different widths, text densities, and the sizes of the internal spaces between words. A document page contains at least one column. Many journals/conferences require two (e.g., *ACM* and *IEEE* templates) or three even four columns. In a document, some lines have the same length as the width of the document column, some are longer (e.g., cross over multiple document columns) or shorter (e.g., the short heading in paper with multiple document columns) than a column. From the internal space perspective, the majority of the lines contain normal space sizes between adjacent words while some lines have large spaces. In this paper, we define the *sparse line* as follows.

DEFINITION 1. ***Sparse Line:*** *A document line is a sparse line if either of the following conditions is satisfied: 1). The minimum space gap between a pair of consecutive words within the line is larger than a threshold* sg*. 2). The length of the line is much shorter than a threshold* ll*;*

Since the majority of the lines in a document belong to the non-sparse category, separating the document lines into sparse/non-sparse categories according to the text

internal space/density and then getting rid of the non-sparse category become a fruitful preprocessing step for the table boundary detection. Such a method has two advantages: 1) the sparse lines cover nearly the entire table content lines; 2) Narrowing down the table boundary to the sparse lines at the early stage can save substantial time and effort to analyze noise lines.

There are tables whose cells can cross over multiple table columns. In order to collect all such cells, method proposed in [99] sets up constraints on the number of such long cells within a table boundary. However, determining a reasonable value is difficult. For example, if the value is set up too tight, part of a table could be missed out. If the value is loose, noise lines will be included into the table boundary. Unlike the approach in [99], our method treats the long cells as non-sparse lines and remove temporally. To decide whether a sparse line should be included into the same table boundary, we only need to check the vertical space gaps between this sparse line and its previous neighbor sparse line. Once we merge these two sparse lines into the same table boundary, the previously temporally removed long lines (if exists any) between those two sparse lines should be retrieved back.

Different definitions of the "much shorter than" may generate different sparse line labeling results. We define it as the half of the document column width. We show a snapshot of a PDF document page in Figure 6.2 as an example. We highlight the sparse lines with red rectangles. Apparently, the table body content lines are labeled as sparse lines. Ten sparse lines are not located within the table boundary: two heading lines, one footer line, three caption lines, and four short lines that are the last line in a paragraph. We label them as sparse lines because they satisfy the second condition. Since such

short-length lines also happen in some table rows with only one filled cell, we consider them as sparse lines to avoid missing out the potential table lines. Such noise non-table sparse lines are very few because they usually only exist at the headings or the last line of a paragraph. In addition, the short length restriction also reduce the frequency. We can easily get rid of them based on the coordinate information later.

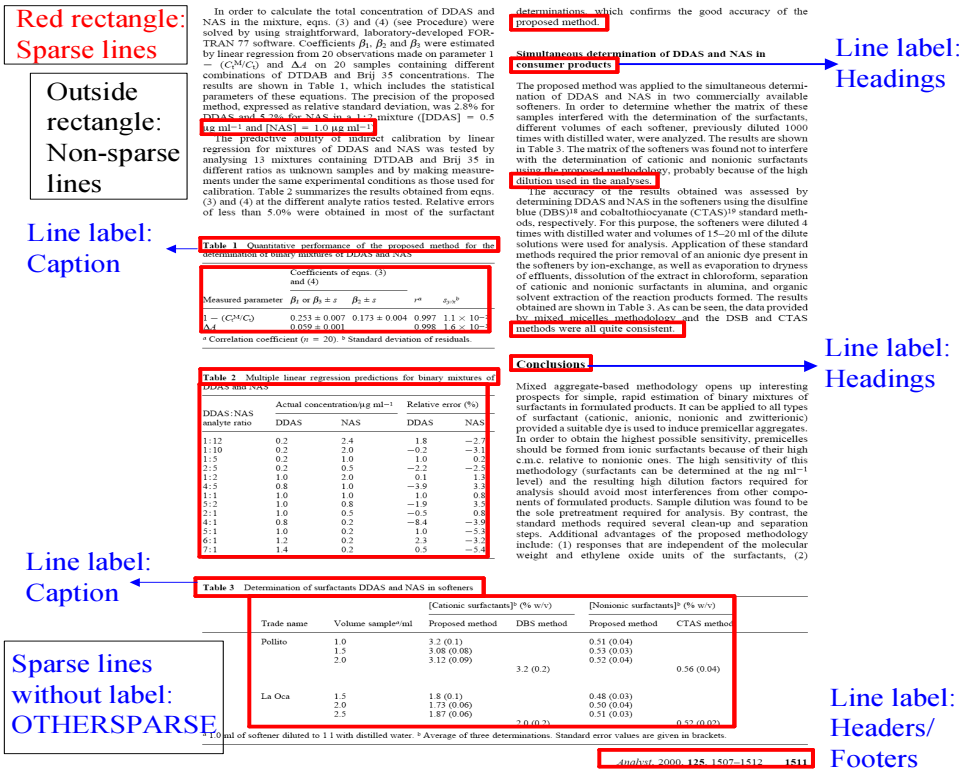


Fig. 6.2. The sparse lines in a PDF page

6.3 Machine learning based Spasrse Lines Detection

6.3.1 Support Vector Machines

SVM [12] is a binary classification method which finds an optimal separating hyperplane $x : wx + b = 0$ to maximize the margin between two classes of training samples, which is the distance between the plus-plane $x : wx + b = 1$ and the minus-plane $x : wx + b = -1$. Thus, for separable noiseless data, maximizing the margin equals minimizing the objective function $||w||^2$ subject to $\forall i, wyi(x_i + b) \geq 1$. In the noiseless case, only the so-called support vectors, vectors closest to the optimal separating hyperplane, are useful to determine the optimal separating hyperplane. Un- like classification methods where minimizing loss functions on wrongly classified samples are affected seriously by imbalanced data, the decision hyperplane in SVM is not affected much. However, for inseparable noisy data, SVM minimizes the objective function: $||w||^2 + C \sum_{i=1}^{n} \varepsilon_i$ subject to $\forall i, wy_i(x_i + b) \geq 1 - \varepsilon_i$, and $\varepsilon_i \geq 0$, where ε_i is the slack variable, which measures the degree of misclassification of a sample x_i. This noisy objective function has included a loss function that is affected by imbalanced data.

In order to increase the importance of recall in SVM, a cut-off classification threshold value $t < 0$ should be selected. In methods with outputs of class probability $[0, 1]$, then a threshold value $t < 0.5$ should be chosen. As noted before, for noiseless data, SVM is stable, but for noisy data, SVM is affected much by imbalanced support vectors. In our work, the latter approach is applied for SVM, i.e., when $t < 0$, recall is to be improved but precision decreases. When $t > 0$, a reverse change is expected.

6.3.2 Conditional Random Fields

Conditional Random Fields (CRFs) are undirected statistical graphical models, which are well suited to sequence analysis. The primary advantage of CRFs over Hidden Mark-ov Models (HMM) is their conditional nature, resulting in the relaxation of the independence assumptions required by HMMs in order to ensure tractable inference. Additionally, CRFs avoid the label bias problem, a weakness exhibited by Maximum Entropy Markov Models (MEMMs) and other conditional Markov models based on directed graphical models. Let $\mathbf{o} =< o_1, o_2, ..., o_n >$ be an sequence of observed input data sequence, for example in our case as a sequence of input lines of text in a PDF document page. Let $\mathbf{S}$ be a set of states in a finite state machine, each corresponding to a label $l \in L$ (e.g., sparse line, non-sparse line, heading line, etc.) Let $s =< s_1, s_2, ..., s_n >$ be the sequence of states in $\mathbf{S}$ that correspond to the labels assigned to the lines in the input sequence $\mathbf{o}$. Linear-chain CRFs define the conditional probability of a state sequence given an input sequence to be:

$$P(s|\mathbf{o}) = \frac{1}{Z_o} exp(\sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i)) \tag{6.1}$$

where Z_o is a normalization factor of all state sequences, the sum of the "scores" of all possible state sequences. $f_j(s_{i-1},\ s_i, \mathbf{o}, i)$ is one arbitrary feature function of m functions that describes a feature over its arguments, and λ_j is a learned weight for each such feature function.

$$Z_o = \sum_{s \in S} exp(\sum_{i=1}^{n} \sum_{j} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i)) \tag{6.2}$$

Intuitively, the learned feature weight λ_j for each feature f_j should be positive for features that are correlated with the target label, negative for features that anti-correlated with the label, and near zero for relatively uninformative features. These weights set to maximize the conditional log likelihood of labeled sequences in a training set $D = < o, \mathbf{l} >_{(1)}, ..., < o, \mathbf{l} >_{(n)}$:

$$LL(D) = \sum_{i=1}^{n} log(P(\mathbf{l}_{(i)}|\mathbf{o}_{(i)}) - \sum_{j=1}^{m} \frac{\lambda_j^2}{2\sigma^2} \tag{6.3}$$

When the training state sequence are fully labeled and unambiguous, the objective function is convex, thus the model is guaranteed to find the optimal weight settings in terms of $LL(D)$. Once these settings are found, the labeling for a new, unlabeled sequence can be done using a modified Viterbi algorithm.

We use a weight parameter θ to boost features corresponding to the true class during the testing process. Similar to the classification threshold t in SVM, θ can tune the trade-off between recall and precision, and may be able to improve the overall performance, since the probability of the true class increases. During the testing process, the sequence of labels s is determined by maximizing the probability model $P(s|\mathbf{o}) = \frac{1}{Z_o} exp(\sum_{i=1}^{n} \sum_{j=1}^{m} \lambda_j f_j(s_{i-1}, s_i, \mathbf{o}, i, \theta_s))$, where $f_j(s_{i-1}, s_i, \mathbf{o}, i, \theta_s) = \sum_{i=1}^{|} o|\theta_{s_i} t_j(s_{i-1}, s_i, \mathbf{o}, i)$, θ_s is a vector with $\theta_{s_i} = \theta$ when $s_i = true$, or $\theta_{s_i} = 1$ when $s_i = false$, and λ_j is the parameters learned while training.

6.3.3 Line Labels

Different from the traditional table boundary detection works, we use an exclusive method to label all the potential table lines. Figure 6.3 shows the inclusion-relation of the line types in a document page. The size of each block does not represent the ratio of a line type in the page. Each line type corresponds to a label in the machine learning methods.

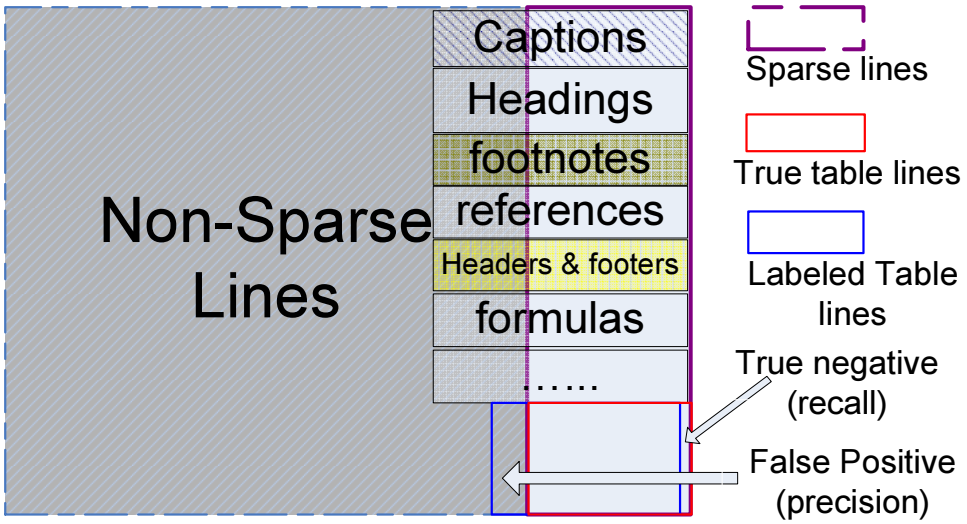


Fig. 6.3. Composition of a PDF page with line types

We design a set of labels by examining a large number of lines in scientific PDF documents. Each line will be initially labeled as either *SPARSE* or *NONSPARSE*. A line labeled as *NONSPARSE* satisfies neither of the conditions in Section 3. *NONSPARSE* lines usually cover the following document components: document title, abstract, paragraphs, etc. *SPARSE* lines cover other specific document components entirely/partially:

tables, mathematical formulas, texts in figures, short headings, affiliations, document headers and footers, and references, etc.

Even though sparse lines cover almost all table lines, a few *non-table* lines mingle in. Removing these noise lines can facilitate the table boundary detection efficiently. Therefore, for the labeled *SPARSE* lines, we label them as the following six categories: *CAPTIONSPARSE*, *HEADINGSPARSE*, *FO-OTNOTESPARSE*, *REFERENCESPARSE*, *HEADERFO-OTERSPARSE*, and *OTHERSPARSE*. *CAPTIONSPARSE* refers to a line that is the first line of a table caption or a figure caption. *HEADINGSPARSE* marks short document headings. Usually the lines labeled with the *HEADINGSPARSE* or the *CAPTIONSPARSE*, or *FOOTNOTESPARSE* only satisfy the second condition mentioned in Section 3. To label a line with these labels, additional features should be considered. For *HEADINGSPARSE* lines, the font size and type are the key features. For *CAPTIONSPARSE* lines, we should examine whether the line starts with the defined keywords or not (e.g., Table or Figure). For *FOOTNOTESPARSE* lines, the specific starting symbol is the most important factor. To identify the *HEADERFOOTERSPARSE* lines, checking the Y-axis coordinate is key. Although a large part of lines with these labels also exist in non-sparse line group, we can easily zoom in the table boundary into the last category *OTHERSPARSE* by removing such lines from sparse line set with this method.

6.3.4 Feature sets

Wise choice of features is always vital to the final results. The feature based statistical model CRFs reduce the problems to finding an appropriate feature set. This

section outlines the main features used in these experiments. Overall, our features can be classified into three categories: the *orthographic features*, the *lexical* features, and the *document layout features*. Instead of the features about white space and separators in [21], we emphasize the layout features.

6.3.4.1 Orthographic features

Most related works treat the vocabulary as the simplest and most obvious feature set. Such features define how these input data appear (e.g., capitalization etc), based on regular expressions as well as prefixes and suffixes. Because the line layout is much more important than their appearance for our line labeling problem, we do not have to consider so many orthographic features as they did. Our orthographic features include: *InitialCaptical, AllCaptical, FontSize, FontType, BoldOrNot, HasDot, HasDigital, AllDigital*, etc.

6.3.4.2 Lexical features

The lexical features includes: *TableKwdBeginning, FigureKwdBeginning, ReferenceKwdBeginning, AbstractKwdBeginning, SpecialCharBeginning, DigitalBeginning, SuperscriptBeginning, SubscriptBeginning, LineItself.*

6.3.4.3 Layout features

Our crucial features come from the layout perspective. The layout features include: *LineNumFromDocTop, LineNum-ToDocBottom, NumOfTextPieces, LineWidth,*

CharacterDensity, LargestSpaceInLine, LeftX, rightX, MiddleX, DisToPrevLine, DisToNextLine. Table 1 lists the detailed description for every layout feature.

Table 6.2. The main document layout features in our experiment

Document Layout Features	Description
$LineNumFromDocTop$	Index number from the page top
$LineNumToDocBottom$	Index number to the page bottom
$NumOfTextPieces$	Number of text pieces
$LineWidth$	Width of the line
$CharacterDensity$	$LineWidth$/Number of characters
$LargestSpaceInLine$	Largest space within the line
$LeftX$	X-axis value of leftmost line end
$EndX$	X-axis value of rightmost line end
$MiddleX$	X-axis value of middle point
$TheDisToPrevLine$	Vertical gap to the previous line
$TheDisToNextLine$	Vertical gap to the next line

6.3.4.4 Conjunction features

So far all the previous features are described over a single predicate. In order to capture relationships that a linear combination of features cannot capture, we look at the conjunction of features. CRFs provide the function to determine the label of a line by taking into account information from another line. In our work, we set the window size of features as *-1, 0, 1* to conjunct the current line with the previous line and the following file. In order to avoid the overflowed memory and exacerbated overfiting generated by the all possible conjunctions and only generate those feature conjunctions with significant improvement functions, we turn to feature induction as described in [65]. We start with no feature and choose new features interactively. In each iteration, we evaluate some sets of candidates using the Gaussian prior, and add the best ones into the model.

6.4 Line Construction in PDFs

Different from most CRF applications, the unit of our problem is a document line, instead of a single word. Before classifying the document lines, we have to construct the lines first. To construct the document lines, we deal with the PDF source file character by character as well as the related glyph information through analyzing the *text operators*[1]. Adobe's Acrobat word-finder provides the coordinate of the four corners of the quad(s) of the word. The PDFlib Text Extraction Toolkit (TET) also provides the function to extract the text in the different levels (character, word, line, paragraph, etc.). However, it only provides the content instead of other style information in all the levels except the character level. If we want to do some further work, content itself is usually not enough. We have to calculate the corresponding coordinates for the higher levels by merging the characters. Similar to Xpdf library, we adopt a bottom-up approach to reconstruct these characters into words then lines with the aid of their position information and saves the results. To convert characters into words then lines, we adopt some heuristics based on the distance between characters/words. For each document page, we construct lines according to their internal word relative position information and width. Within a same word, different characters have the same font properties. However, within a same line, font diversity may exist among different words (e.g., the superscript, the subscript, or mathematical symbols). The main unique place of our method is that we only analyze the coordinate information. Font information, the frequently adopted parameter, is not used in our method as the rule to decide whether merge the next word into the same line

[1]PDF Reference Fifth Edition, Version 1.6

or not. Therefore, the font information does not affect the performance of the sparse line detection.

Table 6.3. The parameter thresholds we adopted for word reconstruction

P	Definition
α	the vertical distance between two top Y-axis values: $alpha = Y_{i+1} - Y_i$
β	the vertical distance between two bottom Y-axis values: $beta = Y_{i+1}^{'} - Y_i^{'}$
γ	the horizontal distance between these two characters: $\gamma = X_{i+1} - X_i^{'}$
δ	the vertical distance of two characters
θ	the maximal width of the space with a word
η	the maximum vertical distance between two characters in a same line

Formally, we define a document as a set of pages D $= \cup_{k=1}^{n}(P_k)$, where n is the total page number. Each page P_k, can be denoted as an aggregation of characters $C \in \{Character\}$. c_i and c_{i+1} are a pair adjacent (no other character exists between them) characters. Initially, we get the coordinate of the first character c_0 in a document page. All the characters in C share a common set of attributes $\{([X, X^{'}], [Y, Y^{'}], W, H, F, T)\}$, where $[X, Y]$ is the pair of coordinators of the upper-left corner of the character while $[X^{'}, Y^{'}]$ is the coordinates of the bottom-right corner of the character. The original point of the X-Y axes is the left-bottom corner of a document page. W/H denotes the width/height of the component, F is the font size, and T is the text. Figure 6.4 shows the coordinates of an example character. not use the font size because in many journals and archives, the font information (the font type and the font size) is not so standard

as we imaged. Considering such unreliable information will incur more error to the final results.

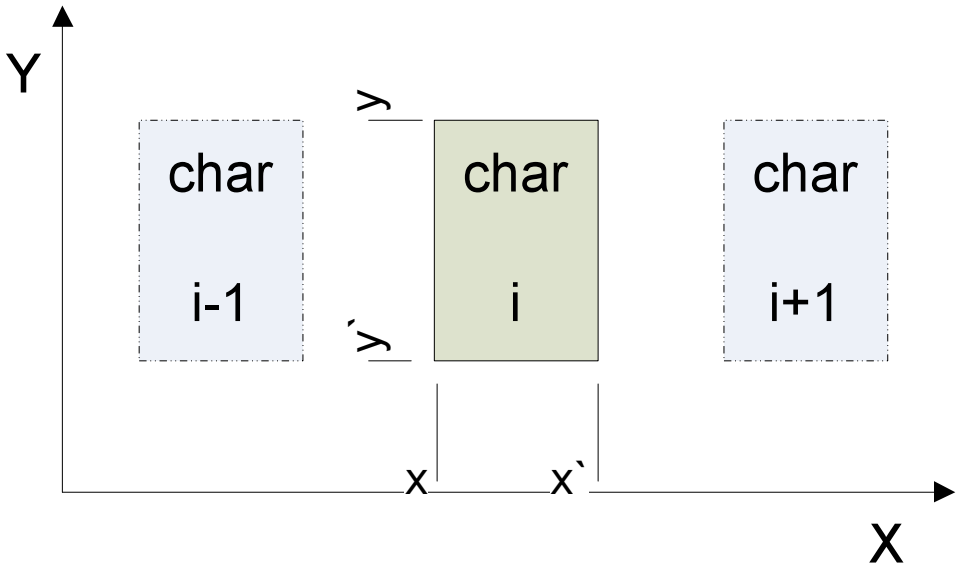


Fig. 6.4. The coordinates of a character in a PDF document page.

Since the character C is the fundamental component of a document, other components can be constructed recursively from it. For example, a document *page* P_k can be denoted as an aggregation of *words* $W = \{w_j | w_j = ([X_{w_j}, Y_{w_j}]), ([X^{'}_{w_j}\ Y^{'}_{w_j}]), W_{w_j}, H_{w_j}, F_{w_j}, T_{w_j}\}$. A document word w_j is equal to $\cup_{i=1}^{m} c_i$, where m is the total number of characters in the word w_j. Figure 6.5 enumerates all the relative positions of a pair of adjacent characters c_i and c_{i+1}. Their coordinates are $([X_i, X_i^{'}], [Y_i, Y_i^{'}])$ and $([X_{i+1}, X_{i+1}^{'}], [Y_{i+1}, Y_{i+1}^{'})$ respectively. For the word reconstruction, we define several parameters and thresholds, which are listed in Table 2:

Figure 6.5 (a) presents a common character pair in the same line. $Y_{i+1} = Y_i(\alpha = 0)$ and $Y_{i+1}^{'} = Y_i^{'}(\beta = 0)$. If γ is smaller than a given threshold θ, the second character

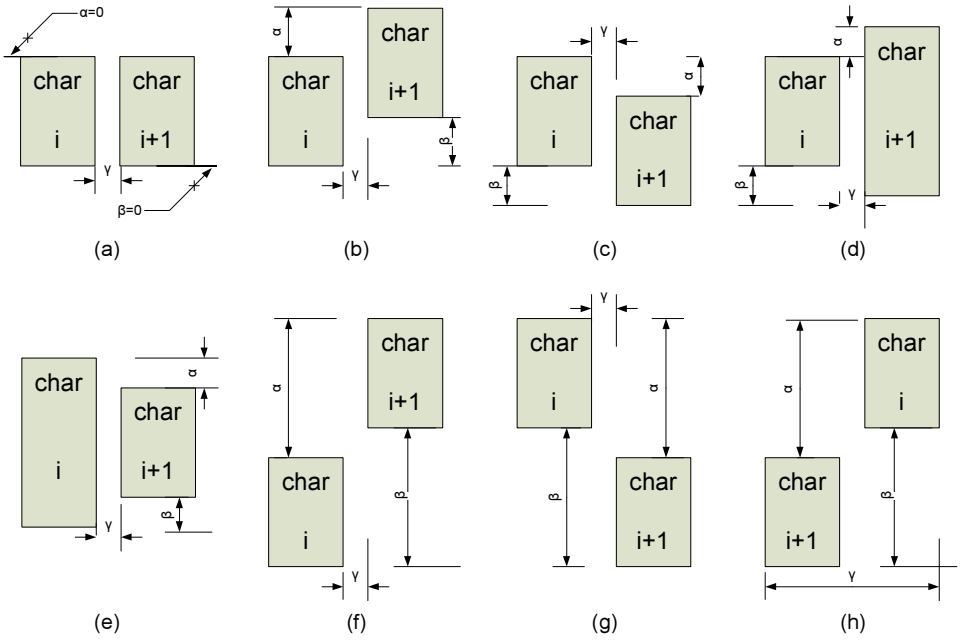


Fig. 6.5. The coordinates of the example cases of the character pairs

c_{i+1} can merge with c_i into a same word. Otherwise, we treat c_i as the last character of the current word and c_{i+1} as the starting character of a new word.

Figure 6.5 (b) – (e) display several examples of the same-line character neighbors with partial vertical overlaps. Superscript is a typical case of Figure 6.5 (b) while subscript is a typical case of Figure 6.5 (c). Figure 6.5 (d) and (e) show the font size changing within a document line. All these character pairs are also same-line characters. Every case has to satisfy some fixed conditions as follows: $Y_{i+1} \geq Y_i \geq Y_{i+1}^{'} \geq Y_i^{'}$ (Figure 6.5 (b)), $Y_i \geq Y_{i+1} \geq Y_i^{'} \geq Y_{i+1}^{'}$ (Figure 6.5 (c)), $Y_{i+1} \geq Y_i \geq Y_i^{'} \geq Y_{i+1}^{'}$ (Figure 6.5 (d)), and $Y_i \geq Y_{i+1} \geq Y_{i+1}^{'} \geq Y_i^{'}$ (Figure 6.5 (e)). For these cases, we decide whether c_i and c_{i+1} go into the same word or not, by comparing γ with the same threshold θ;

To analyze the character pairs in Figure 6.5 (f) – (h), we introduce another threshold η: *the maximum vertical distance between two characters in a same document line.*

In 6.5(f), the fixed constraint is $\delta = (Y_{i+1}^{'} - Y_i) > 0$. In Figure 6.5($g$) and (h), the fixed constraint is $\delta = (Y_i^{'} - Y_{i+1}) > 0$. If $\delta > \eta$, C_i and C_{i+1} will belong to different lines. Otherwise, we treat them as the character neighbors in a same line and decide whether they go to the a same word. Starting a new document column in a page is a typical example with large γ for case f, and starting the next line is a typical example with large but minus γ for case h. Using the Table 1 in Figure6.2 as the example, we show the merged words in Figure 6.6. Each red rectangle refers an independent word.

Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

	Coefficients of eqns. (3) and (4)			
Measured parameter	β_1 or $\beta_3 \pm s$	$\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1 - (C_t^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient ($n = 20$). [b] Standard deviation of residuals.

Fig. 6.6. The merged words in a table after the character → word phase

Now a document page p_k can be denoted as an aggregation of words W. Similar to the characters, we can also treat words as rectangle objects in a document page. The coordinate nature of characters in the previous section is also applicable to the words. For a pair of word neighbors, w_i and w_{i+1}, the possible relative locations are same of the cases listed in Figure 6.5. Using the concept in Section 5.1, we should treat the *word* here as the *character* there and treat the *text piece* here as the *word* there. We believe that in the non-sparse lines, all the words can be merged into one piece. The parameters

and the thresholds in Table 2 can be reused with only the value resets of γ and θ. Still using the Table 1 in Figure6.2 as the example, we show the merged lines in Figure 6.7.

After the combination, we check the number of text pieces in each line along the Y-axis sequence, if the number is larger than one, we label this line as the sparse line. If the number is one but it satisfy the first condition in Section 3, we also treat it as a sparse line. Still using the Table 1 in Figure6.2 as the example, we show the merged lines in Figure 6.7. For all the eight lines, the number of text pieces are *1, 1, 1, 1, 5, 5, 4,* and *1* respectively. We treat line 5, 6, and 7 are sparse lines because they contain more than one text piece. We also treat the line 3 and 4 as sparse lines because of the small width.

Table 1 Quantitative performance of the proposed method for the determination of binary mixtures of DDAS and NAS

	Coefficients of eqns. (3) and (4)			
Measured parameter	β_1 or $\beta_3 \pm s$	$\beta_2 \pm s$	r^a	$s_{y/x}{}^b$
$1 - (C_t^M/C_t)$	0.253 ± 0.007	0.173 ± 0.004	0.997	1.1×10^{-2}
ΔA	0.059 ± 0.001		0.998	1.6×10^{-3}

[a] Correlation coefficient ($n = 20$). [b] Standard deviation of residuals.

Fig. 6.7. The merged lines in a table

6.5 Detecting the table boundary based on the keywords

After the sparse line detection and noisy line removal, we can easily detect the table boundary by combining the lines labeled as *OTHERSPARSE* with the table keywords. Here we define the main table content rows as the table boundary, which does not have to include the table caption and the footnote. In order to enhance the performance of the table starting location detection, we consider the keyword information. We can directly detect the table boundary by detecting the tabular structure within the sparse line areas.

In our method, we define a keyword list, which lists all the possible starting keywords of table captions, such as "Table, TABLE, Form, FORM," etc. Most tables have one of these keywords in their captions. If more than one tables are displayed together, the keyword is very useful to separate the tables from one another. Once we detect a line (not only the sparse line) starting with a keyword, we treat it as a table caption candidate. Then we check other lines that are located around the caption and merge them into a sparse area according to the vertical distances between adjacent lines. Such sparse-line areas are the detected table boundary. The vertical distance is the key feature to filter out most remaining noise lines. Because the texts within the detected table boundary will be analyzed carefully in the later table structure decomposition phase, we treat *recall* more important than *precision* here. Once we locate the table boundary, we check this area and try to retrieve the long table lines that are labeled as non-sparse lines to improve the *recall*.

Chapter 7

Topic V: Table Structure Decomposition

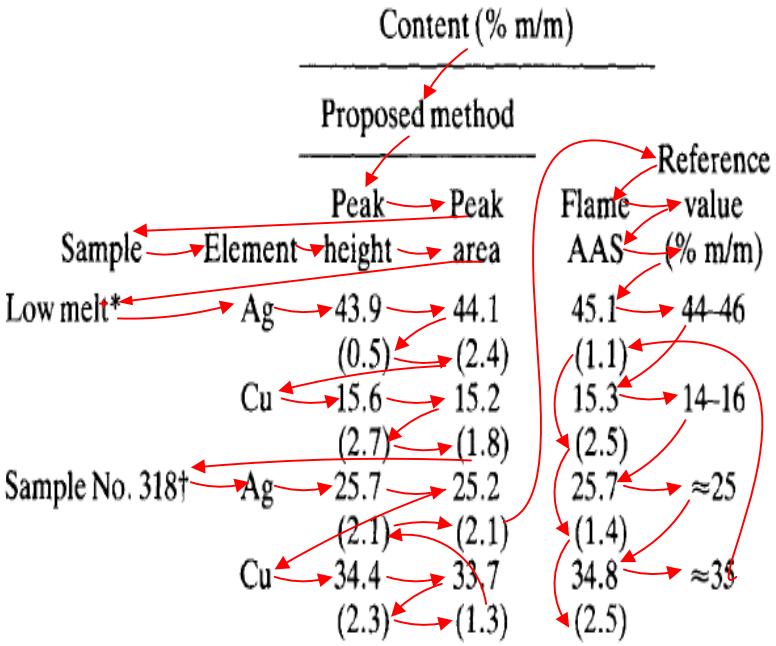
7.1 The text Fixing the Sequence Error of the Sparse Lines

Tables, as a specific document component, are ubiquitous. To thoroughly analyzing the table content, locating the table boundary in a document is the first and crucial step for consequent applications (e.g., the table search). Because most of the tables in scientific documents are text-based tables, solely analyzing the text objects in a PDF file can achieve good performance in terms of saving efforts on other objects. However, existing text extraction tools face a common problem: the extracted text pieces follow a different sequence from its original appearance in PDF files. Although the wrong sequence does not affect the PDF documents displaying for browsing purpose, it can severely interfere the document content analyzing works that rely on the information of the relative location and sequence among the text pieces. Table boundary detection is a typical example. In order to improve the accuracy of the table boundary detection, we need to recover the extracted text sequences to the orders that comply with the human's reading habit. The sequence errors can be classified into two types: *within-table errors* and *beyond-table errors*. In this paper, we propose two text sequence recovering algorithms to fix them. Our algorithms start with the detected sparse lines in each PDF page and detect the table boundaries by analyzing the sparse areas and the caption information. If interested, the details can be found in [57] and [58].

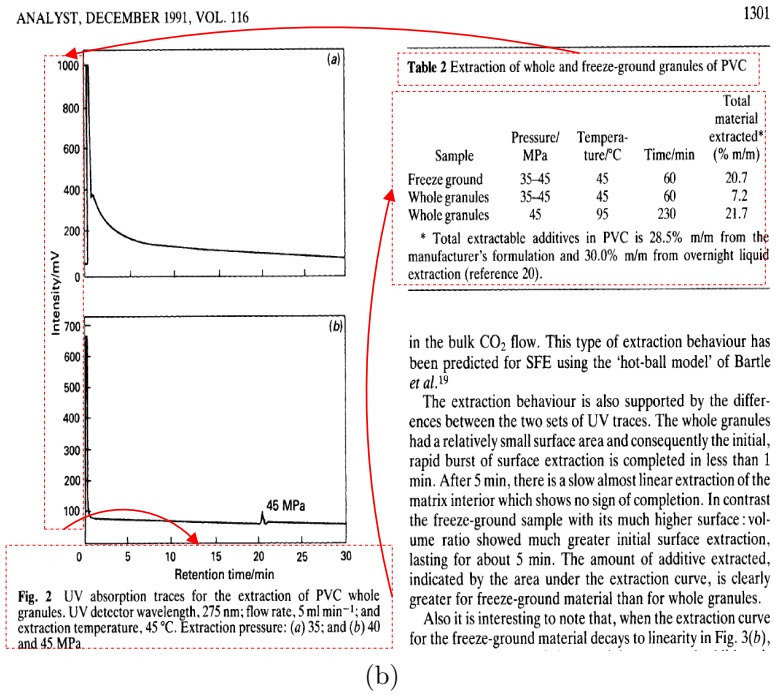
7.1.1 The text sequence problem

Given a PDF document page, the human reading sequence is usually top-down, left-to-right, and line by line, column by column. Particularly, some document components in the page may cross multiple document columns, such as the document title, the affiliation, some wide tables and figures. Although the text extraction tools can extract all the text information from a PDF page, the sequence of the exported texts may not reflect the human reading ordering. The text sequence error is a common problem shared by existing text extraction tools (e.g., PDF2TEXT, PDFBOX, TET, PDFTextStream). Such text sequence error happens frequently in the table contents, which may generates serious wrong results during the table detection, such as segmenting a table into several pieces, wrong column/row information, and omitting cells/whole tables etc. Some of the errors can be fixed with complex post-processing while others can not. According to the scope of the text sequence errors, we category them into two classes: *inside-table* text sequence error and *beyond-table* text sequence error.

Figure *7.1a* provides an example of the *inside-table* text sequence error. The text extraction tools export 33 sparse lines in such a sequence: *"Content (% m/m)", "Proposed method", "Peak Peak", "Sample Element height area", "Low melt Ag 43.9 44.1", "0.5 2.4", "Cu 15.6 15.2", "2.7 1.8", "Sample No.318+ Ag 25.7 25.2", "Cu 34.4 33.7", "2.3 1.3", "2.1 2.1", "Reference", "Flame value", "AAS (%m/m)", "45.1 44.46", "15.3 14-16", "25.7 ≈25", "34.8 ≈35", "1.1", "2.5", "1.4", "2.5".* The sequence is: the first four columns, then the last two columns. Moreover, within the same column, the ordering of rows does not exactly follow the top-down manner. In this type, the errors



(a)



(b)

Fig. 7.1. Two examples of the text sequence errors

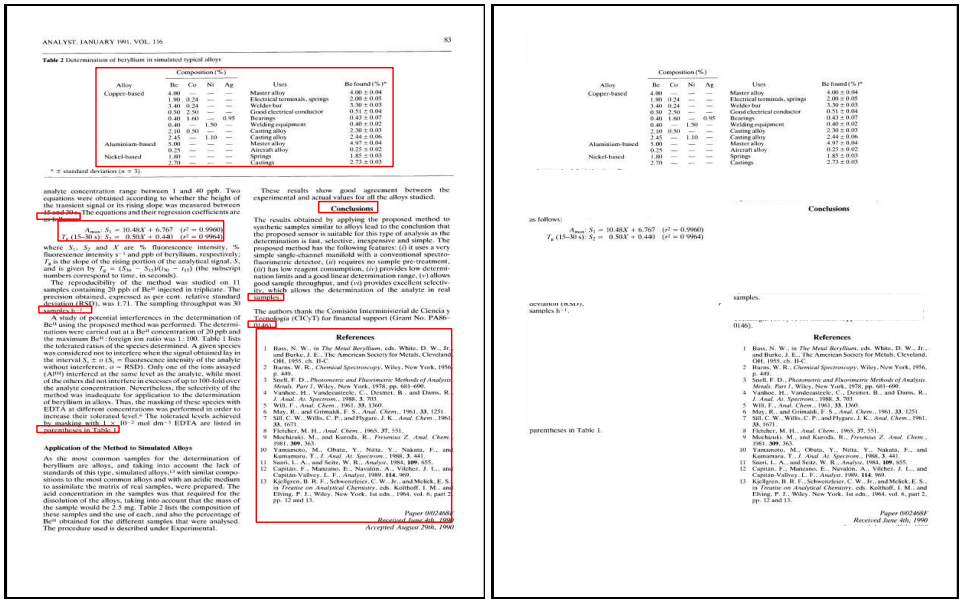


Fig. 7.2. A document page with a cross-column table (left) and its sparse lines (right) happen only within the table boundary and no outside text interrupts them. No matter how disordered the sequence is, all the contents of a table come together as a whole.

In *beyond-table* text sequence error, the table lines are partitioned into at least two parts by non-table lines. The text extraction tools usually export a part of the table content first, jump to another document area and process it, and then come back to finish the rest of the table sections. Such alternation happens at least once. Figure 7.1b shows an example: the text extraction tools output the table caption first on the second document column, then jump to the first document column on the left and process some scattered texts in the figures. After telling the figure caption, they come back to the table area. Sometimes, the tools extract several table columns then jump to other areas (e.g., the following regular document contents) to finish several lines or paragraphs, then return to the table data again. With such *beyond-table* text sequence errors, collecting all the table cells is a challenging task because it is difficult to predict how far the distances are among different table parts.

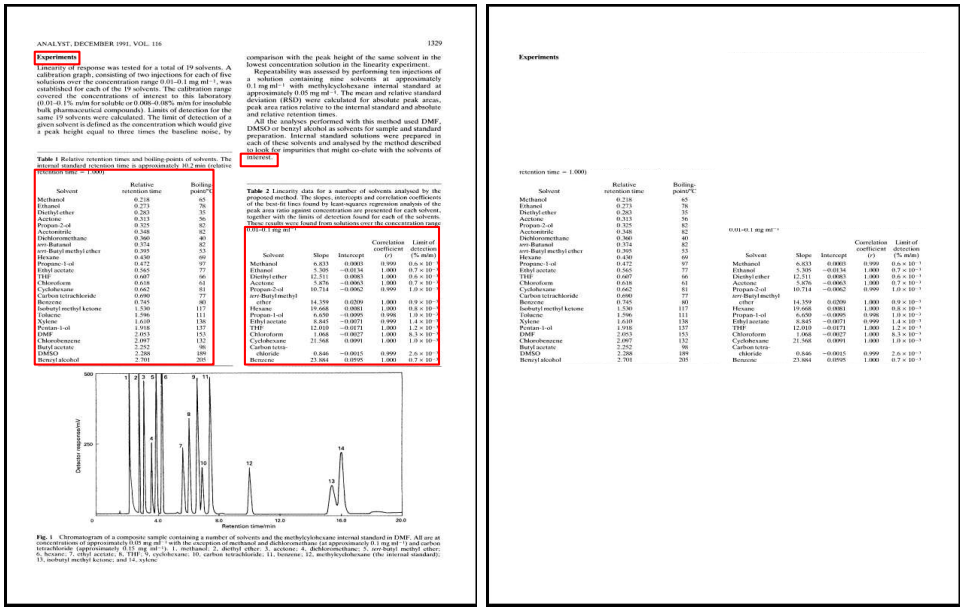


Fig. 7.3. A document page with two parallel tables (left) and its sparse lines (right)

In order to accurately collect the table cells from a page and recover the sequence, resorting the texts in the page is one approach. B. Yildiz et. al mentioned the similar idea in [99] but without any detail about the methodology and the performance. Other proposed approaches [13] exploit geometric or typographic features of the page objects, and go further in exploiting the content of object. X-Y cut approach is a representing method. However, its performance is not satisfying. Moreover, these proposed works need to process both sparse lines and non-sparse lines. The more non-sparse lines involve, the higher possibility errors incur.

7.1.2 Text sequence recovering algorithms

To fix the text sequence error problem, we propose two methods to resort the detected sparse lines within a given area. For the inside-table text sequence errors, the area is the table itself. For the beyond-table text sequence errors, the area is the whole page. Some pages have multiple document columns, whether considering such

information or not may generate different results for different cases. In this paper, we try both of them and analyze their advantages/disadvantages.

7.1.2.1 Algorithm 1: Considering the document column information

The recovering procedure includes two parts: the *cross-column resorting*, and the *within-column resorting*. Here the *column* refers to the document column instead of the table column. How to get the document column information is out of scope of this paper. We adopt an easy but effective method by calculating the average length of the non-sparse lines then comparing with the document width. For the *inside-table text sequence disordering*, we only implement the *within-column resorting*. For the *beyond-table text sequence disordering*, we have to implement both resorting ways.

In the *cross-column resorting*, SL denotes the set of all the sparse lines in a document page: $SL = \bigcup_{i=1}^{m}(l_i)$. For a document column CO_k, its beginning and ending X-axes values are X_{co_k} and $X^{'}_{co_k}$. Each column has a vector v to store its sparse lines. Initially, the vector is empty. For each sparse line, we compare its X-axis coordinates with document columns and append it in the matched vector v, if applicable. Some sparse lines may cross several document columns. Such lines usually are lines in cross-column tables/figures or long document titles or affiliations, etc. We record such lines in another vector v_{cc}.

In the *within-column resorting* part, the inputs include the filled vector V and V_{cc}. The output is a new vector V_{sorted}, which stores all sorted sparse lines. For each document column CO_k, we get all the Y-axis values from the sparse lines in vector v_k. Among these Y-axis values, we identify all the non-duplicated values and sort them, and

Algorithm 2: Sorting sparse lines across the document columns

Input: All sparse lines $SL = \cup_{i=1}^{m} sl_i$ in a page
Input: the information of Document columns $CO = \cup_{k=1}^{p} co_k$. The sparse lines that are located in the column co_k will be stored in the corresponding vector v_k
Input: a vector V_{cc} to store the cross-column sparse lines. $V_{cc} = \emptyset$

```
1  begin
2      foreach sl_i ∈ SL do
3          x_{sl_i} ← the starting X-axis of sl_i;
4          x'_{sl_i} ← the ending X-axis of sl_i;
5          while the next document column co_k exists do
6              x_{co_k} ← the starting X-axis of co_k;
7              x'_{co_k} ← the ending X-axis of co_k;
8              if (x_{co_k} ⩽ x_{sl_i} < x'_{sl_i} ⩽ x'_{co_k}) then
9                  v_k = v_k + sl_i;
10                 break;
11         if sl_i fits no document column then
12             v_cc = v_cc + sl_i;
13 end
```

store them in a new vector V_y. With these Y-axis values, we sort all sparse lines in vector v_k accordingly. This method works well for the tables that are located within one document column (e.g., the table in Figure 7.1b). However, for the wide tables that cross more than one document column (see Figure 7.2a), this algorithm may generate errors because it is difficult to know which lines should cross the document columns in advance. If there are overlaps between the table column spaces and the document column spaces, this method will segment a whole table into at least two parts. Although some table lines may block the document column space, it is difficult to identify the last line of the table.

7.1.2.2 Algorithm 2: Without considering the document column information

To deal with the above special cases, we propose another text sorting algorithm without considering the document column information at the beginning. The algorithm has four steps: 1) obtaining all the non-duplicated Y-axis values of the sparse lines within an area; 2) sorting these Y-axis values from the top to the bottom; 3) ordering all the sparse lines in the area according to the sorted Y-axis values; 4) exporting the sparse lines in the area orderly. For the *inside-table* text sequence error problem, the area is the table itself. For the *beyond-table text sequence error* problem, this algorithm works better for the wide tables because the area refers to the whole page. All the sparse lines of cross-column tables will be exported out uninterruptedly. The high-risk case of this algorithm is the parallel tables. Parallel tables mean two tables that are located within two adjacent document columns and both tables have overlap Y-axis areas (see Figure 7.3a). Such cases happen very infrequently. In addition, they can be fixed easily by considering the width and the location of table captions.

7.1.2.3 Detecting the Table Boundary based on the sorted sparse lines

With the sorted sparse lines, we detect the table boundary by combining the table caption information. We define a keyword list, which lists all the possible starting keywords of table captions, such as "Table, TABLE, Form, FORM," etc. Each table caption starts with a keyword. If more than one table is displayed together, the keyword is very useful to separate the tables from each other. At the beginning, we check all the lines (not only the sparse line) in a page. Once we detect a line starting with a keyword,

Algorithm 3: Sorting sparse lines within a document column

```
   Input: V, V_cc, V_y, V_sorted
 1 foreach v_k ∈ CO_k do
 2     V_y ← all the Y-axis values in v_k;
 3     V_y ← all the non-duplicated objects in V_y;
 4     V_y = V_y − V_cc;
 5     sorted(V_y) ascendingly;
 6     foreach Y_q ∈ V_y do
 7         foreach sl_i ∈ V_k do
 8             if Y_sl_i == Y_q then
 9                 V_sorted += sl_i;
10                 V_k -= sl_i;
11 sort lines in V_cc according to Y-axis;
12 insert V_cc into V_sorted according to Y-axis;
```

we treat it as a table caption candidate. Then we check the subsequent sorted sparse lines and merge them into a sparse area, according to the vertical distance between the lines. Different table captions provide different hints and restraints on whether we include the cross-column sparse lines or not.

If a table caption starts in the second document column (e.g., the right table in Figure 7.3a), the table boundary can not span the first column. If a table caption satisfies one of the following conditions, it is a cross-column table: 1) the width of the table caption is larger than the document column width; 2) the starting X-axis value of the caption is on the right of the midpoint of the first document column. Once we confirm a table as a cross-column table, we treat each Y-axis value as a table row. Otherwise, every table row can only include the sparse lines within the current document column. Because we will zoom in and analyze the detected table boundary carefully in the later table structure decomposition phase, we treat *recall* more importantly than *precision* by accepting false positive table boundary contents.

If a table caption is shorter than the document column width, and its beginning position aligns with that of the first document column (see the table caption in Figure 7.2), it is difficult to judge whether this table a cross-column table or not. For such a case, we assume it as a single-column table initially, after figure out the document column where the caption belongs to, we check the same Y-axis area in the next document column. If this area is also a sparse area, we merge the new area into the table boundary and treat this table as a cross-column table. Once we find a new table caption in the new area, it indicates the existing of parallel tables.

7.2 Table Structure Analyzer

The input of the table structure analyzer is the detected table boundary with starting position of a table candidate. The text sequence error problems have already fixed by the methods in [ICDAR2009]. Existing text extraction tools face a common problem: the extracted text pieces follow a different sequence from its original appearance in PDF files. Although the wrong sequence does not affect the PDF documents displaying for browsing purpose, it can severely interfere the document content analyzing works that rely on the information of the relative location and sequence among the text pieces. Table boundary detection is a typical example. In order to improve the accuracy of the table boundary detection, we proposed recover the extracted text sequences to the orders that comply with the human's reading habit. We analyze each text piece in the table boundary. From the layout position perspective, each line must satisfy one of the following five cases:

- Case 1: Current row, the next column;
- Case 2: The next row, the first column;
- Case 3: The next line in the current cell;
- Case 4: The line of table caption;
- Case 5: The line of table footnote;

For Case 1 and Case 2, we further divide them into the following three different types according to the cell alignment: Type a: Normal cell; Type b: Nested cell; Type c: Spanned cell; Figure 7.4 shows an example table with all the cells and their corresponding scenarios.

Table 3: xxxxxxxxxxxxxxxxxxxx (Case 4)

--		C(ase) 1a	C 1a	C 1a	--
C 2a		C 1c C 3		C 1a	C 1b C 3	C 1a
C 2a	C 1b	C 1a	C 1a	--	C 1a
C 2c		C 1a	C 1a C 3	C 1a C 3	C 1b	C 1a
--		C 2a	C 1a	C 1c	C 1a

Note: xxxxxxxxxxxxxxxxxxxxxxxxx (Case 5)

Fig. 7.4. An example table with all the cell scenarios

Our table structure analyzer proves/denies each table candidate box by recognizing the table structure and matching each text piece in table candidate box into one of above cases. The detailed table structure decomposition is not a main topic of this paper. Actually, the extracted table metadata is the input of our table classification.

Chapter 8

Interface Designing

In order to facilitate the browsing of results, *TableSeer* provides a user-friendly interface to present the sorted results. Figure 8.1 shows a screen shot of the search result interface generated by the TableSeer. For each matched table, we list the table-related information (e.g., table caption, table column heading), the basic document information (e.g., the document title, author, and affiliation), as well as the table location information in the document. We also highlight the referenced text to the table in the document. Users are enabled to view the napshot of the matched tables. Once clicked, table information is enlarged for better visualization. If they want to read the whole document further, they can open the original whole PDF document. The users can specify a fielded query via the query interface of *TableSeer* (See Figure 8.1). For example, for a given query Q (e.g. "water sample" in Figure 8.1), if the user specifies the query term location, e.g., the metadata *"Table Caption,"* the query term will only be filled in the metadata *"Table Caption"* columns.

TableSeer offers two levels of searches: basic searches and advanced searches. A basic search allows a search with simple keywords and then the matched results are returned in ranked order. For advanced searching, users can set more complex queries (see Figure 8.2). Unlike the current search engines that index and check whole documents, *TableSeer* indexes and checks only the table metadata files.

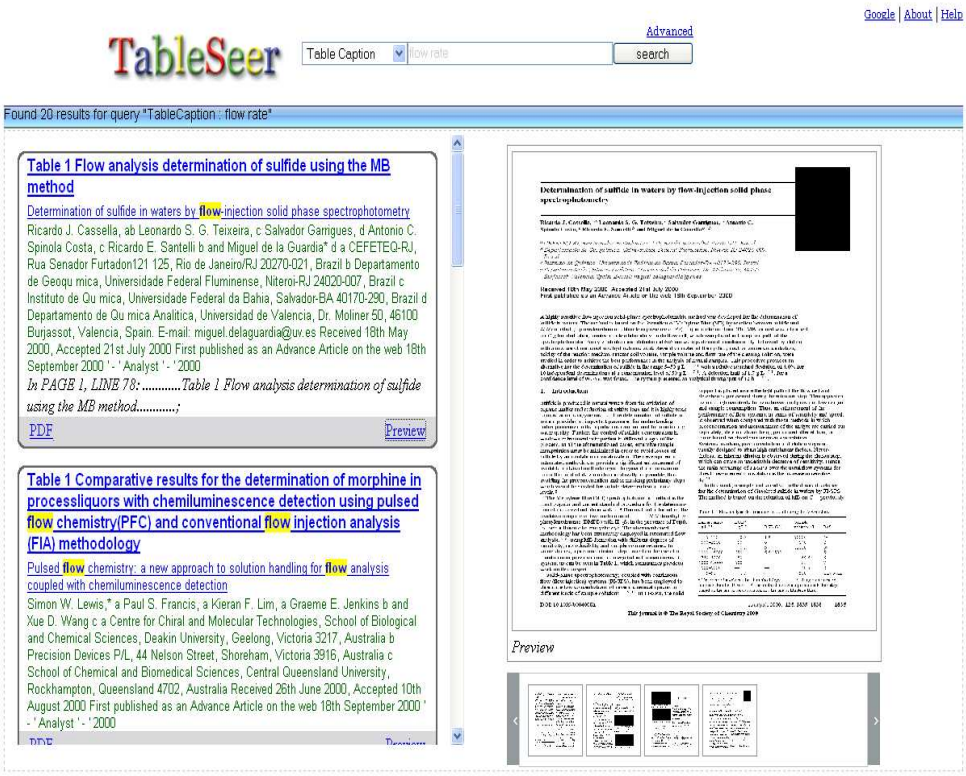


Fig. 8.1. The Query Interface of TableSeer system based on RSC repository

Fig. 8.2. An Example of the Advanced Search Interface

Fig. 8.3. The Query Interface of TableSeer system in *CiteSeer*^{x}

Chapter 9

Quantitative Study

9.1 Introduction

Although tables are widely used, there is not a standard on the table structure design. There are many issues that go into the design of tables and will impair the table data readability, accessibility, and reusability. Unfortunately, most issues are ignored. Designers often feel that they have carte blanche to use tables as they will to layout web pages and document pages. The main reason can be traced back to the beginning of the table appearance. At that time, people adopts tables just for a concise information presentation and a better data readability. How to facilitate an accurate table structure decomposition and an efficient table data sharing is out of consideration at that moment.

Most publishers do give guidelines on the table format, e.g., the "Figures and Tables" section in the IEEE Paper style file. However, such guidelines usually only give suggestion on the table caption location and the text font size (e.g., IEEE defines that "Tables must be numbered using uppercase Roman numerals. Table captions must be centered and in 8 pt regular font with small caps. Every word in a table caption must be capitalized except for short minor words. Captions with table numbers must be placed before their associated tables"). No guideline is given on the table structure and layout. Authors can design the tables as a go-as-you-please job. Different people may design different table structures.

We processed more than two million scientific documents in digital libraries from several fields, such as computer science, chemistry, archeology, etc. After processing these documents and extracted approximately one million PDF tables from the English documents in three fields: CiteSeer repository in computer science field, RSC repository in chemistry, and archeology journals, we noticed that the table diversity is a very important but easily ignored phenomenon. In order to have a better table extraction performance, we have to have a deep understanding on the nature of tables in digital libraries. To the best of our knowledge, no table characterization is implemented.

This section has the following contributions: 1) we processed the largest scientific table collection in scientific PDF format, which is an overlooked document format in table analysis field; 2) we implemented the largest quantitative study on tables in scientific digital libraries, from not only the table layout and structure but also the document background and table location perspectives. 3) Based on the results of table quantitative study, we evaluate the performance of our TableSeer system as well as each part (e.g., table boundary detection, table structure decomposition, etc). 4) Combining the results of table quantitative study and the TableSeer performance evaluation, we try to classify tables in the structure level. According to a set of features, we defined well-designed tables and bad-designed tables.

9.2 Table Characterization

9.2.1 Table Terminology

Before elaborating the table characterization and the quantitative study, we first introduce the common table terminologies, which will be used in this section.

The famous Wang's model is shown in *Figure 9.1.* He defined the terminology for the parts of a table (such as Stub/box head, Stub header, Boxhead, Column, Row, Cell, Box, Stub separator, and Boxhead separator) represented as a row column structure. Most terminologies are inherited from The Chicago Manual of Style except the Stub header and block.

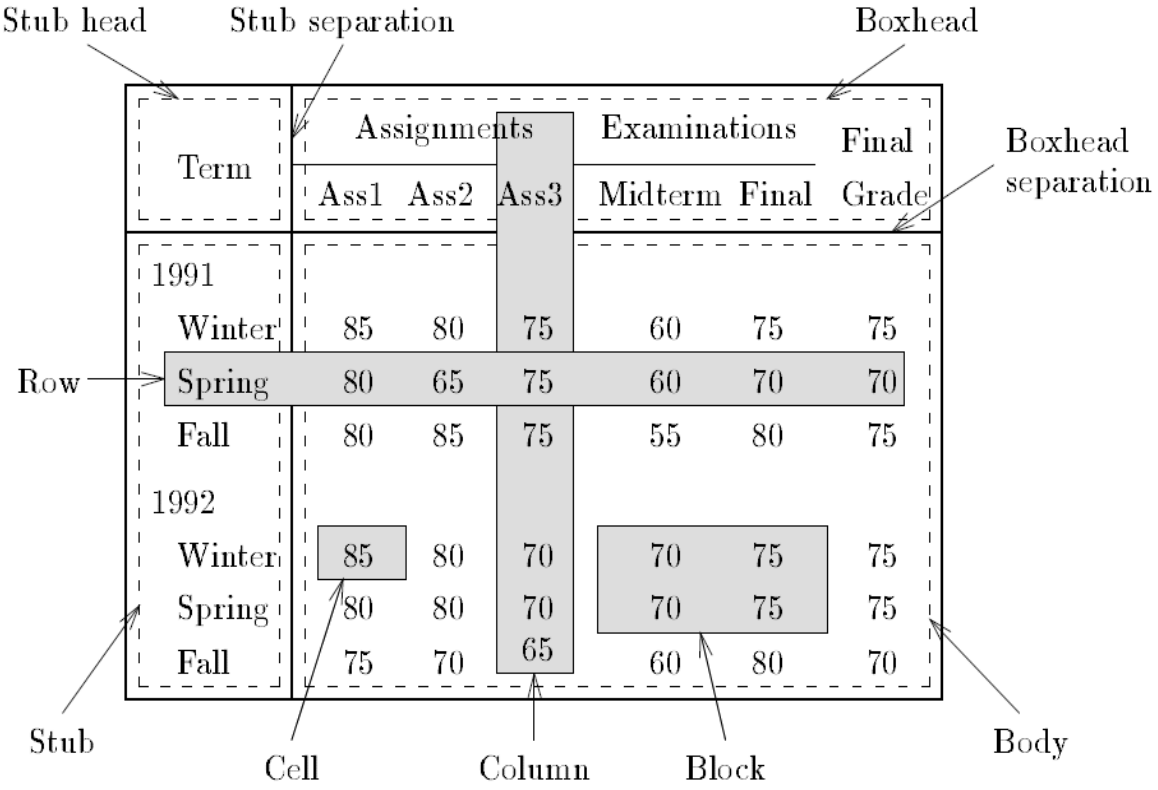


Fig. 9.1. The terminology of row-column structure presentational structure of tables of Xinxin Wang

Because many tables have irregular structures generating by the nesting cells or missing boundary lines. People may have different opinions on the definition of cells. Two definitions are proposed: *virtual cells* and *real cells*. *Virtual cells* refer to the smallest rectangles obtained by extending all the borders between real cells up to the bounding box of the entire tables. *Real cells* are composed of one or several virtual cells and are not necessarily rectangles.

The simplest tables are regular matrices of cells: all the cells of a row have the same height and all the cells of a column have the same width. The borders of all the cells are marked by graphic lines. However, very few tables follow such a strict prescription. Many tables can be much more complex, such as: 1) Regarding the physical layout, we can often notice some over-expanded tables: the presence of cells extending over several rows or columns (see *Figure 9.2 d*); 2) only a few of the borders are marked by graphic lines (see examples in *Figure 9.4*); 3) Not all the cells are filled (*Figure 9.6*); 4) A partitioned table: the table with a multi-dimensional structure can be treated as a combination of several tables (Fig. xxxxx); 5) Huge tables with extremely large width/length; 6) tables with specific cell types, e.g., figures, formulas, etc (please see *Figure 9.2 b*). To better understanding tables, we analyze them from the following perspectives.

9.2.2 Perspectives

Table Location: we define the table location as the first place where authors introduce the table in the document. In order to evaluate all the table locations with

consistent terms, we take advantage of the basic section headings in most scientific documents. Comparing with the IMRAD [1] (Introduction, Materials and Methods, Results, and Discussion) model, we combine "Results" and "Discussion" together and add two additional sections: "Background" and "Conclusion" because some authors may use tables to compare the related works. *Table 9.1* shows examples of typical section headings for each section.

Table 9.1. The typical headings of document sections

Background	Introduction/ Motivation/ Background/ Scenario/Related work/ Related approach/ Previous work/ The state of the art/ ...
Methods	Approach/ Framework/ Model/ System overview/ Methodology/ Concept/ Algorithm/ Proposed method/...
Results and Discussion	Performance evaluation/ Experiments/ Experimental result/ Dataset/ Analysis/...
Conclusion	Summary/Conclusion/ Future work/...

Table Sizes: we classify table size into two categories: *Cross-column* vs. *Single-column* tables. *Figure 9.2* shows several table examples with diverse sizes. A table is a cross-column table if it satisfies both the following two requirements: 1) the width of table is larger than the half of the document width; 2) the table spans at least one space between two document columns (e.g., *Figure 9.2 d*). Otherwise, this table is a *single-column* table. We can also call them as *wide* table and *narrow* table. In this study, we concern more about the table size for better table displaying and representation. If a table is a *cross-column* table with a very large width, we should consider whether we

[1] http://www.infotech.oulu.fi/GraduateSchool/ICourses/2006/phd/structure.pdf

need to modify such a page-width table to a single-column width table for the PDA displaying purpose.

Table Caption Location: According to the location of the table caption, there are two types: *top caption* and *bottom caption.* Although most guidelines suggest authors to put table caption above the table body, there are still many tables that the caption is below the table body. Different tables may have different caption locations even within a same document (see Table 5 and 6 in *Figure 9.3 a*). Such inconsistent caption locations heavily impair the performance of the table boundary detection. In addition, some tables have no caption (see *Figure 9.3 c*) or wrong captions (see *Figure 9.3 e*) at all.

Table Caption Size: A meaningful table caption is very important for future table understanding and semantic analysis. *Figure 9.3* shows several table examples with diverse caption sizes. If a caption is very simple (e.g., "Table 1: Experimental results."), it can not provide readers with any quick idea about the table content. On the other hand, some tables have extremely large captions, which even span more than a whole document page (see *Figure 9.3 d*). We also dislike such large captions because they will bring difficulty to the table caption detection. We have to judge whether it is a real table caption or a regular document paragraph.

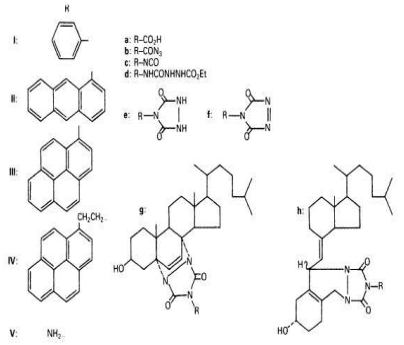
Frames and separator lines: *Figure 9.4* shows the typical examples with different frames and separator lines. From the table structure decomposition perspective, we welcome tables with complete frames and separator lines for each cell (e.g., *Figure 9.4 a*). However, most tables miss a part of the lines. For example, some journals/proceedings have their specific requirements (e.g., most journals in chemistry field only require horizontal lines, see *Figure 9.4 c, d, e*). *Figure 9.4 b* only uses vertical

A	B	C	D	Count
$a1$	$b1$	$*$	$*$	2
$a1$	$*$	$*$	$*$	1
$a2$	$*$	$c3$	$d4$	2

Table 4: **Compressed Base Table**: After star reduction.

(a)

Table 1 Compound structures



(b)

Table 1 Library analysis by LC-MS

Compound No.	X	Y	Z	t_R/min	$M^+ + H$	Compound No.	X	Y	Z	t_R/min	$M^+ + H$
1	Ph	Ph	Me	8.07	271.3	**26**	Ph	Ph	CH_2Ph	8.57	347.4
2	Ph	CH(Me)Ph	Me	10.93	299.4	**27**	Ph	CH(Me)Ph	CH_2Ph	10.85	375.5
3	Ph	$3,5\text{-Me}_2C_6H_3$	Me	6.22	299.4	**28**	Ph	$3,5\text{-Me}_2C_6H_3$	CH_2Ph	6.47	375.5
4	Ph	Allyl	Me	13.59	235.3	**29**	Ph	Allyl	CH_2Ph	13.26	311.4
5	Ph	Cyclohexyl	Me	7.12	277.4	**30**	Ph	Cyclohexyl	CH_2Ph	7.43	353.5
6	2-ClC_6H_4	Ph	Me	14.59	305.8	**31**	2-ClC_6H_4	Ph	CH_2Ph	14.84	381.9
7	2-ClC_6H_4	CH(Me)Ph	Me	18.34	333.8	**32**	2-ClC_6H_4	CH(Me)Ph	CH_2Ph	17.03	409.9
8	2-ClC_6H_4	$3,5\text{-Me}_2C_6H_3$	Me	10.24	333.8	**33**	2-ClC_6H_4	$3,5\text{-Me}_2C_6H_3$	CH_2Ph	10.38	409.9
9	2-ClC_6H_4	Allyl	Me	24.65	269.8	**34**	2-ClC_6H_4	Allyl	CH_2Ph	22.88	345.9
10	2-ClC_6H_4	Cyclohexyl	Me	12.45	311.8	**35**	2-ClC_6H_4	Cyclohexyl	CH_2Ph	11.40	387.9
11	$4\text{-O}_2NC_6H_4$	Ph	Me	15.50	316.3	**36**	$4\text{-O}_2NC_6H_4$	Ph	CH_2Ph	16.49	392.4
12	$4\text{-O}_2NC_6H_4$	CH(Me)Ph	Me	21.11	344.4	**37**	$4\text{-O}_2NC_6H_4$	CH(Me)Ph	CH_2Ph	20.05	420.5
13	$4\text{-O}_2NC_6H_4$	$3,5\text{-Me}_2C_6H_3$	Me	10.17	344.4	**38**	$4\text{-O}_2NC_6H_4$	$3,5\text{-Me}_2C_6H_3$	CH_2Ph	10.43	420.5
14	$4\text{-O}_2NC_6H_4$	Allyl	Me	12.22	280.3	**39**	$4\text{-O}_2NC_6H_4$	Allyl	CH_2Ph	26.41	356.4
15	$4\text{-O}_2NC_6H_4$	Cyclohexyl	Me	10.69	322.4	**40**	$4\text{-O}_2NC_6H_4$	Cyclohexyl	CH_2Ph	11.06	398.5
16	4-MeC_6H_4	Ph	Me	8.65	285.4	**41**	4-MeC_6H_4	Ph	CH_2Ph	9.20	361.5
17	4-MeC_6H_4	CH(Me)Ph	Me	12.36	313.4	**42**	4-MeC_6H_4	CH(Me)Ph	CH_2Ph	11.94	389.5
18	4-MeC_6H_4	$3,5\text{-Me}_2C_6H_3$	Me	6.54	313.4	**43**	4-MeC_6H_4	$3,5\text{-Me}_2C_6H_3$	CH_2Ph	6.87	389.5
19	4-MeC_6H_4	Allyl	Me	15.22	249.3	**44**	4-MeC_6H_4	Allyl	CH_2Ph	14.86	325.4
20	4-MeC_6H_4	Cyclohexyl	Me	7.89	291.4	**45**	4-MeC_6H_4	Cyclohexyl	CH_2Ph	7.95	367.5
21	1-Naphthyl	Ph	Me	9.56	321.4	**46**	1-Naphthyl	Ph	CH_2Ph	9.71	397.5
22	1-Naphthyl	CH(Me)Ph	Me	11.88	349.4	**47**	1-Naphthyl	CH(Me)Ph	CH_2Ph	11.31	425.5
23	1-Naphthyl	$3,5\text{-Me}_2C_6H_3$	Me	7.23	349.4	**48**	1-Naphthyl	$3,5\text{-Me}_2C_6H_3$	CH_2Ph	7.23	425.5
24	1-Naphthyl	Allyl	Me	14.71	285.4	**49**	1-Naphthyl	Allyl	CH_2Ph	13.76	361.5
25	1-Naphthyl	Cyclohexyl	Me	7.78	327.4	**50**	1-Naphthyl	Cyclohexyl	CH_2Ph	7.27	403.5
51	Ph	Ph	Allyl	6.86	297.4	**76**	Ph	Ph	n-Bu	6.10	313.4
52	Ph	CH(Me)Ph	Allyl	8.51	325.4	**77**	Ph	CH(Me)Ph	n-Bu	7.44	341.5
53	Ph	$3,5\text{-Me}_2C_6H_3$	Allyl	5.35	325.4	**78**	Ph	$3,5\text{-Me}_2C_6H_3$	n-Bu	4.89	341.5
54	Ph	Allyl	Allyl	10.44	261.4	**79**	Ph	Allyl	n-Bu	8.96	277.4
55	Ph	Cyclohexyl	Allyl	5.84	303.4	**80**	Ph	Cyclohexyl	n-Bu	5.54	319.5
56	2-ClC_6H_4	Ph	Allyl	11.34	331.8	**81**	2-ClC_6H_4	Ph	n-Bu	9.74	347.9
57	2-ClC_6H_4	CH(Me)Ph	Allyl	12.84	359.9	**82**	2-ClC_6H_4	CH(Me)Ph	n-Bu	11.22	375.9
58	2-ClC_6H_4	$3,5\text{-Me}_2C_6H_3$	Allyl	8.22	359.9	**83**	2-ClC_6H_4	$3,5\text{-Me}_2C_6H_3$	n-Bu	7.21	375.9
59	2-ClC_6H_4	Allyl	Allyl	17.13	295.8	**84**	2-ClC_6H_4	Allyl	n-Bu	14.64	311.8
60	2-ClC_6H_4	Cyclohexyl	Allyl	9.11	337.9	**85**	2-ClC_6H_4	Cyclohexyl	n-Bu	8.24	353.9
61	$4\text{-O}_2NC_6H_4$	Ph	Allyl	12.75	342.4	**86**	$4\text{-O}_2NC_6H_4$	Ph	n-Bu	10.21	358.4
62	$4\text{-O}_2NC_6H_4$	CH(Me)Ph	Allyl	15.15	370.4	**87**	$4\text{-O}_2NC_6H_4$	CH(Me)Ph	n-Bu	11.59	386.5
63	$4\text{-O}_2NC_6H_4$	$3,5\text{-Me}_2C_6H_3$	Allyl	8.38	370.4	**88**	$4\text{-O}_2NC_6H_4$	$3,5\text{-Me}_2C_6H_3$	n-Bu	6.96	386.5
64	$4\text{-O}_2NC_6H_4$	Allyl	Allyl	21.21	306.3	**89**	$4\text{-O}_2NC_6H_4$	Allyl	n-Bu	15.85	322.4
65	$4\text{-O}_2NC_6H_4$	Cyclohexyl	Allyl	9.09	348.4	**90**	$4\text{-O}_2NC_6H_4$	Cyclohexyl	n-Bu	7.03	364.5
66	4-MeC_6H_4	Ph	Allyl	7.40	311.4	**91**	4-MeC_6H_4	Ph	n-Bu	6.58	327.4
67	4-MeC_6H_4	CH(Me)Ph	Allyl	9.39	339.5	**92**	4-MeC_6H_4	CH(Me)Ph	n-Bu	8.27	355.5
68	4-MeC_6H_4	$3,5\text{-Me}_2C_6H_3$	Allyl	5.71	339.5	**93**	4-MeC_6H_4	$3,5\text{-Me}_2C_6H_3$	n-Bu	5.22	355.5
69	4-MeC_6H_4	Allyl	Allyl	11.70	275.4	**94**	4-MeC_6H_4	Allyl	n-Bu	10.11	391.4
70	4-MeC_6H_4	Cyclohexyl	Allyl	6.58	317.4	**95**	4-MeC_6H_4	Cyclohexyl	n-Bu	6.15	333.5
71	1-Naphthyl	Ph	Allyl	7.50	347.4	**96**	1-Naphthyl	Ph	n-Bu	6.48	363.5
72	1-Naphthyl	CH(Me)Ph	Allyl	8.71	375.5	**97**	1-Naphthyl	CH(Me)Ph	n-Bu	7.33	391.5
73	1-Naphthyl	$3,5\text{-Me}_2C_6H_3$	Allyl	5.86	375.5	**98**	1-Naphthyl	$3,5\text{-Me}_2C_6H_3$	n-Bu	5.11	391.5
74	1-Naphthyl	Allyl	Allyl	10.56	311.4	**99**	1-Naphthyl	Allyl	n-Bu	8.68	327.4
75	1-Naphthyl	Cyclohexyl	Allyl	6.38	353.5	**100**	1-Naphthyl	Cyclohexyl	n-Bu	5.41	369.5

(c)

membrane NCAMs lend themselves to facile storage and regeneration.

0.118 M Na_2SO_3, 0.68 M 37% formaldehyde in water, and 0.007 M Oromerse gold solution, part B. The membranes were

Table 1 Names and sequences of modified oligonucleotides used

HS-(7)17E-Dy	5′-(C_6Thiol)-TTTTTAAAGAGACATCTCTTCTCCGAGCCGGTCGAAATAGTGAGT-Dabcyl-3′
Dy-(7)17DS-Fl	5′-Fluorescein-ACTCACTATrAGGAAGAGATGTCTCTTT-Dabcyl-3′
(7)17DSnc-Cy5	5′-Cy5-ACTCACTATAGGAAGAGATGTCTCTTT-3′
(7)17DSnc-Alexa546	5′-Alexa546-ACTCACTATAGGAAGAGATGTCTCTTT-3′

(d)

Table 2 Conditions for the *in situ* derivatization of different alkyl halide samples

Primary alkyl iodide	1-Iodobutane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 90 min
Primary alkyl bromide	1-Bromobutane 1-Bromopropane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 150 min
Primary alkyl chloride Secondary alkyl bromide	1-Chlorobutane 2-Bromobutane	Reflux with 3 equiv. of NaI in 40 ml of 95% ethanol for 24 h, then reflux with excess of thiourea for 150 min

(e)

1: DT determiners	2: NP proper nouns	3: V verbs	4: N nouns
5: I prepositions	6: G genitive	7: J adjectives	8: PU punctuation
9: R adverbs	10: RQ questioners	11: C conjunctions	12: TO infinitive mrk
13: P pronoun	14: M numbers	15: VM modal verbs	16: PDT pronom. DT
17: EX existentials	18: LS lists	19: UH interjection	20: FO foreign words

Table 3: Identifiers and ordering of the "perfect" clusters.

(f)

Fig. 9.2. Table examples with different sizes

Policy	Complexity
Processing Farm	$O(n)$
Job Splitting	$O(n^2)$
Out of order	$O(n\sqrt{n})$
Delayed Scheduling	$O(n)$

Table 5: Complexities for scheduling one job according to the different scheduling policies in function of the number n of nodes in the cluster

Table 5 gives the complexity of scheduling one job as a function of the number n of nodes in the cluster. However, in a real case, the number of jobs processed per unit of time is also dependant on the number of nodes, typically proportional to it. Taking this fact into account, theoretical complexities have to be multiplied by n for a comparison with simulation data. Figure 16 shows the average load of the master node as a function of the number of slave nodes under the following conditions :

1. The cluster load is proportional to the number of slave nodes present in the cluster
2. The total data space size is proportional to the number of slave nodes

and their position is derived from the simulation data.

Simulations and theory match well for the job splitting and out of order policies. In the case of the delayed policy, the load represents both the subdivision of data into the desired stripe size and job splitting. Thus an extra curve of simulated data shows the time spent only in the job splitting part of the delayed scheduling algorithm. It clearly shows that the complexity of the job splitting part matches the theoretical complexity. However, the simulations show that the job splitting part becomes preponderant only for a large number of nodes, around 50000.

From the theoretical extrapolations of the simulated data, the expected maximum number of nodes that a single master node can handle is around 5000 for the job splitting policy, around 20000 for the out of order scheduling policy and around 300000 for the delayed scheduling policy. The period delay introduced by the delayed scheduling policy reduces the job scheduling load for very large clusters by an order of magnitude.

8.2 Space complexity

The space complexity is bound by the amount of data needed by the master node to schedule the jobs, namely the list of nodes and the status of their disk caches (when applicable). The required memory space is linear with the number of nodes and remains relatively small :

12

Table 6: Single job theoretical time complexities of the different scheduling policies in function of the number n of nodes in the cluster.

Processing Farm	Job Splitting
Schedling one job requires traversing the list of nodes to find an idle node. Scheduling complexity is $O(n)$.	Upon job arrival, the worst case requires scanning all nodes to find the subjob to be suspended. Complexity of this case is $O(n)$.
Our Of order	Upon job or subjob end, the worst case consists in finding the subjob to split. By looking at all of them (one per node) the complexity is $O(n)$ per split.
Upon job arrival, the new job has to be split into subjobs. Splitting takes caches into account by reading the content description of each node cache, yielding a complexity of $O(n)$. When a node becomes available, a subjob needs to be split. By looking at all of them (one per node), the complexity is $O(n)$ per split. The global complexity per job is thus of the order of $O(n)$ multiplied with the average number of splits per job, i.e. the average number of subjobs. Let $j(t)$ be the number of jobs running in the cluster and $s(t)$ the average number of subjobs per job at time t. If the cluster is not idle, the job splitting algorithm ensures that all nodes have a running subjob. Thus : $\forall t > 0 \;\; j(t)\, s(t) \simeq n$ The simulation shows that $s(t) = O(\sqrt{n})$ and $j(t) = O(\sqrt{n})$. Therefore the scheduling complexity is $O(n\sqrt{n})$.	With a same hara that the number of jobs in the system depends only on the load, and is independant of the number of nodes n. Since the jobs are parallelized across all nodes, the number of splits per job is $O(n)$. Since the number of splits per job is $O(n)$ and each split operation is $O(n)$, the overall complexity of single job splitting is $O(n^2)$.
	Delayed Scheduling
	At each new period, the new jobs are split into subjobs. Job splitting takes caches into account by reading the content description of each node cache, yielding a complexity of $O(n)$. The non cached subjobs are further split. The number of splits per job is given by the stripe size and is independent of the number of nodes yielding a complexity of $O(1)$. The overall complexity per job is thus $O(n)$.

(a)

Table 10
Summary of cross-sectional regression results of absolute effective bid/ask spreads of Nasdaq stocks. The notation is defined as follows. $TWES_i$ is the volume-weighted effective spread, $ProTS_i$ is the inverse of the number of shares traded, MM is the modified Hasbrouck index, and IHP_i is the expected inventory-holding premium. The value of each variable, except IHP_i and MM_i, is computed each trading day and then the values are averaged across all days during the month. To be included in the sample, the stock must have traded at least five times each day in every day during the month. For March 1998 (April 1998), only stocks with price quotes in eighths (sixteenths) are included. The first set of regression results in the panel on the left is for the regression specification,

$$TWES_i = \alpha_i + \alpha_1 ProTS_i + \alpha_2 MM_i + \alpha_3 IHP_i + \epsilon_i .$$

The value of IHP_i is computed as an at-the-money option

$$IHP_i = S_i \left[2N\left(.5\sigma_i \sqrt{T_i} \right) - 1 \right],$$

where S_i is the average share price, σ_i is the annualized return volatility of the stock computed over the most recent 60 trading days prior to the estimation month, and $\sqrt{T_i}$ is the average of the square root of the time between trades. In the second regression we constrain the coefficient α_1 to equal one and estimate the average of the square root of the time between trades for a single dealer. The regression results reported in the right panel are for the regression specification,

$$TWES_i = \alpha_i + \alpha_1 ProTS_i + \alpha_2 MM_i + \alpha_3 IHP_{i,+}(\sigma_i) + \alpha_4 (IHP_{i,+}(\sigma_i) - IHP_{i,-}(\sigma_i)) + \epsilon_i .$$

where $IHP_{i,+}$ is the expected inventory-holding premium for trades with uninformed traders and $IHP_{i,-}$ is the expected inventory-holding premium for trades with informed traders. For a trade at the ask, the value of $IHP_{i,+}$ is computed using

$$IHP_{i,+} = S_i \cdot N\left[\frac{\ln(S_i \sqrt{N_i})}{\sigma_i \sqrt{T_i}} + .5\sigma_i \sqrt{T_i} \right] - N \cdot N\left[\frac{\ln(S_i \sqrt{N_i})}{\sigma_i \sqrt{T_i}} - .5\sigma_i \sqrt{T_i} \right].$$

$IHP_{i,+}$ is valued as an out-of-the-money call option with an exercise price equal to the ask price and a stock price equal to the bid/ask midpoint. $IHP_{i,-}$ is valued as an in-the-money (ITM) call option with an exercise price equal to the ask price and a stock price Percent ITM above the exercise price. For a trade at the bid, the IHP is valued using a put option formula with an exercise price equal to the bid price. All coefficient α_3 are significantly different from zero at the five percent probability level, except for those in the December 2001 subsample. None of the coefficients α_4 in the right are significantly different from zero except those in the April 1998 subsample.

49

(d)

Table 3 shows the image compositing results.

	im-off	xr-on	xr-off	gl-on	gl-off
over	3.809	3.870	3.850	0.109	0.107
scale	16.444	85.504	86.924	0.126	0.132
blend	4.349	73.222	69.613	0.264	0.263
blur	26.499	-	-	3.089	3.078

Table 3: *Seconds to complete composite test (lower is better)*

(b)

1. Polynomials with real random coefficients, of degrees 10000 and 100000.
2. Polynomials with complex random coefficients, of degrees 10000, 100000 and 1000000.
3. Exponential polynomials $\sum_i \frac{1}{i!}$, for some degrees of 100, 150, 200.

	d	x_0	k	T
double	10000	0.0	8	22.03s
double	100000	0.0	8	346.52s
complex	10000	(0.4,0.6)	6	16.86s
complex	100000	(0.4,0.6)	6	292.02s
complex	1000000	(0.4,0.6)	7	2520s
exp	100	(0.0,0.8)	6	45s
exp	150	(0.0,0.8)	6	59s
exp	200	(0.0,0.8)	6	101s

where d is the degree of the polynomial, x_0 the starting value for the iterative process, k is the number of iterations to compute one root and T is the total time of the computation. These experimentations have been performed on a Dec workstation with 496M of local memory.

(c)

Figure 1. A summary of the different data sources used during the study

	Site 1 Alvik Airport	Site 2 Arlanda Airport	Site 2 Gardermoen Airport	Off site
Participant observation				
Quantity (days)	3	2	2	—
Length (total, in hours)	32	6	14	—
Interviews				
Quantity	2	4	3	—
Length (total, in hours)	3	6	5	—
Documentation review				
Extensive reviewing	Yes	Yes	—	Yes
Minor reviewing	—	—	Yes	—
Project meetings				
Quantity	1	2	—	—
Length (hours, total)	1	10	—	—

(e)

Fig. 9.3. Table examples with different captions

lines to separate columns. *Figure 9.4 i* only has lines outside the table boundary as well as *Figure 9.4 f* only has lines within the table boundary. *Figure 9.4 g* does not have any frame and separator line at all.

Cell Alignments: this attribute specifies the alignment of data and the justification of text in a cell. it includes *horizontal alignment* and *vertical alignment*. For the table structure decomposition goal, *horizontal alignment* is much more important than *vertical alignment*. *Figure 9.5* displays several examples.

For the horizontal alignment, the possible values are: *left* (Left-flush data/Left-justify text, See *Figure 9.5 a*), *center* (Center data/Center-justify text, such as *Figure 9.5 d*), *right* (Right-flush data/Right-justify text, see the last column in *Figure 9.5 d*), *justify* (double-justify text), and *character* (align text around a specific character, e.g., the dot character of numerical cells or the operators of the mathematical formula, see the numerical cells in *Figure 9.5 b*). *Left* is the default value for table data and *right* is the default value for table headers.

Vertical alignment specifies the vertical position of the data within a cell. The typical values are: *top*(cell data is flush with the top of the cell), *middle* (cell data is centered vertically within the cell), *bottom* (cell data is flush with the bottom of the cell), etc. *Middle* is the default value.

Cell Types: Table cells can record different objects. From the structure view, table cells can be divided into *atomic* cells and *composite* cells. A *composite* cell is composed by several *atomic* cells. We call it *cross-column* or *cross-row* cells (please see examples in *Figure 9.7*). Besides those typical objects, such as texts and numerical data, some specific object types may also appear in table cells. For example, in the

1: DT determiners	2: NP proper nouns	3: V verbs	4: N nouns
5: I prepositions	6: G genitive	7: J adjectives	8: PU punctuation
9: R adverbs	10: RQ questioners	11: C conjunctions	12: TO infinitive mrk
13: P pronoun	14: M numbers	15: VM modal verbs	16: PDT pronom. DT
17: EX existentials	18: LS lists	19: UH interjection	20: FO foreign words

Table 3: Identifiers and ordering of the "perfect" clusters.

(a)

Table 1: Mean Collection Statistics Across All Indices

	num docs	num ents	max(nidf)	min(nidf)	words	phrases
Max	1795443	400094218	14.4	0.79	6924754	1816753
Min	1631840	377033932	14.31	0.7	5820360	1479404
Mean	1743419	385991502	14.37	0.74	6348646	1633945
Max-Min	163603	23060286	0.1	0.09	1104394	337349
Std Dev	43443	7832298	0	0	340602	112354

(b)

Table 3 SEC-DF-ICP-MS (μg g^{-1} wet tissue[a]) quantitative multi-elemental fractionation of metals in different pools of cytosolic fractions of mussel samples (in three different coastal regions)

DF-ICP-MS resolving power 3000—

Villaviciosa estuary

Pool	Al	Ca	V	Cr	Mn	Fe	Co	Ni	Zn
HMW[b]	0.18 ± 0.01	12.5 ± 0.5	0.22 ± 0.03	0.12 ± 0.01	0.13 ± 0.01	115 ± 3	0.02 ± 0.01	0.42 ± 0.02	14.0 ± 0.6
MMW[c]	0.11 ± 0.01	11.2 ± 0.8	0.18 ± 0.03	0.091 ± 0.004	0.070 ± 0.003	41 ± 1	0.010 ± 0.001	0.14 ± 0.01	7.0 ± 0.6
LMW[d]	0.30 ± 0.01	341 ± 22	0.57 ± 0.01	0.35 ± 0.02	0.19 ± 0.05	161 ± 14	0.072 ± 0.003	0.29 ± 0.02	20.1 ± 0.2
VLMW[e]	0.21 ± 0.01	285 ± 9	0.19 ± 0.01	0.093 ± 0.003	0.080 ± 0.005	46 ± 3	0.011 ± 0.001	0.21 ± 0.01	1.12 ± 0.05
Cytos[f]	0.86 ± 0.03	625 ± 14	1.20 ± 0.02	0.68 ± 0.03	1.50 ± 0.05	371 ± 17	0.121 ± 0.005	1.10 ± 0.05	43 ± 1

Galicia coast

Pool	Al	Ca	V	Cr	Mn	Fe	Co	Ni	Zn
HMW	0.088 ± 0.002	6.6 ± 0.3	0.25 ± 0.01	0.15 ± 0.01	0.16 ± 0.01	231 ± 6	0.041 ± 0.002	1.1 ± 0.04	24.0 ± 0.2
MMW	0.060 ± 0.001	7.3 ± 0.2	0.090 ± 0.005	0.076 ± 0.004	0.078 ± 0.003	55 ± 3	0.010 ± 0.001	0.29 ± 0.01	12.0 ± 0.5
LMW	0.26 ± 0.01	437 ± 16	0.48 ± 0.01	0.38 ± 0.02	1.06 ± 0.02	254 ± 11	0.046 ± 0.002	0.47 ± 0.02	16.0 ± 0.5
VLMW	0.57 ± 0.03	268 ± 10	0.23 ± 0.01	0.09 ± 0.003	0.080 ± 0.005	28 ± 1	0.011 ± 0.001	0.23 ± 0.01	3.1 ± 0.1
Cytos	1.11 ± 0.04	664 ± 22	1.11 ± 0.03	0.72 ± 0.02	1.40 ± 0.04	573 ± 20	0.100 ± 0.005	2.1 ± 0.04	56 ± 1

(c)

Table 1 Different effects of 1.00 × 10^{-3} mol dm^{-3} cysteine on hydrogen peroxide–luminol–cobalt(II) chemiluminescence in borate and carbonate buffers

Buffer	pH	[Luminol]/ mol dm^{-3}	[H_2O_2]/ mol dm^{-3}	[Co^{2+}]/ mol dm^{-3}	Relative signal	SEM[a]	*n*
Borate	10.2	9.9 × 10^{-6}	1.0 × 10^{-4}	1.00 × 10^{-5}	0.0296	0.0027	5
Carbonate	10.1	1.00 × 10^{-4}	1.00 × 10^{-6}	1.00 × 10^{-5}	2.424	0.079	5

[a] SEM, standard error of the mean.

Table 1 ICP-MS operating conditions and measurement parameters

Rf power/W	1050
Argon flow rate/l min^{-1}:	
outer	15.0
intermediate	0.8
nebulizer	1.0
Dwell time/ms	50
Scanning mode	Peak hop
Sweeps per reading	25
Replicates	10
Points per spectral peak	1
Specific Elan 5000 lens settings	P 50, B 49, S2 40 and E1 20
Ions monitored	^{127}I, ^{129}I, ^{131}Xe
Delay time/s	90
Wash time/s	300

(e)

Table 2 Effects of cysteine and cystine on cobalt-catalysed chemiluminescence of hydrogen peroxide (1.00 × 10^{-6} mol dm^{-3}) and luminol in 0.100 mol dm^{-3} carbonate buffer, pH 10.4

Sulfide	[Sulfide]/ mol dm^{-3}	[Luminol]/ mol dm^{-3}	[H_2O_2]/ mol dm^{-3}	[Co^{2+}]/ mol dm^{-3}	Relative signal	SEM	*n*
Cysteine	9.8 × 10^{-4}	1.15 × 10^{-4}	1.00 × 10^{-6}	1.00 × 10^{-4}	9.048	0.144	5
Cysteine	9.8 × 10^{-5}	1.15 × 10^{-4}	1.00 × 10^{-6}	1.00 × 10^{-4}	2.628	0.137	6
Cystine	5.0 × 10^{-4}	1.00 × 10^{-4}	1.00 × 10^{-6}	1.00 × 10^{-5}	3.596	0.168	7
Cystine	5.2 × 10^{-4}	1.00 × 10^{-4}	1.00 × 10^{-6}	1.00 × 10^{-5}	0.564	0.012	5

(d)

Problem	Divide & Conquer	Binary Search	lp_solve
stein9	0.02 s	0.03 s	0.01 s
stein15	0.07 s	0.04 s	0.02 s
stein27	2.62 s	0.41 s	4.59 s
stein45	1290.90 s	127.85 s	355.03 s
p0033	0.31 s	758.97 s	0.32 s
p0040	0.10 s	—	0.03 s
bm23	14.19 s	30.18 s	0.09 s

Table 1: Run time comparison for ILP+BDD approach

(f)

Table 2 Conditions for the *in situ* derivatization of different alkyl halide samples

Primary alkyl iodide	1-Iodobutane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 90 min
Primary alkyl bromide	1-Bromobutane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 150 min
	1-Bromopropane	
Primary alkyl chloride	1-Chlorobutane	Reflux with 3 equiv. of NaI in 40 ml of 95% ethanol for 24 h, then reflux with excess of thiourea for 150 min
Secondary alkyl bromide	2-Bromobutane	

(g)

Table 1: Major Inputs of Nitrogen and Phosphorus to the Potomac River Basin, 1990.

Source of Nutrient Pollution	Percentage of Nutrient Loading (%)	
	Nitrogen	Phosphorus
Atmospheric Deposition	32	<1
Municipal and Industrial Wastewater Discharges	12	4
Septic Systems	< 2	3
Animal Manure	29	45
Commercial Fertilizer	26	48

Source: Ator *et al.*, 1998

(h)

# of patterns	egrep	fgrep	GNU-grep	gre	agrep
10	6.54	13.57	2.83	5.66	2.22
50	8.22	12.95	5.63	9.67	2.93
100	16.69	13.27	6.69	11.88	3.31
200	42.62	13.51	8.12	14.38	3.87
1000	-	-	12.18	23.14	5.79
2000	-	-	15.80	28.36	7.44
5000	-	-	21.82	38.09	11.61

Table 1: A comparison of different search routines on a 15.8MB text.

(i)

Fig. 9.4. Table examples with different boundary lines

Table 5 Nitrogen detection limits for various materials, irradiated in vacuum, and counted for 24 h. Detection limits were calculated using the nitrogen 10 MeV gamma-ray peaks and equations given by Currie

Matrix	Detectable N mass fraction (%)
Biological (SRM 1573a, tomato leaves, 300 mg sample)	0.13
Soil (SRM 2710, Montana soil, 500 mg sample)	0.05
Rock (USGS GSP-1, 500 mg sample)	0.12
Titanium (SRM 354, unalloyed, 200 mg)	0.7

(a)

Original Arabic Term	First Match Method
مصباح	light
ضوئي	brightness
وهـّـاج	glowing

Table 2. Terms of the original Arabic query, and the result of the First-Match

For English-Arabic CLIR, Table 3 illustrates an example of the Arabic terms retained by the FM method.

Original English query	First Match Method
Information	اعلام
Technology	التكنلوجيا
Arab	العربي
World	الدنيا

Table 3. English query terms and their translation using FM method

(c)

Table 1 Calibration equations (results from least squares regression), limit of detection (LOD), limit of quantification (LQD) standard deviation (SD) and relative standard deviation (RSD) for levofloxacin determination by luminescence and fluorescence methods

	Luminescence method	Fluorescence method
Concentration range/μg mL^{-1}	0.008–0.600	0.020 –0.300
Intercept on the ordinate (a)	1.3	1.6
SD of the intercept on ordinate (s_a)	0.3	0.2
Slope (b)	386	204
SD of slope (s_b)	1	2
Number of points (n)	30	27
Coefficient of correlation (r)	0.999	0.999
LOD/μg mL^{-1}	10	10
LOQ/μg mL^{-1}	30	30
RSD (%) (n = 10)	1.1	1.2

(b)

Table 3. Lists of the words most predictive of the course class in the WebKB data set, as they change over iterations of EM for a specific trial. By the second iteration of EM, many common course related words appear. The symbol D indicates an arbitrary digit.

Iteration 0	Iteration 1	Iteration 2
intelligence	*DD*	*D*
DD	*D*	*DD*
artificial	lecture	lecture
understanding	cc	cc
*DD*w	*D**	*DD*:*DD*
dist	*DD*:*DD*	due
identical	handout	*D**
rus	due	homework
arrange	problem	assignment
games	set	handout
dartmouth	tay	set
natural	*DD*am	hw
cognitive	yurttas	exam
logic	homework	problem
proving	kfoury	*DD*am
prolog	sec	postscript
knowledge	postscript	solution
human	exam	quiz
representation	solution	chapter
field	assaf	ascii

(d)

Fig. 9.5. Table examples with different aligned cells

statistics/mathematics papers, the table cells can be percentages or mathematical equations. In chemistry tables, the objects can be chemical formulas or molecular structures. It is not surprising to see images cells, in art or image-processing documents. Sometimes, people just leave cells blank if no value is applicable. Please see examples in *Figure 9.6*.

Although we process the tables in scientific PDF documents and almost all of them are *genuine* tables according to the previous definition, a part of them are not *well-structured* tables from the layout/structure perspective. We classify tables in the structure level. We process more than two million scientific documents in digital libraries from several fields, such as computer science, chemistry, archeology, etc. We extracted approximately one million PDF tables from the English documents in three fields: CiteSeer repository in computer science field, RSC repository in chemistry, and archeology. Based on our observation, we concluded that most *well-defined* tables are similar to each other as well as different *poorly-defined* tables have different problems. Classifying and labeling tables as *well-structured* or *poorly-structured* tables has the following advantages:

1) Detecting the *poorly-designed* tables is beneficial to later improvement for table analysis by changing the content, the topological arrangement, or the typographic style of the table.

2) we can remove unnecessary information and use shorter texts to make a large table smaller, or replace abbreviations with their complete forms to make a narrow column wider, to transpose a fat and short table or remove some texts from row headers to the column headers.

Table 1. Number of participants per track and total number of distinct participants in each TREC

Track	TREC										
	1992	1993	1994	1995	1996	1997	1998	1999	2000	2001	2002
	TREC-1	T-2	T-3	T-4	T-5	T-6	T-7	T-8	T-9	T-2001	T-2002
Ad Hoc	18	24	26	23	28	31	42	41	—	—	—
Routing	16	25	25	15	16	21	—	—	—	—	—
Interactive	—	—	3	11	2	9	8	7	6	6	6
Spanish	—	—	4	10	7	—	—	—	—	—	—
Confusion	—	—	—	4	5	—	—	—	—	—	—
Database Merging	—	—	—	3	3	—	—	—	—	—	—
Filtering	—	—	—	4	7	10	12	14	15	19	21
Chinese	—	—	—	—	9	12	—	—	—	—	—
NLP	—	—	—	—	4	2	—	—	—	—	—
Speech	—	—	—	—	—	13	10	10	3	—	—
Cross-Language	—	—	—	—	—	13	9	13	16	10	9
High Precision	—	—	—	—	—	5	4	—	—	—	—
Very Large Corpus	—	—	—	—	—	—	7	6	—	—	—
Query	—	—	—	—	—	—	2	5	6	—	—
Question Answering	—	—	—	—	—	—	—	20	28	36	34
Web	—	—	—	—	—	—	—	17	23	30	23
Video	—	—	—	—	—	—	—	—	—	12	19
Novelty	—	—	—	—	—	—	—	—	—	—	13
Total participants	22	31	33	36	38	51	56	66	69	87	93

(a)

Table 4

Overview of the results (1): traces against properties

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
GP1	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
GP2	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
GP3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
GP4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
GP5	X	-	-	-	-	-	-	-	-	-	-	-	-	X	-
GP6	X	X	X	X	-	-	X	X	X	X	X	X	X	X	X
GP7	X	X	X	X	X	-	X	X	X	X	X	X	X	X	-
GP8	X	X	-	X	X	X	X	X	X	-	X	X	X	X	X
GP9	-	-	-	-	-	X	X	X	X	-	X	X	X	-	-
GP10	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
LP3	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
LP4	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
LP5	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
LP2	X	X	X	X	X	-	X	X	X	-	X	X	X	X	-
LP1'	X	X	-	X	X	-	X	X	X	-	X	X	X	X	-
steps	1	2	3	3	3	3	3	2	3	3	2	3	2	1	3

(b)

le 6 Comparison of preconcentration factors

Immobilized ligand	Metal ion					
	Cd	Co	Cu	Fe	Ni	Zn
Support: Amberlite XAD-2						
Thiosalicylic acid	200	180	200	400	200	200
Alizarin Red S[6]	40				40	40
Salicylic acid[7]						180
Pyrocatechol violet[8]	50				18	60
o-Aminophenol[8]	50	100	50		65	40
Calmagite[12]			50			
Chromotropic acid[14]	100	150	100	120	200	200
Tiron[15]	48	56	200	80	150	180
o-Vanillinthiosemicarbazone[18]			90			140
1-(2-Pyridylazo)-2-naphthol[24]					50	
Support: Amberlite XAD-7						
Xylenol Orange[16]	50	100	50	200	100	100
Dimethylglyoxal bis(4-phenyl-3-thiosemicarbazone) (DMBS)[21]	100		100			
8-(Benzenesulfonamido)quinoline[20]	10					10
Support: Silica gel						
Salicyldoxime[28]		40	40	40	40	40
3-Hydroxy-2-methyl-1,4-naphthoquinone[30]		10	10	10		10
3-Methyl-1-phenyl-4-stearoyl-5-pyrazolone[31]		40	40		40	
Support: activated carbon						
8-Hydroxyquinoline[33]	100					
Cupferron[33]	100					
Support: Polyurethane foam						
2-(2-Benzothiozolylazo)-*p*-cresol[36]		2.5				
Support: Polyacrylonitrile						
Aminophosphonic and dithiocarbamate[37]	200	200	200			

(c)

Table 1: Relational Table for Evaluation. A_j denotes the jth attribute and the "?" denotes an unknown value, a piece of confidential datum, or a previously modified value.

key	A1	A2	..	A_k	..	A_M	class_label
training data	.. ? ..	?	.. .. ..	.. ? ..	.. .. ..	.. ? ..	.. .. ..
testing	.. ..	.. ?	.. ..	? ..	.. ..	.. ..	? ?

(d)

	Public UMTS	CPN Class A	CPN Class B	CPN Class C	CPN Class D
Bearer Control	√	√	√	√	√
Call Control	√		√	√	√
Mobility Control	√			√	√
Data Management	√				√
Example	*Any public UMTS operator*	*DCPN, Simple MCPN*	*Small BCPN, MCPN*	*BCPN*	*Large BCPN*

Table 1: Classification of UMTS operators according to their functionality

(e)

Fig. 9.6. Table examples with missing cells

Table 2 MIC concentrations in workplace air during manufacture of mineral wool insulation material

Measurement site	MIC concentration, mean ± *s* (RSD in parentheses)[a]/mg m^{-3}		
	DBA impinger[b]	2MP impinger[c]	2MP filter[c]
High-concentration site	0.086 ± 0.023 (27%)	0.068 ± 0.021 (31%)	0.069 ± 0.023 (33%)
Low-concentration site	0.0025 ± 0.0003 (12%)	0.0020 ± 0.0003 (15%)	Below detection limit

[a] Mean value of six parallel samples. [b] LC-MS. [c] LC-UV.

(a)

PARAMETER	DESCRIPTION	VALUE (for constants)	UNITS
m_{lox}	mass of LOX		lbm
m_{gox}	mass of GOX		lbm
T_{lox}	temperature of LOX		deg R
T_{lox}	temperature of LOX		deg R
u_{gox}	internal energy of GOX		BTU/lbm
$\dot{B}$	boil rate		lbm/s
r_{vent}	venting mass rate		lbm/s
q_{gox}	heat dissipation of GOX		BTU/s
q_{lox}	heat dissipation of LOX		BTU/s
q_{total}	total heat dissipation		BTU/s
h_v	heat of vaporization of LOX	91.5	BTU/lbm
C_p	specific heat of LOX	0.4	BTU/deg R
h_{gasin}	input enthalpy of GOX		BTU/lbm
h_{gasout}	output enthalpy of GOX		BTU/lbm
P_{lox}	pressure of LOX		psi
P_{gox}	pressure of GOX		psi
P_{sat}	saturation pressure of LOX		psi
w_{gox}	molecular weight of GOX	0.0708	lbm/mol
V_{lox}	volume of LOX		ft^3
V_{gox}	volume of GOX		ft^3
V_{tank}	volume of LOX tank	16800	ft^3
d_{lox}	density of LOX	71.5	lbm/ft^3
d_{gox}	density of GOX		lbm/ft^3
m_{PVT}	mass of He in PVT		lbm
T_{PVT}	temperature of He in PVT		deg R
P_{PVT}	pressure of PVT		psi
r_{in}	PVT input mass rate		lbm/s
h_{PVT}	input enthalpy of He in PVT		BTU/lbm
V_{PVT}	volume of PVT	0.04	ft^3
d	diameter of the pneumatic valve	0.065	ft
α	loss coefficient	2	
w_{he}	molecular weight of He	0.00882	lbm/mol
R	gas constant	2.365×10^{-2}	$\frac{\text{psi ft}^3}{\text{mol mol}}$

Table 1. Nomenclature

(b)

Table 4 Effect of the sample composition and the carrier used on the relative standard deviation (RSD) of the signal obtained with the SBHPPN and the PCN. Q_e = 0.9 l min^{-1}; Q_l = 1.2 ml min^{-1}; 10 µg g^{-1} Mn

Oil content (% w/w)	RSD (%)			
	Methanol		IBMK	
	SBHPPN	PCN	SBHPPN	PCN
10	1.8	2	0.9	0.6
20	2	4	1.5	2
50	6	14	2	3
60	7	30	1.94	4
70	13		1.70	5
80			3	
100			3	

(c)

TABLE II
MEAN EFFORT COST COMPARISON TABLE

Approach	CPU time[1]	$P_{threshold}$	α	β	Mean Effort Cost[1]
BP	93.58	0.7	5	5	205.87
		0.75	2	8	514.69
		0.8	1	9	1029.38
GA	374.45	0.7	9	1	457.65
		0.75	9	1	457.65
		0.8	8	2	514.86

(d)

Table 1: Comparison of Disruptive Computing

Computing Trend	Disruption *(detaching from)*	Benefits		Challenges	
distributed	decentralization of computing and data *(a single computer)*	remote access replication for high availability redundancy for failover	Increased deployment, accessibility and usability	remote nodes dependency distributed state end-to-end argument	heterogeneity, interoperability, security, trust, transparency, scalability, etc.
mobile	attaching data and computing to user *(fixed computer/network)*	mobile information access mobile computing access ad-hoc networking		disconnection battery lifetime context awareness	
pervasive	user- v. technology-centric focus *(technology)*	broad availability computing invisibility context-adaptivity		personalization scalability in locale user interfaces	
peer-to-peer	peer-provided service/ computing/data *(fixed service providers)*	shared cost-of-ownership ad-hoc component availability self-organization		reputation release of control fragmentation of peer base	

(e)

Fig. 9.7. Table examples with nesting cells

Table 1 Main dimensions of the nebulizers and FAAS experimental conditions used

Nebulizers:			
PCN	Gas outlet section area: 31.1×10^{-3} mm^2		
SBHPPN	Outlet section area: 6.6×10^{-3} mm^2		
FAAS operating conditions:			
Nebulizer gas (air) flow/l min^{-1}	0.9		
Sample uptake rate/ml min^{-1}	1.2		
Gas flow rates/l min^{-1}:			
Acetylene	1.5		
Air	12.0		
Viewing height /mm	15		
Total measurement time/s	80		
Wavelengths, slit-widths and lamp intensities:			
Element	λ/nm	Slit-width/nm	Intensity/mA
Cr	357.9	0.2	4
Cu	324.8	0.5	4
Fe	248.3	0.2	4
Mn	279.5	0.2	4
Ni	232.0	0.2	4
Pb	217.0	1.0	5

(a)

Operator Identity	Frequency Identity	Priority Number	Access Flag (Yes/No)

Table 3: Information stored in an entry of the AOL

(c)

high-level control and learning (mission planning, scheduling)
human interaction modules ("emotional" FSA, Web interface)
navigation modules (localization, map learning, path planning)
hardware interface modules (motors, sensors, Internet)

Table 1: Minerva's layered software architecture

(d)

Arabic query	Translated English Query
الوثائق التي تذكر اسم الشركة أو المجموعة التي تُنتج مُعدّات التشفير فقط ، و لكن لا تذكر بالمصدّر و / أو الاستثمار التجاريّ لمُعدّات التشفير ليست ذات صلة. الوثائق التي تشير إلى الوصول الحكوميّ لأنظم التشفير لغايات ضدّ نشاطات الاستخبارات أو نشاطات مكافحة الجريمة ، تكون ذات صلة.	The documentations which she remembers the name of the company or the group which produces encryption equipments of the encryption only , but not you remember by exported and / or the commercial investment equipments of the encryption have no relevancy. The documentations which she points out the governmental arrival for organisms of the encryption the purposes of a briskness opposite have the secret services or the briskness of the crime struggle , you are a relevancy.

Table 15. The narrative of the original Arabic and the translated English query using the ALKAFI MT system

(b)

Table 1: The job splitting scheduling policy

Upon *job arrival*

- If some nodes are idle, the new job is split into subjobs of equal sizes, one per idle node. All subjobs are launched in parallel. To avoid too small jobs, we do not split beyond a minimal job size (10 events).
- If no node is idle but some job(s) is/are running in parallel on several nodes, one node is released by the job having the largest number of nodes per event to process. The corresponding subjob is suspended and the new job is launched on the released node.
- If there are as many jobs running as nodes, the new job is queued.

Upon *subjob end* (but not job end)

- If there are suspended subjobs within the same job, one of them is activated and runs on the node becoming free.
- Otherwise the node is allocated to an already running job. The largest subjob running on the cluster is split into two equal parts, one of them being launched on the free node. Again, jobs below a minimal size are not split.

Upon *job end*

- If there are queued jobs, the first queued job is run.
- Otherwise the free node is allocated to an already running job, as in the case of *subjob end*.

(e)

Fig. 9.8. Examples of true/false tables

3) We can also adjust the font sizes, the white space between columns and rows, or change the sizes of columns and rows to fix unpleasant proportions.

4) With detected *well-structured* tables, we can provide good examples of table designing and summarize valuable guidelines for further table authors.

5) Such classification has widely usage for many applications such as web mining, knowledge management, web content summarization, etc.

It may not be easy to define a *poorly-structured* table, however, some researchers gave requirements on a *well-structured* table: simple structure (easy to construct by authors, users to quickly learn the structure, economy of space, better visualization, consistent design structure allows for comparisons by highlighting the difference in the information). In Xingxing Wang's dissertation [89], he stated that a *well-designed* table should be "organized in such a way that the underlying logical structure is made obvious and tabular items are located and interpreted easily". According to the definition of *free dictionary*[2] a table should:

- has headers for better data accessibility;
- has caption (summary);
- Avoid nesting where ever possible;
- has suitable table width;
- Avoid vertical/rotated tables;

[2]http://www.thefreedictionary.com/table

Combing with other researchers' opinions, we notice that different people with different backgrounds and goals differ widely on some opinions. The spanned cell is an example: some researchers believe that by presenting the common value once the spanned cells can simplify the table contents, enable an easy table comprehension, and even reduce the table size if the common items occur very frequently. Many others do not encourage spanning cells because they impair the data accessibility. They argued that "If someone is using a text reader, each time someone is moving from cell to cell, the interpreter will make a sound to indicate a cell change. What happens to someone who is using a text reader on a page that is full of nested tables and row-span and column-span attributes? They are faced with a confusing series of beeps that do not in any way pertain to a textual interpretation of the page. "

9.2.3 Layout Design Guidelines

Although there is no strict definition on the table structure and layout designing, there are many recommendation for the good table design from the researchers/pulishers as followings:

- Avoiding source/English tables that "just fit" the depth of the page, if you want to maintain the same page breaks in your target language.
- If you want to maintain the same page breaks, keep about 30% of the body column free (or white space) below the table.
- Avoiding use of ALL CAPS in table header cells because the tables may be translated into another language, where language expansion may occur.

- Avoiding "rotated" (e.g. counterclockwise/vertically oriented) text in header rows.
- Avoiding nested, indented lists, even if table cells have been straddled to provide more width. Use wider cell margins in English (source) which can be reduced for target languages. For example, left/right table cells of 6 pts may be reduced to as little as 2 pts in German and Italian
- Some of the major WYSIWYG development software uses tables for exact positioning of elements on the page to the extent that every element in the page will be in its own table, nested inside other tables. This is not immediately apparent on a normal computer screen, but can cause problems for some users.

9.3 Experiment Results of Quantitative study

9.3.1 The Datasets

We focus on tables presented in scientific documents with PDF format. Although Wang [97] introduced a general table ground truth database where he focused on the web tables in HTML format, no benchmark dataset exists for PDF table analysis. To generate a PDF table benchmark dataset in our work, we directly extract tables from PDF documents instead of converting them to HTML or Image files based on the following three arguments: 1) extracting tables from PDF documents provides us essential information about a table, eg., table caption position. 2) converting PDF documents to HTML does not provide any additional help because detecting/analyzing the table is the foundation of all subsequences. 3) external tools (eg., OCR) can introduce unwanted errors and low performance of table analysis.

To create diversity in the PDF table benchmark, we extracted tables from a verity of scientific document digital libraries to evaluate the different classification methods described above. As our first data set we collected tables from Citeseer[3], the digital library for scientific and academic papers, primarily in the fields of computer and information science and engineering. This corpus contains about more than *20,000* articles evenly distributed among *20* different conference proceedings. After extracting and preprocessing the collected documents, we included *30,000* tables in our benchmark set. We also extract *10,000* and *5,000* tables, respectively, from the chemical scientific digital libraries (Royal Chemistry Society[4]), and archeology journal[5]. All the documents span years *1950* to *2008*.

The quantitative study is finished in two methods: *manually* and *automatically*. Although the sample size for the former way can not be very large (we randomly selected *100* PDF scientific documents from each repository), we can guarantee *100%* accurate results. The performance of the automatically quantitative study is determined by the quality of the extracted table metadata files. Although there is some inaccuracy in some sense, the inaccuracy can cancel out each other and the experiments show that the results of both methods are consistent with each other. In some sense, the second method (automatically) is used to prove the result reliability of the manual method.

[3]http://citeseer.ist.psu.edu/
[4]http://www.rsc.org/
[5]http://www.saa.org/publications/AmAntiq/AmAntiq.html

9.3.2 The Experiment Results

Table Location: Document section headings are of vital importance to the detection of the table location. It is easy to identify and understand the section headings in the documents in computer science field because most of documents still adopt these popular headings. However, we may need additional domain knowledge to understand them in some other fields, (e.g., the section heading "Ashing of Materials" in the ANALYST journal in RSC [6]).

Old papers usually talk about the experiments from the beginning of the documents and overlook the introduction and related work sections comparing with recent papers. Also, no section heading keyword exists at all for many papers in the old publications. We can only identify them by analyzing the text location and font size information. For the CiteSeer repository, our manual experiment shows that the ratio of table location distribution over three main sections (Backgrounds vs. Methods vs. Results) is *13% : 39% : 48%*. Similarly, the result of the automatical part is *11.82% : 41.94% : 46.24%*. For Chemistry field, the corresponding ratio of manual experiment is *0.0% : 11% : 89%*.

Table Sizes: For the CiteSeer repository, our manual experiment shows that the ratio of Cross-column vs. Single-column tables: is *72% : 28%*. Similarly, the result of the automatical part is *68.37% : 31.63%*. In addition, we manually examined the VLDB journal [7] and DAS [8] proceeding, the percentage of cross-column are *21.7%* and *21.05%* respectively. Although many journals in RSC repository have double-column layout, the ratio of wide tables is *71.33%* because of large experiment results. The reason for the

[6] http://www.rsc.org/
[7] http://www.vldb.org/archives/public/
[8] http://www.informatik.uni-trier.de/ ley/db/conf/das/index.html

large tables is that for the documents with a single column, all the tables should be cross-column tables. For documents with multiple columns, there are still large tables that span several document columns.

In document level, the ratio of documents with wide tables / the documents with tables is *82.93%*. the ratio of documents with wide tables / the total documents is *63.55%*. Table 9.2 shows the detailed information of the wide tables in RSC repository.

Table 9.2. The percentage of the wide table in RSC repository

Journal Names	TableLevel	DocLevelWithTables	DocLevel
QR(1950-1970)	100.00%	100.00%	72.00%
P1(1975-2000)	42.86%	50.00%	30.33%
DT(1975-2005)	56.59%	83.87%	74.29%
AN(1950–1970)	100.00%	100.00%	88.23%
RSC Average	71.33%	82.93%	63.55%
CiteSeer	72%	61.54%	32%

Table Caption: Almost all the table locations in the documents of RSC repository are top. The main reason for the non-100% is that there is no table caption in the early documents. However, many below table captions in CS field exist. The percentage of tables with top caption in DAS is *84.21%*. For VLDB proceedings, the percentages along each years are *18.35% (1985: 15.38%, 1990: 17.65%, 1995: 21.43%; 2000: 0%; 2005: 35.71%)*. Table 9.4 shows the detailed information of the top table caption in RSC repository. The average caption size in term of lines is 1.93 lines/table.

Frame and separators: We classify the frame and separator lines into four categories: *Complete* vs. *Vertical only* vs. *Horizontal only* vs. *None*. In CiteSeer repository the ratio of the four categories are *58.06% vs 0% vs 41.94% vs 0%*. In RSC

Table 9.3. The percentage of the top table caption location in RSC repository

Journal Names	TableLevel	DocLevelWithTables	DocLevel
QR(1950-1970)	92.11%	94.44%	68.00%
P1(1975-2000)	95.24%	88.89%	53.33%
DT(1975-2005)	92.22%	100.00%	88.57%
AN(1950-1970)	100.00%	100.00%	88.24%
Average	96.77%	96.34%	73.83%
CiteSeer	53.76%	46.24%	25.5%

repository, no vertical lines between data rows exists in any time. Several journals have different requirements on frame and separators along different ages, Table 9.4 shows some example results.

Table 9.4. The frame and separator lines of several RSC journals in different ages

Journal Names	The information of the table frame and separator lines
QR	1950–1955: complete (no vertical line between data rows), Between 1960-1970: none
P1	1970–1985: none, 1990–: horizontal only
DT	1975–1995: none; 2000–: horizontal only
AN	before 1985: none; 1985–: horizontal only

Among the frame and separator lines, the *stub separator* and the *boxhead separator* (See Figure 9.1) are extremely important because they divide a table into four main regions: the *stub*, the *boxhead*, the *stub head*, and the *body*. "The stub is the lower left region that contain the row headings. The boxhead is the upper right region that contains the column headings. The stub head is the upper left region that contains the categories in the stub. The body is the region to the right of the sub and below the boxhead that contains the entries". According to the completeness of these separators, we classify them into four types: *Complete* vs. *Boxhead separator only* vs. *Stub separator*

only vs *None*. Our experimental results on CiteSeer repository is *56.99% vs. 32.26% vs. 2,15% vs. 8.6%* respectively.

Data Separators: In order to get an accurate row/column structure information in a table, the data separators are very important. We classify them into four types also: *Complete* vs. *Vertical only* vs. *Horizontal only* vs. *None*. The percentages of each categories in the CiteSeer repository are: *46.24% vs. 8.6% vs. 8.6% vs. 36.56%*. RSC journals also have different requirements on data separators along different ages, Table 9.5 shows some example results. Because many tables do not have separators, space between texts and text alignment information are very important for structure decomposition.

Table 9.5. The data separators of several RSC journals in different ages

Journal Names	The information of the data separators
QR	before 1960: Complete but no horizontal data separator; 1965 –: none
P1	P1: before 1990: none; 1990–: boxhead separator
DT	before 2000: none; 2000–: boxhead separator
AN	before 1985: none; 1985–: boxhead separator

Alignments: We checked the vertical alignment and table cells manually. The column alignment is very important to the column structure decomposition. We noticed that different cell types have different alignment values. For CiteSeer tables, the distribution among Left/Central/Right-aligned texts is *54.84% : 23.66% : 3.2%*. Because almost all the cells in RSC tables are numerical data, most of them are aligned by the characters (e.g., dot).

Cell Types: We classify the table cells into three types according to the content types: *Numerical* vs. *Text* vs. *Both*. In CiteSeer repository, *70.97%* tables have numerical cells, *54.84%* tables have text cells and *25.81%* tables have both of them. We zoom in the DAS proceedings, *92.11%* tables have numerical cells, *15.79%* tables have text cells and *14.26%* tables have both types. For VLDB proceedings, *76.67%* tables have numerical cells, *70%* tables have text cells, and *32.71%* have both types. In RSC repository, *90.33%* tables have numerical data. In the document level, among all the documents with tables, *97.56%* have tables with numerical cells. Among all the documents, *74.77%* of them have tables with numerical cells. *78.81%* tables have text cells. In the document level, among all the documents with tables, 91.46% have tables with text cells. Among all the documents, *70.09%* of them have tables with text cells. We also examined other unusual cell types such as units, chemical formulas or images. Among all the tables in RSC, *5.38%* tables contain **units** and among all the tables in VLDB proceedings *5.4%* tables contain **units**.

In RSC repository, *65.80%* tables have chemical formula cells and *31.07%* tables have chemical formula images. In the document level, among all the documents with tables, *70.73%* have tables with chemical formula cells and *28.77%* have tables with chemical structure images. Among all the documents, *54.21%* of them have tables with chemical formula and *22.83%* of them have tables with chemical structure images.

Nesting/Spanning cells over columns: we measure the tables with physical nesting/spanning cells, as defined in *Section 9.2.2*. The nesting/spanning cells can be divided into two types: *Over columns* and *Over rows*. In RSC repository, early PDF documents use *braces* to annotate the nesting/spanning cells (see Figure 9.9). Within

TABLE II

HARDNESS FOUND BY DIRECT TITRATION

Figures for hardness are expressed as $CaCO_3$

	Added hardness			Hardness found		
	Calcium, p.p.m. w/v	Magnesium, p.p.m. w/v	Total, p.p.m. w/v	Calcium, p.p.m. w/v	Magnesium, p.p.m. w/v	Total, p.p.m. w/v
Distilled water	150	100	150	150·4	101·4	251·8
				150·1	101·7	251·8
	200	50	250	200·6	50·6	251·2
				200·6	50·6	251·2
	220	30	250	219·4	31·8	251·2
				218·7	32·7	251·4
	240	10	250	240·5	9·0	249·5
				240·5	8·7	249·2
Welwyn raw water				286·5	9·5	296·0
				285·9	9·9	295·8
Welwyn raw water + 50 p.p.m. of added magnesium ..				285·6	60·2	345·8
				285·6	61·1	346·7

Fig. 9.9. A table example in RSC repository with braces

all the tables *36.20%* of them contain nesting/spanning cells. Within the documents with tables, *54.55%* documents have tables with nesting/spanning cells. Among all the documents, *44.86%* of them contain such cells. Within all the tables in CiteSeer repository, *37.6%* tables contain nesting/spanning cells. Within the documents with tables, *46.15%* documents have such cells. Among all the documents, *24%* documents have tables with nesting/spanning cells.

We observed the documents in RSC repository in the chemistry field. The documents span from *1950 to 2005*. Table 9.6 shows the detailed statistical results based on several RSC journals and CiteSeer repository. Overall, among documents with tables, more than half *(54.55%)* of the documents contain cross-column nesting cells. Among all documents, almost half *(44.86%)* of them contain cross-column nesting cells.

Nesting/Spanning cells over rows: the detailed quantitative results of nesting/spanning cells over rows are displayed in Table 9.7.

We also calculate all the nesting cells in RSC repository. In the table level, 67.7% tables contain nesting cells over either rows or columns. In the document level, *63.41%* documents with tables contain such cells and *50.98%* documents contain such cells.

Table 9.6. The percentage of the nesting/spanning cells over multiple columns in RSC tables

Journal Names	TableLevel	DocLevelWithTables	DocLevel
QR(1950-1970)	18.42%	29.17%	28.00%
P1(1975-2000)	35.71%	44.40%	26.67%
DT(1975-2005)	41.09%	74.19%	65.71%
AN(1950–1970)	59.38%	67.70%	58.82%
Average	36.20%	54.55%	44.86%
CiteSeer	37.6%	46.15%	24%

Table 9.7. The percentage of the nesting/spanning cells over multiple rows in RSC tables

Journal Names	TableLevel	DocLevelWithTables	DocLevel
QR(1950-1970)	18.42%	25.00%	24.00%
P1(1975-2000)	0%	00%	0%
DT(1975-2005)	2.33%	3.23%	2.86%
AN(1950–1970)	12.50%	13.30%	11.76%
Average	7.53%	10.23%	8.41%

Missing/Empty cells: in RSC repository, about one third (*32.97%*) tables have missing cells. In the document level, almost half (*42.05%*) documents with tables and one third (*34.58%*) documents have such cells. In CiteSeer repository, similar ratios exist: (*35.48%* table have missing cells. *34.62%* documents with tables have missing cells; *18%* documents have tables with missing cells). The detailed results based on multiple RSC journals can be found in Table 9.8. We notice that the journal ANALYST does not contain lots of missing cells as well as P1(J. Chem. Soc., Perkin Trans.) is the journal with lot of missing cells.

Basic tables vs Composite tables: We adopt the same definition in [70]. Please see the example in Figure 9.10. Different people may have different definition on

Table 9.8. The percentage of the empty cells in RSC tables

Journal Names	TableLevel	DocLevelWithTables	DocLevel
QR(1950-1970)	31.58%	29.17%	28.00%
P1(1975-2000)	47.62%	61.10%	36.67%
DT(1975-2005)	34.88%	54.84%	48.57%
AN(1950-1970)	9.38%	13.30%	11.76%
Average	32.97%	42.05%	34.58%
CiteSeer	35.48%	34.62%	18%

1-D table and 2-D table. Please see Figure 9.11. DAS has 7.9% composite tables and VLDB has 14.68% composite tables;

Other Results: all the other important results are listed in Table 9.9. The *standard* table refers the *non-rotated* table that does not have nesting cells, missing cells. Based on the observation on our table repositories, we notice that the most frequent table structure is 4 rows * 6 columns or 6 rows * 4 columns.

Trip Code	Trip Duration (in days)	Cost	Insurance
LM202	-7	520	50
LM208	9-15	725	70
LM209	16-23	949	90
LM311	23-31	1495	120
LM223	32-45	2275	195
XM001	46-60	3180	220

(a) 1D

	Departure	Arrival
City:	Dallas/Ft Worth, TX (DFW)	Honolulu, HI (HNL)
Scheduled:	Mar 16 - 11:45am	Mar 16 - 4:20pm
Actual:	Mar 16 - Not Available	Mar 16 - Not Available
Gate/Terminal:	A29	18
Baggage Claim:	-	F2

(b) 2D

Rate (%)	Regular	Float
Regular Fixed Deposit		
1 Year	5,05	5,05
2 Years	5,1	5,1
3 Years	5,1	5,1
Fixed Deposit		
3 Months	4,35	4,35
6 Months	4,6	4,6
9 Months	4,7	4,7
1 Year	5	5
2 Years	5,05	5,05
3 years	5,05	5,05

(c) Partition table

Fig. 9.10. Examples of 1D, 2D, and Composite Tables in [70]

Summarization based on RSC repository After implementing such a huge table quantitative study and comparing the results among different field repositories, we

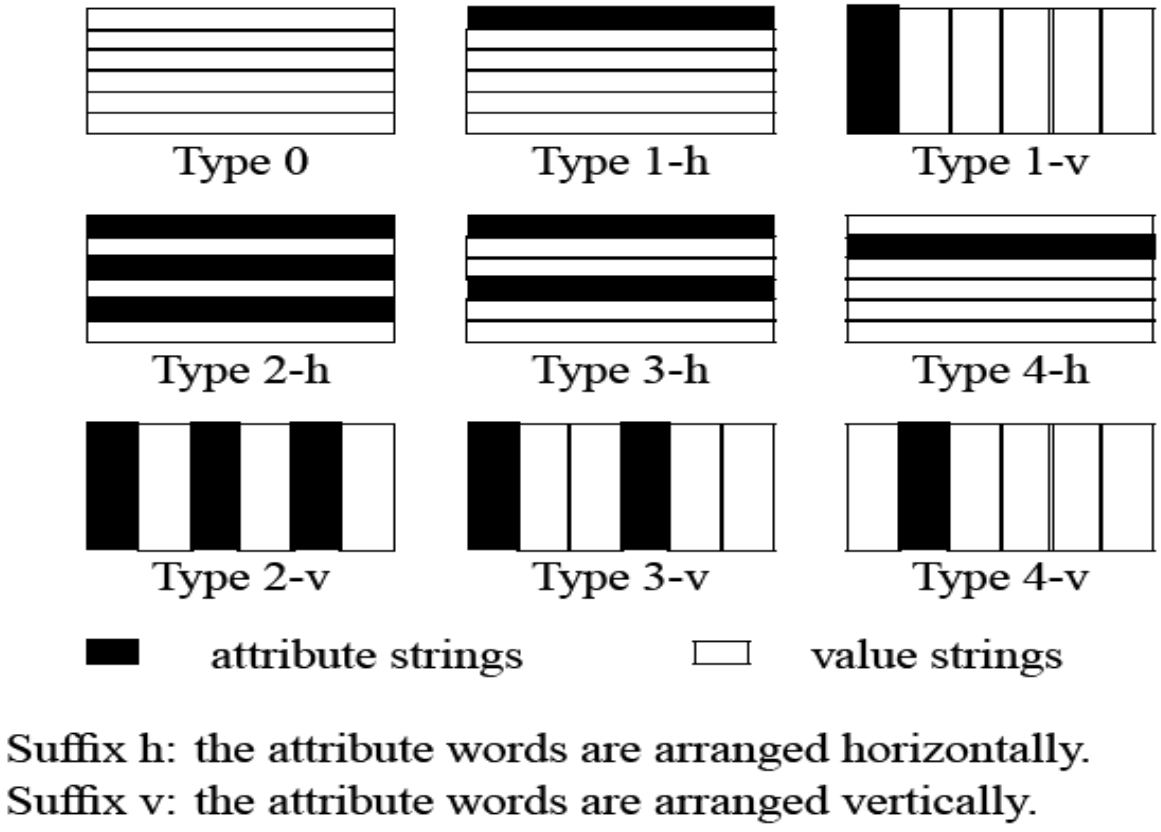


Fig. 9.11. 9 table types in [101]

Table 9.9. The results of other important parameters in RSC and CiteSeer repositories

Journal Names	Perspectives	TableLevel	DocLevelWithTables	DocLevel
CiteSeer	Super Row Label	34.41%	42.31%	22.00%
RSC	Super Row Label	23.73%	44.45%	29.09%
CiteSeer	Multi-dimension Tables	1.08%	3.85%	2.01%
RSC	Multi-Dimension Tables	32.30%	23.17%	18.63%
CiteSeer	Composite Tables	12.9%	10.2%	6.48%
RSC	Composite Tables	8.68%	9.23%	4.34%
CiteSeer	Rotated Tables	1.08%	3.85%	2.01%
RSC	Rotated Tables	4.35%	4.88%	3.92%
CiteSeer	Standard Tables	73.29%	61.54%	32.00%
RSC	Standard Tables	6.83%	26.83%	21.47%

have the following observations and summarizations, which are beneficial for our table understanding.

- In early time, there is no clean document structure or sections;
- In RSC repository, there are many two-dimension or three-dimension tables, especially in the early journals;
- There are many miss-labeled tables in early RSC repository. Such phenomenon also happens frequently in current CiteSeer repository.
- There are many tables that have keywords but do not have caption content;
- Many early journals use dotted lines to separate the columns;
- Most tables have clear column structures, but unclear row structures, especially in Chemistry field;
- Semantic analysis is needed to identify whether a text piece in a table is a nested cell over multiple rows or multiple cells in the same column over multiple rows;

- For survey/review papers, some tables are located in the beginning of the document;
- If we find a table in a document, this document has a very large possibility of having multiple tables;
- Many table footnotes do not start with the special symbols, especially in CS field.
- Some tables are so large that span over several pages; Such tables always contain multiple parts. We suggest to segment these tables into several small parts;
- In early chemistry publications, many authors put the Results and Discussion section ahead of the Experiment section;
- Some tables in chemistry put chemical formula structure figures between the table caption and the column headers. It is difficult to know whether they belong to the caption or the column headers;
- In order to save the space, they usually put all the footnotes together into a paragraph instead of several separate lines (see example in Figure 9.12);
- Some tables use index numbers to facilitate the row identification;

Table 1 Determination of drugs in pharmaceutical preparations

Drugs	Commercial formulation[a]	Drugs nominal value	Drugs found (s)[b]
Ibuprofen	Neobrufen (effervescent granulates)	92.7 mg g^{-1}	93.1 (0.4) mg g^{-1}
Diclofenac	Voltarén Retard (tablets)	314 mg g^{-1}	314 (1) mg g^{-1}
Indomethacin	Inacid (suppositories)	60 mg g^{-1}	60 (1) mg g^{-1}
Carbenoxolone	Sanodin (gel)	20 mg g^{-1}	20.1 (0.4) mg g^{-1}
Diflunisal	Dolobid (tablets)	590 mg g^{-1}	590 (7) mg g^{-1}
Piroxicam	Feldene (blisters)	20 g l^{-1}	20 (0.2) g l^{-1}
Phenylbutazone	Butazolidina (suppositories)	125.4 mg g^{-1}	125.3 (0.7) mg g^{-1}
Dexamethasone	Fortecortín (tablets)	10.05 mg g^{-1}	10.1 (0.1) mg g^{-1}

[a] Composition of commercial formulations: Neobrufen (Abbott Laboratories, Spain): saccharose, 51.45% (w/w) and others excipients (sodium carbonate, cellulose, sodium croscarmellose, malic acid, povidone, sodium hydrogen carbonate, orange essence, sodium dodecyl sulfate and sodium saccharin), 39.28% (w/w). Voltarén Retard (Novartis Pharmacéutica, Spain): saccharose, 40.16% (w/w) and others excipients (colloidal anhydrous silica, cetylic alcohol, magnesium stearate, povidone, hypromellose, red iron oxide, polysorbate 80, talc, titanium IV oxide and polyethylene glycol 8000), 26.08% (w/w). Inacid (Merck Sharp & Dohme, Spain): excipients (butylhidroxyanisol and butylhidroxytoluene), 93.9% (w/w). Sanodin (Altana Pharma, Spain): excipients (karaya gum, vaseline oil and polyethylene), 98% (w/w). Dolobid (Frosst Laboratories, Spain): excipients (sodium croscarmellose, hydroxypropyl methylcellulose, starch, propylene glycol, sodium stearyl fumarate, and titanium dioxide), 41% (w/w). Feldene (Nefox Pharma, Spain): ethanol, 12.8% (v/v) and others excipients (benzilic alcohol, sodium phosphate, nicotinamide, 1,2 propanediol, sodium hydroxyde, hydrochloric acid and water), 85.2% (w/v). Butazolidina (Padró Laboratory, Spain): excipients (triglycerides), 87.45% (w/w). Fortecortín (Merck Pharma and Chemistry, Spain): excipients (lactose, starch, talc and magnesium stearate), 98.99% (w/w). [b] s denotes standard deviation (n = 6).

Fig. 9.12. A table example with multiple footnotes.

Chapter 10

Experimental Results

10.1 Document Collection

Although *TableSeer* can be used to search for tables contained in documents of varying formats and media types, in this work, we focus on tables in scientific documents in Portable Document Format *(PDF)*. The document collection comes from three sources: 1) scientific digital libraries (Royal Chemistry Society), 2) the web pages of research scientists in chemistry departments in universities, e.g., UCLA[1] lists numerous addresses of academic institutions in the field of chemistry, which are set as seeds for the table crawler. and 3) the CiteSeer archive. The crawler uses a depth-first crawling policy with a maximum depth of five (from the seed URLs). The total number of collected PDF documents is approximately $10,000$. These documents belong to more than 50 journals and conferences in a variety of research fields, e.g., chemistry, biology, computer science, etc. All the documents span the years 1990 to 2006. A random selection of 100 documents indicates that 78% of them have at least one table and most of them have more than one.

[1] http://www.chem.ucla.edu/VL/Academic.html

10.2 Experimental Results of Table Detection

We performed a five-user study to evaluate the quality of the table detection and the table metadata extraction separately. Each user randomly checked 20 documents and all the corresponding table metadata files. The evaluation metrics are precision and recall. The experiment on table detection was conducted on a document set with 200 randomly selected PDF documents, which were from journal/conference categories of Chemistry, Biology, and Computer Science from RSC and CiteSeer. Given the number of true tables extracted by *TableSeer* A, the number of true positive tables but overlooked B, and the number of true negative tables that is misidentified as tables C, the *Precision* is $\frac{A}{A+C}$, and the *Recall* is $\frac{A}{A+B}$. Within the 200 documents, 397 tables exist (see Table 10.1). *TableSeer* recognizes 371 tables and all of them are real tables, which means the number of true negative tables is 0. Based on these limited experimental results, the precision value is 100% and the recall value is 93.5%.

Table 10.1. The testing set for table detection

Fields	Documents	Tables
Chemistry	100	245
Biology	50	84
Computer Science	50	68

10.3 Experimental Results of Table Metadata Extraction

We evaluated the table metadata extractor of *TableSeer* using the 371 recognized tables from the document set. Precision and recall remain the measurement functions. A is the number of true positive metadata extracted by our algorithms and labeled with

the correct metadata labels, B is the number of true positive metadata but overlooked by *TableSeer*, and C is the number of true negative metadata that are misidentified as another metadata.

Table 10.2 demonstrates the performance of the metadata extraction over the test set. Apparently, *TableSeer* has good performance on table metadata extraction. The main reasons for inaccuracy of the recall are: First, some journal names and volume information were mislabeled as "Document Title" metadata. Most journals put the journal names and the publication information in the page header or footer. However, some of the documents display the journal/volume information in the ordinary position where the document title is. This problem can be solved by adding heuristic rules that look for patterns to identify journal/volume information. Second, some documents omit the keyword "Abstract" when they start the abstract section using the same font size as the size of the author/affiliation section. This can be fixed by restricting the size of the author/affiliation section and by looking for text similarity. The abstract usually has similar keywords as in the main text, whereas the author/affiliation will have distinct words than the main text. Third, some table footnotes, especially short ones, are recognized as a row of the table because some tables have such cross-column rows.

In order to obtain the table reference text metadata, a "keyword matching" method is used to collect all the sentences where the table keywords appear. Some explanations also exist around without the keywords. However, it is extremely difficult to separate them from other texts. Furthermore, for those tables that use a cross-line to separate different parts, we only recognize the first part because of the difficulty in

Table 10.2. Table Metadata Extraction Performance

Table Metadata	Precision(P %)	Recall(R %)
Document Type	100.00	100.00
Document Page Number	100.00	100.00
The page number of table	100.00	100.00
Document Title	95.65	95.65
Document Author	92.58	92.58
Table Caption	95.96	95.96
Table Column Header	93.79	93.79
Table Content	90.15	90.15
Table Caption Position	100.00	100.00
Table Footnote	82.77	82.77
Table Reference text	100.00	100.00

telling whether the cross-line belongs to the table or the body text of the document. This can be solved by using the font-size difference, existence of cells, and the indication of the white spaces.

10.4 Experimental Results of Table Ranking Results

We compared different ranking schemes and show their results. We also measure the differences and correlations among these ranking schemes using the Kendall or Spearman distance[74]. However, this computation is not sufficient for determining which ranking scheme is better. Comparison of ranking methods is complicated for two main reasons. First, it is difficult to find a common test-bed for different search engines. Although Google Co-op[2] can enable the comparison between TableRank and Google by harnessing the power of Google search technology, the dynamic web and continued crawlers imply that the two sets of documents may not be the same. Second, even with a

[2]http://www.google.com/coop/

common test-bed, no universally recognized measurement of quality for a table ranking scheme exists.

10.4.1 Comparative Retrieval Effectiveness of TableRank

In order to set up a well-accepted measurement, we established a "*golden standard*" to define the "*correct*" ranking based on human judgement. A survey of six experienced testers, who frequently use search functions of different search engines, generated the "*golden standard*" of each document set in the test-bed. For each issued query, the testers determine how many results, among the returned hits, the testers order the relevant lists.

For each ranking scheme, we applied *pairwise accuracy* to evaluate the ranking quality. If H_R is the ranking decided by human judgement and T_R is ranking decided by the search engines, the *pairwise accuracy* can be defined as the fraction of times that search engines and human judges agree on the ordering of tables: *pairwise accuracy* = $\frac{|H_R \bigcap T_R|}{H_R}$. In this section, we provide quantitative comparisons between *TableRank* and other popular web search engines based on experimental results. The average ranks of the six human testers is obtained and the results are ranked using this average to obtain H_R. We will use a distance based measure among the ranked orders in the future.

Before evaluating the performance of the table ranking, we identified the returned true hits because most search engines can not identify documents with tables. Among their returned results, a large part do not have table or do not match the query, we call these results as false tables. We should filter out all the false tables and only compare the ranking on these real tables. We randomly selected 100 terms from the *Popular*

Term List as mentioned in Section 4.1, such as "gene", "protein", "query,", and used each term as a query in *TableSeer*. In addition, we also selected 20 terms beyond the *Popular Term List* because users may pose free queries with any possible terms, such as "result", "document", etc. For the other two search engines: *Google Scholar* and *CiteSeer*, the query was set as the term together with an additional keyword "Table." An example is applying "Alkaline" to *TableSeer* and applying the combination of the term "alkaline" and "Table" to *Google Scholar*. The combination includes "alkaline, Table", "alkaline Table", "Table, alkaline", "Table alkaline", etc. Because it is not easy to know the total number of tables in the test-bed, we used the precision value as the measurement. Another reason is that with the growing size of the Web collections, users are now primarily interested in high accuracy, defined as high precision. High precision is very important because users typically look at very few results and mostly look at the top N items of the returned results from the search engines. Recognizing this trend, we manually examined the first 20 results returned by both each search engine. Comparing with the precision value of our *TableSeer* (100%), the precision value of *Google Scholar* is 73.4% and the precision value of *CiteSeer* is 76.7%. Thus, it can be seen that *TableSeer* outperforms the other two search engines.

The remaining question is which search engine has the best ranking scheme? We adopted two methods to set up the common test-bed: the manually "bottom-up" method and the custom search engine method. The manually "bottom-up" method constructs a test-bed using the following two steps: 1) applying a query (e.g., DNA) together the keyword "Table" to an existing search engine, e.g., *CiteSeer*; 2) from the numerous returned results, we pick out a set of "real" hits in PDF format together with

their ranking orders. We set these PDF documents as the common test-bed for both *TableRank* and other search engines. In the second custom search engine method, we set up a common test bed for *TableRank* and *Google* using the *Google* Custom Search engine API. We registered a *Google* account and used the *Google* custom search engine API[6] to build a customized search engine. We set the seed URLs (e.g., a chemistry journal website[3]) as one or several websites where *TableSeer* crawls. All the documents in the seed URLs construct the common test-bed. For all the documents in the common test-beds, *TableSeer* extracts and indexes the metadata file for each table. We tried 20 randomly selected search queries on both TableSeer and the Google custom search engine to compare their search results. We collected 20 document sets for 20 random queries. Table 10.3 displays the average pairwise accuracy results made by six users.

Table 10.3. The Basic Ranking Results on the Manually Created Document Sets

Ranking	The Method to set-up the test-bed	Accuracy (%)
Google	Custom search engine	51.8
Google Scholar	bottom-up method	52.72
CiteSeer	bottom-up method	55.35
TableSeer	Both methods	69.61

[3]http:/www.rsc.orgdelivery_ArticleLinking DisplayArticleForFree.cfm?doi=b2*

10.4.2 Factor Influence in TableRank

The *TableRank* algorithm decides the relevance score for each result by comprehensively considering multiple impact factors from different perspectives. No absolutely correct parameter setting exists. In order to find out how well each impact factor performs and how heavily each of them influence the final ranking, we implemented *TableRank* algorithm on each impact factor, independently, and applied varied combination of the impact factors by gradually adding one new factor at a time. Such implementation can not only reveal how sensitive *TableRank* is to each impact factor, but also show how to adjust the parameters for better results.

Table 10.4. Results for Individual Impact Factor in TableRank Algorithm

Impact Factors	Accuracy (%)
TFIDF	50.19
TTFITTF without MW	61.46
TTFITTF with MW	63.55
TLB	29.60
DLB	40.33
All factors	69.61

The results for individual impact factors are shown in Table 10.4, which shows the application of a single factor each time. The first run with the applied traditional *TFIDF* weighting scheme on the whole document shows the accuracy rate compared with the "golden standard". On the second and the third runs, we updated the *TFIDF* to the *TTFITTF* weighting scheme without/with the metadata weighting scheme. Next, the *TLB* factors are tried, and in the last run, the *DLB* factors are tested.

TTFITTF with/without metadata weight contributes heavily to the final ranking quality, and *TLB* and *DLB* do not perform well in isolation. Even Table 10.4 shows the effectiveness of each individual factor.

Table 10.5. The Ablation Experiment to Learn the Real Contribution of Each Factor of TableRank Algorithm

Without Factor	Accuracy (%)	Decrease in Accuracy (%)
DLB	68.50	1.11
TLB	69.46	0.15
MW	68.19	1.42
TTFITTF, MW	58.05	9.56

In order to have a better measurement of the contribution of each factor, in the context of all the other factors, we performed an ablation experiment. The basic idea was to compare the ranking results based on all the factors except the studied ones with the ranking results generated by *TableRank* with all the factors. The detailed results are shown in Table 10.5. The results in Table 10.4 are consistent with the results of the ablation experiment in Table 10.5. Table 10.5 reconfirms the vital role of the *TTFITTF* weighting scheme.

From above experimental results, we can say that our *TableRank* largely outperforms other web search engines in the table searching and ranking field. Within the *TableRank* algorithm, *TTFITTF* acts as the most important impact factor, and the second and the third in importance are metadata weight and *DLB* respectively.

10.5 Experimental Results of Text Sequence Fixing Algorithms

Our experiments can be divided into two parts: the evaluation of the text sequence recovering and the table boundary detection. A five-user study is implemented.

Each user checked the sequence of the resorted sparse lines in 20 selected PDF pages. Half of them have within-table sequence errors and the other half have beyond-table sequence errors. The evaluation metric is *pairwise accuracy*. If H_R is the sequence decided by human judgement and A_R is the sequence decided by the algorithms, the *pairwise accuracy* can be defined as the fraction of times that our algorithms and human judges agree on the sequence: *pairwise accuracy* $= \frac{|H_R \cap A_R|}{H_R}$. For algorithm 1, the correct sequence of human judge is column by column. For algorithm 2, humans treat each page as a single-column page. Comparing such "*golden standard*", the *pairwise accuracy* of both algorithms are 100%. Still using the Figure 7.1a as the example, with our algorithms, the 13 non-duplicated Y-axis values after the sorting step are: *625.47974, 643.07983, 651.9796, 660.68005, 669.5801, 681.4798, 690.27966, 699.0797, 707.7796, 716.3797, 725.1799, 734.0797, 744.37976*. Because the text sequence error always starts with the left columns first, we do not need to sort the text pieces with the same Y-axis values according to the X-axis coordinate. After the sorting, the new sequence of these sparse lines are: *"Content (% m/m)", "Proposed method", "Reference", "Peak Peak Flame value", "Sample Element height area AAS (%m/m)", "Low melt Ag 43.9 44.1 45.1 44.46", "0.5 2.4 1.1", "Cu 15.6 15.2 15.3 14-16", "2.7 1.8 2.5", "Sample No.318+ Ag 25.7 25.2 25.7 ≈25", "2.1 2.1 1.4", "Cu 34.4 33.7 34.8 ≈35", "2.3 1.3 2.5".*

Table 10.6. The performance evaluation of two text sequence resorting algorithms

	Algo1(P)	Algo1(R)	Algo2(P)	Algo2(R)
Cross-column tables	95.7%	49.8%	100%	99.6%
Single-column tables	96%	94.8%	94.5%	99.8%

The evaluation metrics for the table boundary detection are precision P and recall R. Given the number of true table lines in our detected table boundary A, the number of overlooked true positive table lines B, and the number of misidentified true negative non-table lines C, P=$\frac{A}{A+C}$, and R=$\frac{A}{A+B}$. The performance of two algorithms are listed in Table 1. The reason for C are some non-table sparse lines surrounding the table boundary. They usually also belong to the table data.(e.g., the short lines of the table caption and footnote). For single-column tables, Algorithm 2 may have more C because of the possible sparse lines with similar Y-axis values in the next document column. Because the widths of cells in single-column table can not be large and the noise lines in table boundary can be easily removed in later table structure decomposition step, we prefer high recall values. For cross-column tables, algorithm 1 will cut each table into at least two parts. A merge postprocessing is required. Algorithm 2 achieves better performance on such tables. In summary, in the real application when we can not know the table type in advance, algorithm 2 works better than algorithm 1. For tables with within-table sequence errors, our recovering algorithms do not fulfill much performance improvement on both P and R because all the lines belong to the table boundary come together without interruption. However, for the tables with beyond-table sequence errors, the performance is much worse without our algorithms: with the same test PDFs, the R is only 63.5% without our algorithms because a table will be segmented into several parts and some of them are filtered out because of the small scope.

10.6 Parameter Settings

Before analyzing the experimental results, we discuss the settings of important parameters in $TableRank$ algorithm.

10.6.1 Parameter Setting of the Metadata Weight

Intuitively, the weight of a metadata MW_k is proportional to the occurrence frequency of the meaningful words in the metadata. Meaningful words refer to those representative terms which reflect the end users' query interests. In the experiments, the determination of each metadata's weight MW is based on a study of the keyword distribution over different metadata. First, we randomly select a series sets of sample tables from different research fields, e.g, chemistry, computer science, etc. For each table tb_j, a *Term Dictionary* is generated . After deleting the *stop list words*, the *Term Dictionary* is created in a descendent order according to term-occurrence frequency. The top φ meaningful terms from the ordered *Term Dictionary* are identified manually to construct a *Popular Term List*. All the selected terms have maximum likelihood for use to construct queries for searching these sample tables. Although different tables have different *Popular Term Lists*, we tried 10 different sample table sets, the term occurrence distributions over the metadata are similar to each other.

A term may appear in multiple table metadata. For example, in a randomly select table, term "Microcystins" appear in three metadata: table caption, document title, and the column heading while the term "dye" from another table appear in four metadata: table caption, table column heading, table footnote, and the referenced text.

For each metadata, calculate the summation of the term-occurrence frequency of all the terms in the *Popular Term List*. A large summation means that this metadata has larger possibility to contain the terms, which are candidates to construct good queries to search these tables. We should give high weight to this metadata.

In our experiment, to determine the metadata weights, 100 tables were randomly selected as the sample set. Half of them come from the field of chemistry and the other half come from the field of computer science. For each table, φ is set to 10 and the term-occurrence frequency of the top 10 keywords within each metadata is calculated. The empirical results of the term distribution over each table metadata is shown in Figure 10.1. The X-axis lists seven main text-based table metadata, ordered by the weights we given to them. The Y-axis shows the percentage of the term occurrence frequency of each metadata over all the metadata. From Figure 10.1, we can see that the metadata "Document Title", "Table Column Heading", and "Table Caption" are the top three metadata and should get the highest metadata weight.

Based on that, an approximate metadata weights MW_k is set as described in table 10.7. We keep the metadata with the lowest percentage ("Table Footnote" and "Author" in Figure 10.1) intact by assigning their MW as 1. All the other MWs are calculated according to their ratio comparing with those lowest MW.

Table 10.7. The metadata weights in TableRank algorithm

Metadata Names	Metadata Weight
Document Title	4.60
Table Column Heading	4.40
Table Caption	4.00
Table Reference Text	1.75
Table Cells	1.25
Author	1.00
Table Footnote	1.00

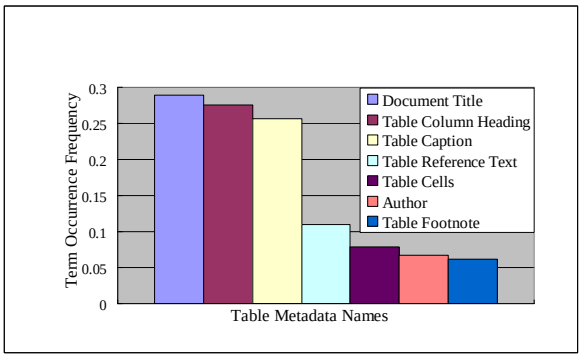


Fig. 10.1. An Example of the Metadata Distribution

10.6.2 Parameters for Document Origination

In each research field, scholarly journals or conferences are scored and ranked based on the opinions of field experts. *CiteSeer* gives an estimation of the impact ranking for the computer science publications and *Wikipedia*[4] estimates for chemistry papers. A comprehensive journal impact factor list spanning the years 2002-2004 for all the fields can be found in *CNCSIS*[5]. Different schemes have different score ranges from $[0, 3.31]$ to $[0, 53.055]$. We normalized the different ranges into the same range $[1, 10]$. For those journals or conferences that appear in both *Wikipedia* and *CNCSIS*, we normalize them respectively, then take the average value as the final score. Table 10.8 lists examples of the document originations with the corresponding importance scores.

Table 10.8. An Example of The Importance of Conference/Journal

Conference/Journal	Field	Importance Weight
WWW	CS	7.9
Analytical Chem	Chemistry	7.5
Nature	Biology	9.8

[4]http://en.wikipedia.org/wiki/List_of_scientific_journals_in_chemistry/

[5]http://www.cncsis.ro/PDF/IF_2004.pdf

10.6.3 Parameter Setting of Document Freshness

Document freshness (DF) refers to the age of a document. *TableRank* assigns more weight to fresher documents for two reasons: 1) Researchers usually are inclined to search for the more recently published documents because these documents typically reflect the latest research trends and results. 2) There is a need to eliminate "The Rich Get Richer" phenomenon which is begot by the boosting value of the citation frequency (the more citation, the higher boosting is). The documents with high citation counts are inclined to be cited again because they are more likely to be found. However, newly published documents are unlikely to attract attention and be cited within a short period of time. Considering these two reasons, we boost the rankings of recent documents. The more recent the document, the greater the boost value is. We define DF as the length of the gap between the year when the document is published and the current year. Because of the high-speed of information propagation and the convenient source sharing (e.g., DBLP bibliography[6]), in our *TableRank*, we apply the DF boost to brand-new documents, which are published within two years.

[6]http://www.informatik.uni-trier.de/ ley/db/

Chapter 11

Error analysis

Because text parser tools (PDFBOX or TET) are used to get all the textual information from the corresponding PDF document, the correctness of the textual information directly affects the performance of the table metadata extraction results. For example, missing characters and space insertions are the two typical errors inherited from text parser tools, which tamper our work. In addition, both of them may generate the errors of the coordination system, which seriously harms the performance of later steps, such as the table column detection. PDFlib TET, as a newly released commercial software package, has much less such errors than the open-source package, PDFBOX. However, such errors inevitably exist.

Because the performance of the table metadata extraction is crucial to the performance of the later table ranking and searching, in this chapter, we address the typical errors we have encountered and analyze the reasons of these errors.

Our table metadata extraction follows these steps:

- Table keyword detection;
- Table caption detection;
- Table boundary detection;
- Table footnote detection;

- Table column analysis;
- Table column heading detection;
- Table row analysis;

Our error analysis follows the same order as the table metadata extraction. We discuss the errors one by one in the following sections.

11.1 Error analysis on the table keyword detection

We define a keyword list to start the detection of the table candidates. This predefined keyword list contains all the possible starting keywords of the table captions, such as "Table, TABLE, Form, FORM," etc. Statistically, most tables have one of these keywords in their captions. However, we encountered many cases that the text extraction tools parse the crucial keyword "Table" to "Lable", "able", "a ble", "ble", or even worse by missing the whole keyword. This problem is orthogonal to our work and beyond my control to solve. We hope that these problems will be addressed independently by the designers of the text extraction tools.

An alternative solution to such problems is that, for these returned words "Lable", "able", "a ble", and "ble", we can still treat them as the starting places of the potential tables. If the whole keyword is missed, we are not able to detect the tables. An alternative solution to such cases is that we can analyze the structure of the entire page using the same methods as applied in the image table detection domain. However, to process detect such infrequent cases (3 over 123 testing tables) is costly.

Moreover, some tables are labeled using other keywords, especially in the documents from computer science field. The authors in this domain usually name a table as "Figure.". This confuses with the real figures. To avoid real figures and to keep high efficiency, *TableSeer* overlooks such "wrongly" labeled tables in the current stage. However, this problem can be overcome by heuristic rules that identify the grid-structure cells by white-space analysis.

11.2 Error analysis on the table caption detection

After we identify the table keyword, we need to decide which sentences constitute the table caption. For each text piece next to the keyword, we have to judge whether it belongs to the table caption or not according to a rule: the distance between the current line and the previous line should be no more than a threshold. At the current stage, we set this threshold to the average line gap size of the whole document. Because of the diverse document layouts and the publisher requirements, it is unavoidable to have some special cases. Among the 123 testing tables, there are five tables that omit the last short line of the caption. Figure 11.1 shows an example that does not include the line $c_{flame} \times 100$ into the table caption. This type of errors should be fixed by combining the line gap information.

11.3 Error analysis on the table boundary detection

After we fix the table caption, the next task is to decide the table boundary. Currently, we use a loose rule to detect the stop line of a table. Because the disordered

Table 5 Comparison of flame photometry, ETH 2137–BBPA ISE and di-*n*-butylamide–oNPOE ISE, using concentrations directly. Concentration units, mmol dm^{-3}; c_{un}, uncorrected concentrations; c_{cr}, concentrations corrected for plasma water; refractive index for plasma sample = 1.354; plasma water = 91.5%; [Na] in plasma determined by atomic absorption spectrometry = 148 mmol dm^{-3}; error (%) = $(c_{ISE} - c_{Flame})/c_{Flame} \times 100$

	ETH 2137*			Di-*n*-butylamide*			Flame†
Solution	*c*		Error (%)	*c*		Error (%)	*c*
AQ1	0.40		+1.1	0.40		+1.1	0.39
AQ2	0.86		+6.2	0.78		−3.7	0.81
AQ3	1.24		−13.9	1.39		−3.5	1.44
AQ4	1.58		−9.2	1.70		−2.3	1.74
AQ5	2.19		−14.0	2.41		−5.5	2.55
AQ6	3.28		−15.0	3.67		−4.9	3.86
AQ7	3.79		−14.4	4.22		−4.7	4.43
	c_{un}	c_{cr}	Error (%)	c_{un}	c_{cr}	Error (%)	
PL1	0.57	0.62	+19.2	0.46	0.50	−3.8	0.52
PL2	0.97	1.06	+0.9	0.99	1.08	+2.8	1.05
PL3	1.43	1.56	+3.3	1.37	1.50	−0.7	1.51
PL4	1.69	1.85	+0.5	1.65	1.81	−1.6	1.84
PL5	2.40	2.62	−7.1	2.43	2.66	−5.6	2.82
PL6	3.34	3.65	−6.6	3.39	3.71	−5.1	3.91
PL7	4.29	4.69	−6.6	4.37	4.78	−4.8	5.02

* Error on ISE values, about 1% (equivalent to ±0.3 mV).
† Error on flame values, about 3%.

Fig. 11.1. An example table omitting a piece of the table caption

text sequence exported by the PDFBOX and the TET heavily disturbs the later metadata extraction, we prefer to have a larger table boundary in order to not miss a table candidate. Otherwise, it is very often to only detect a part of the table boundary. For the larger table boundary, we can filter out those non-table text lines later. The loose rule is that if the width of the next line is shorter than a threshold, or if this line starts with some specific characters (e.g., the footnote line), we treat this line within the table boundary. We set the threshold of the line width as the width of the document column. Currently, TableSeer does not work on those tables with one super-wide column because such columns can be the document columns.

Once we detect a long line that is not the footnote, we treat it as the stop place of this table. However, such condition may cause us miss a part of the table contents (long cross-column table cells). To deal with this problem, in the next step, we plan to allow at most two continuous long lines within a table boundary.

11.4 Error analysis of the table footnote detection

There are two error types on the table footnote detection: treating a short table footnote as the table content line, and treating a table footnote line as the document body text line.

Currently, we use a simple but workable method to detect the table footnote: if a line starts with one of the predefined specific characters (e.g., *, or $), we treat it as the starting place of the table footnote. All the lines within the detected table boundary after this line are also treated as the footnote lines. Because of the disorder of the text pieces parsed by the PDFBOX and the TET, many footnote lines are located in the middle of the parsed text sequence among whole boundary. To avoid treating all the later lines as the table footnote, we sort all the lines with the detected table boundary according to their Y-axis values. Also, the performance of the table boundary detection impacts the performance of the table footnote detection. Among 123 testing tables, 12 of them have minor footnote errors. Hopefully, after the improvement of the table boundary detection, we can also improve the footnote detection.

11.5 Error analysis of the table column detection

After the detection of the table boundary and the table footnote, we treat all the other lines as the table body. To detect the table column information, the location (X-Y coordination) and the width information of each text piece are two important parameters. We have tried three different methods to detect the table column: 1) analyzing the relative positions among the text pieces which follow the original sequence, 2) projecting

all the texts into the X-axis to detect the space between columns, and 3) counting the maximum text pieces in each line. After comparing the performance of all these three methods, we fine that the last one outperforms the other two. However, for some unknown reasons, the text parsing tools always cut a text piece into several small ones. In order to get the accurate text pieces, we need to check each line and do some text merging if needed. Currently, the main reason for the errors in this step is the width error inherited from PDFBOX and TET. Among 123 tables, 7 of them have minor errors.

11.6 Error analysis of the table heading detection

After we get the number of the table column, we treat the first line in the detected table boundary as the beginning of the table heading until the first line that all the columns are occupied. Such method may incur errors. For example in Figure 11.2, we will stop at the first row because all the columns are occupied. However, the second line still belongs to the column heading. Although with some semantic knowledge, we can fix a part of them, it is difficult to fix it all. Among 123 tables, 7 of them have such minor errors.

11.7 Error analysis of the row detection

Because there is no ruling lines to help us to detect separate the rows from each other, and there is no much distance difference between the rows and the lines within a cell, to detect the row information, we use this rule: after we detect all the table columns, we count the number of rows in the first column. We treat this number as the row number.

Table 7 Effect of EDTA on nitrate reduction at a copper electrode (all potentials *versus* SCE). Scan speed, 10 mV s^{-1}; results are the average of three scans

[EDTA]/ mg l^{-1}	Peak potential/V	Peak current/μA
0.00	−0.48	175
4.95	−0.57	186
9.80	−0.57	190
19.2	−0.58	186

Fig. 11.2. An example table with minor column heading error

11.8 Other sources of error

In some PDF documents, tables or some table columns are created as embedded images. In the current state, *TableSeer* only processes the text-based tables and simply filters out the image tables. For the image tables, we can detect the table caption, but no table content. For the tables with some image columns, we can still detect all the other text metadata. In Figure 11.3, all the contents in the first column are missing, therefore, TableSeer can think the column number is 2. Figure 11.4 is another example, however, because all the column headings are text-based, TableSeer still can get the correct column number information.

Some documents have severe sequence disorder in the text lines among not only the table boundary, but also the whole document page. The text parsing tools may extract a small part of the table lines then jump out to continue some other figures, or finish a whole document column first. After that, the text parsing tools may go back and continue the text parse of the remaining table texts, or omit the rest of the table texts.

Table 1 Influence of concomitant ions on the peak current of copper(II). Accumulation on a CPE containing 2% DQDS at −0.05 V for 5 min in 0.1 mol dm^{-3} acetate buffer (pH 4.5) containing 4×10^{-7} mol dm^{-3} copper(II)

Ion	Concentration of concomitant ion/ μmol dm^{-3}	Signal ratio (%)
Ag^{I}	20	100
	40	99
Bi^{III}	20	95
	40	89
Co^{II}	20	103
	40	99
Fe^{III}	20	102
	40	101
Hg^{II}	20	98
	40	99
Pt^{IV}	20	102
	40	97
Ru^{III}	20	100
	40	100
Sb^{III}	20	102
	40	102
Se^{IV}	20	97
	40	98
Zn^{II}	20	103
	40	100

Fig. 11.3. An example table with the missing column

Table 2 Effect of humic acid on the determination of iron(III)

Amount of iron(III)/μg	Humic acid added*/μg	Iron found/μg	Number of experiments
2.00	14.2	2.06	2
	42.6	2.00	2
	71.0	2.08	2
		Mean 2.05	6
		SD† 0.21	
5.00	14.2	5.10	3
	42.6	5.11	3
	71.0	5.02	2
		Mean 5.08	8
		SD 0.09	

* Humic acid from Wako.
† SD = standard deviation.

Fig. 11.4. Another example table with the missing column

Some large tables are rotated as the vertical tables. After checking the detailed parsed texts and the coordinate values for each text pieces, we notice that there are many location errors. Many of the width values of texts are less than zero. Among 123 testing tables, one table is the vertical table.

Some tables have huge distance between rows (see Figure 11.5). If the distance value is larger than the defined threshold, we may stop in the middle of the table. If we do not keep such a tight threshold, many noise lines will be included into the table boundary. Among 123 testing tables, 6 of them have such problem.

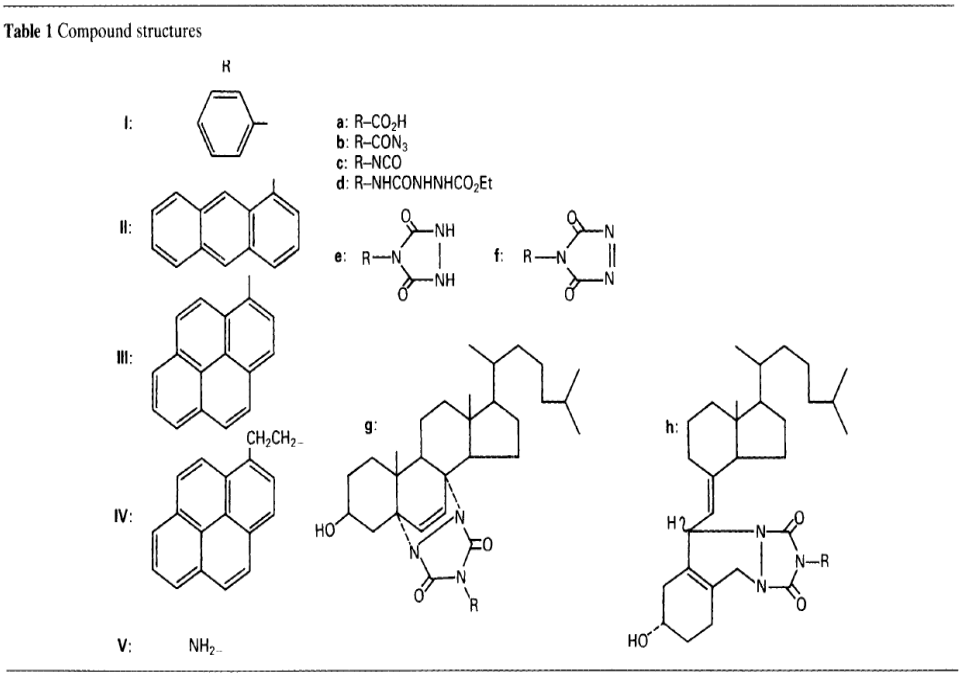


Fig. 11.5. An example table with large row gap

For some tables, it is difficult to identify the column headings and the row headings, even with the human judgement. Please see Figure 11.6.

Table 2 Conditions for the *in situ* derivatization of different alkyl halide samples

Primary alkyl iodide	1-Iodobutane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 90 min
Primary alkyl bromide	1-Bromobutane	Reflux with excess of thiourea in 30 ml of 95% ethanol for 150 min
	1-Bromopropane	
Primary alkyl chloride	1-Chlorobutane	Reflux with 3 equiv. of NaI in 40 ml of 95% ethanol for 24 h, then reflux with excess of thiourea for 150 min
Secondary alkyl bromide	2-Bromobutane	

Fig. 11.6. An example table that is difficult to identify the column headings and the row headings

Chapter 12

Conclusion

To facilitate table extracting and searching, we have devised a novel but efficient table-specific search engine, *TableSeer*. An extensive set of table metadata is proposed to precisely represent a table. Because the traditional TF-IDF approach is no longer suitable for table search, we also study how to calculate the ranking scores of tables contained in scientific documents based on multiple ranking factors from three levels: the term level, the table level, and the document level. We propose a novel table ranking algorithm, *TableRank*, which combines the query-dependent features with the query-independent features of a document to rank tables and their containing documents in response to an end-users query. Experimental results demonstrate that *TableSeer* outperforms the widely used search engine, e.g. *Google Scholar* on searching information contained in tables.

Comparing with the previous works on table extraction, we propose two new methods: the box-cutting method and the sparse-line based method to improve the table boundary detection, which is the vital step for the performance of the table metadata extraction and table structure decomposition. We implement both heuristic- and machine learning based methods. we also analyze a typical problem shared by the PDF text extraction tools: the text sequence error. We propose two algorithms to recover the sequence of extracted sparse lines, which improve the table content collection. The

experimental results not only compare the performance of both algorithms, but also demonstrate the effectiveness of text sequence recovering for the table boundary detection. The results show that the second algorithm achieves better results for both single-column table and cross-column table. It is proved that the text sequence recovering work plays a crucial role in the table boundary detection field. In order to have a better table understanding, we implement the first large-scaled table quantitative study on the table characterization. The detailed table distribution in multiple perspectives is observed.

Scientific documents are the focus of our current work. However, there are a large numbers of tables in documents in other areas. Current parameter settings are based on empirical results and further work is needed to establish optimal settings for ranking and searching. Future user studies will indicate the validity of the ranking methods. There are several future works I want to continue: 1) table classification semantically; 2) table similarity analysis and table distance measurement; 3) Designing table recommendation systems; 4) Applying tables into document summarization, classification, and clustering; 5) improving the performance of the table metadata extraction and ranking algorithm; 6) Extending the table work to other document components, such as Mathematical equations, biomedical and health information, chemistry formula, figures, etc.

Bibliography

[1]

[2] http:/en.wikipedia.orgwikisearch_engine.

[3] http://lucene.apache.org/java/docs/index.html.

[4] N. Asada A. Amano. Graph grammar based analysis system of complex table form document. In *International Conference on Document Analysis and Recognition (ICDAR)*, pages 916–920, 2003.

[5] C. Buckley A. Singhal and M. Mitra. Pivoted document length normalization. In *Proc. 19th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval, ACM Press*, page 21C29, 1996.

[6] Eugene Agichtein, Eric Brill, and Susan T. Dumais. Improving web search ranking by incorporating user behavior information. In *SIGIR*, pages 19–26, 2006.

[7] A. Anjewierden. Aidas: Incremental logical structure discovery in pdf documents, 2001.

[8] Kemafor Anyanwu, Angela Maduko, and Amit P. Sheth. Semrank: ranking complex relationship search results on the semantic web. In *WWW*, pages 117–127, 2005.

[9] W. Gatterbauer B. Krupl, M. Herzog. Using visual cues for extraction of tabular data from arbitrary html documents. In *In Proc. of the 14th Int'l Conf. on World Wide Web*, pages 1000–1001, 2005.

[10] Ricardo Baeza-Yates and Berthier Ribeiro-Neto. Modern information retrieval. In *ACM Press/Addison-Wesley*, 1999.

[11] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web*, pages 107–117, 1998.

[12] Christopher J. C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2(2):121–167, 1998.

[13] R. Cattoni, T. Coianiz, S. Messelodi, and C.M.Modena. Geometric layout analyis techniques for document image understanding: a review. *ITC-IRST Technical Report 9703-09.*

[14] Francesca Cesarini, Simone Marinai, L. Sarti, and Giovanni Soda. Trainable table location in document images. In *ICPR (3)*, pages 236–240, 2002.

[15] Soumen Chakrabarti, Kriti Puniyani, and Sujatha Das. Optimizing scoring functions and indexes for proximity search in type-annotated corpora. In *WWW*, pages 717–726, 2006.

[16] S. Chandran, S. Balasubramanian, T. Gandhi, A. Prasad, R. Kasturi, and A.K. Chhabra. Structure recognition and information extraction from tabular documents. 7(4):289–303, 1996.

[17] J.S. Chen and D.C. Tseng. Overlapped-character separation and reconstruction for table-form documents. pages II: 233–236, 1996.

[18] S. Tsai H. Chen and J. Tsai. Mining tables from large scale html texts. In *In Proc. 18th Int'l Conf. Computational Liguistics, Saarbrucken, Germany*, 2000.

[19] Wenyuan Wang Wei-Ying Ma Chong Wang, Xing Xie. Improving web browsing on small devices based on table classification. In *Advances in Multimedia Information Processing - PCM*, pages 88–95, 2004.

[20] M.; Cohen, W. W.; Hurst and L. S. Jensen. A flexible learning system for wrapping tables and lists in html documents. In *In Proc. of the WWW conference, ACM*, pages 232–241, 2002.

[21] X. Wei W. Bruce D. Pinto, A. McCallum. Table extraction using conditional random fields. In *In proceeding of Proceedings of the 26th ACM SIGIR, Toronto, Canada*, July 2003.

[22] Pavel A. Dmitriev, Nadav Eiron, Marcus Fontoura, and Eugene J. Shekita. Using annotations in enterprise search. In *WWW*, pages 811–817, 2006.

[23] D.W. Embley, D.P. Lopresti, and G. Nagy. Notes on contemporary table recognition. pages 164–175, 2006.

[24] Guang Feng, Tie-Yan Liu, Ying Wang, Ying Bao, Zhiming Ma, Xu-Dong Zhang, and Wei-Ying Ma. Aggregaterank: bringing order to web sites. In *SIGIR*, pages 75–82, 2006.

[25] Miriam Fernández, David Vallet, and Pablo Castells. Using historical data to enhance rank aggregation. In *SIGIR*, pages 643–644, 2006.

[26] H. Luo G. Penn, J. Hu and R. McDonald. Flexible web document analysis for delivery to narrow-bandwidth devices. In *In International Conference on Document Analysis and Recognition (ICDAR01), Seattle, WA, USA*, September 2001.

[27] C. Buckley G. Salton. Term-weighting approaches in automatic text retrieval. In *Information Processing and Management 24(5)*, pages 513–523, 1988.

[28] Basilios Gatos, Dimitrios Danatsas, Ioannis Pratikakis, and Stavros J. Perantonis. Automatic table detection in document images. In *ICAPR (1)*, pages 609–618, 2005.

[29] Wolfgang Gatterbauer and Paul Bohunsky. Table extraction using spatial reasoning on the CSS2 visual box model. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI 2006)*. AAAI, MIT Press, July 2006.

[30] Edward A. Green and Mukkai S. Krishnamoorthy. Model-based analysis of printed tables. In *ICDAR*, pages 214–217, 1995.

[31] Lin Guo, Feng Shao, Chavdar Botev, and Jayavel Shanmugasundaram. Xrank: Ranked keyword search over xml documents. In *SIGMOD Conference*, pages 16–27, 2003.

[32] Tamir Hassan and Robert Baumgartner. Table recognition and understanding from pdf files. In *ICDAR*, pages 1143–1147, 2007.

[33] Yuki Hirayama. A method for table structure analysis using dp matching. In *ICDAR*, pages 583–586, 1995.

[34] H.R.Stabler. Experimences with high-volume, high-accuracy document capture. In *In A. L. Spitz and A. Dengel (Eds.), IAPR 1994 Workshop on Document Analysis Systems*, pages Vol. 14, Series in Machine Perception and AIm World Scientific, 38–51, Singapore, 1995.

[35] Hsin hsi Chen, Shih chung Tsai, and Jin he Tsai. Mining tables from large scale html texts. In *In Proceedings of the 18th International Conference on Computational Linguistics (COLING00*, pages 166–172, 2000.

[36] J. T. Koo H.T. Ng, C. Y. Lim. Learning to recognize tables in free text. In *In Proc. of the 37th Annual Meeting of the Association of Computational Linguistics on Computational Linguistics*, pages 443–450, 1999.

[37] M. Hurst. The interpretation of tables in texts. In *PhD thesis, University of Edinburgh, School of Cognitive Science, Informatics*, 2000.

[38] M. Hurst. Layout and language: Challenges for table understanding on the web, 2001.

[39] M. Hurst. Layout and language: Challenges for table understanding on the web. In *In Web Document Analysis, Proceedings of the 1st International Workshop on Web Document Analysis, Seattle,WA, USA*, September 2001.

[40] Matthew Hurst. Towards a theory of tables. *IJDAR*, 8(2-3):123–131, 2006.

[41] Heasoo Hwang, Vagelis Hristidis, and Yannis Papakonstantinou. Objectrank: a system for authority-based search on databases. In *SIGMOD Conference*, pages 796–798, 2006.

[42] K. Itonori. Table structure recognition based on textblock arrangement and ruled line position. In *Prc., IAPR 2nd International Conference on Document Analysis and Recognition, Tsukuba Science City, Japen*, pages 765–768, October 1993.

[43] R.M. Haralick J. Ha and I.T. Philips. Recursive x-y cut using bounding boxes of connected components. In *In Proc. Third Int'l Conf. Document Analysis and Recognition*, pages 952–955, 1955.

[44] D. Lopresti J. Hu, R. Kashi and G. Wilfong. Experiments in table recognition. In *In Proc. Workshop on Document Layout Interpretation and Applications, Seattle, Washington*, 2001.

[45] D. Lopresti J. Hu, R. Kashi and G. Wilfong. Medium-independent table detection. In *SPIE Document Recognition and Retrieval VII, San Jose, California*, pages 291–302, January 2000.

[46] D. Lopresti J. Hu, R. Kashi and G. Wilfong. Table structure recognition and its evaluation. In *SPIE Document Recognition and Retrieval VIII, San Jose, California*, January 2001.

[47] N. Guerette J. Shin. Table recognition and evaluation. In *In Proc. of the Class of 2005 Senior Conference, Computer Science Department, Swarthmore College.*, pages 8–13, 2005.

[48] N. Guerette J. Shin. Table recognition and evaluation. In *In Proc. of the Class of 2005 Senior Conf., Computer Science Department, Swarthmore College*, pages 8–13, 2005.

[49] T.L Wood J.H. Shamilian, H.S. Baird. A retargetable table reader. In *In Proc. of the 4th Int'l Conf. on Document Analysis and Recognition*, pages 158–163, 1997.

[50] T.L Wood J.H. Shamilian, H.S. Baird. A retargetable table reader. In *In Proc. of the 4th Int'l Conf. on Document Analysis and Recognition*, pages 158–163, 1997.

[51] T. G. Kieninger. Table structure recognition based on robust block segmentation. In *In Proc. Document Recognition V, SPIE, volume 3305*, pages 22–32, January 1998.

[52] B. Kruplupl and M Herzog. Visually guided bottom-up table detection and segmentation in web documents. In *In Poster Proc. WWW06. ACM*, 2006.

[53] Oren Kurland and Lillian Lee. Pagerank without hyperlinks: structural re-ranking using links induced by language models. In *SIGIR*, pages 306–313, 2005.

[54] John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th ICML*, pages 282–289. Morgan Kaufmann, San Francisco, CA, 2001.

[55] A. Laurentini and P. Viada. Identifying and understanding tabular material in compound documents. pages II:405–409, 1992.

[56] Ying Liu, Kun Bai, Prasenjit Mitra, and C. Lee Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *JCDL*, pages 91–100, 2007.

[57] 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 *DAS*, 2008.

[58] Ying Liu, Prasenjit Mitra, and C. Lee Giles. Identifying table boundaries in digital documents via sparse line detection. In *CIKM*, pages 1311–1320, 2008.

[59] Ying Liu, Prasenjit Mitra, C. Lee Giles, and Kun Bai. Automatic extraction of table metadata from digital documents. In *JCDL*, pages 339–340, 2006.

[60] Vanessa Long, Robert Dale, and Steve Cassidy. A model for detecting and merging vertically spanned table cells in plain text documents. In *ICDAR*, pages 1242–1246, 2005.

[61] Daniel P. Lopresti and George Nagy. A tabular survey of automated table processing. In *GREC*, pages 93–120, 1999.

[62] S. Mandal, S. P. Chowdhury, A. K. Das, and Bhabatosh Chanda. A simple and effective table detection system from document images. *IJDAR*, 8(2-3):172–182, 2006.

[63] Harendra Guturu Alex Ksikes Preslav Nakov Michael A. Wooldridge Marti A. Hearst, Anna Divoli and Jerry Ye. Biotext search engine: beyond abstract searchex. In *Bioinformatics 23(16):2196-2197*, 2007.

[64] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, 2003.

[65] Andrew McCallum. Efficiently inducing features of conditional random fields. In *Nineteenth Conference on UAI*, 2003.

[66] Zaiqing Nie, Yuanzhi Zhang, Ji-Rong Wen, and Wei-Ying Ma. Object-level ranking: bringing order to web objects. In *WWW*, pages 567–574, 2005.

[67] Chang C.H. Alam H. Peterman, C. A system for table understanding. In *Proceedings of the Symposium on Document Image Understanding Technology (SDIUM'97)*, pages 55–62, Annapolis, MD, 1997.

[68] D. Pinto, W. Croft, M. Branstein, R. Coleman, M. King, W. Li, and X. Wei. Quasm: A system for question answering using semi-structured data, 2002.

[69] David Pinto, Andrew McCallum, Xing Wei, and W. Bruce Croft. Table extraction using conditional random fields. In *DG.O*, 2003.

[70] Aleksander Pivk, Philipp Cimiano, Er Pivk, and York Sure. From tables to frames. In *Journal of Web Semantics*, pages 166–181, 2004.

[71] J. M. Ponte and W. B. Croft. A language modeling approach to information retrieval. In *Proc. 21st Annual International ACM SIGIR Conference on Research and Development in*

Information Retrieval, Melbourne, Australia, August 1998. ACM Press, New York., pages 275–281, 1998.

[72] Bhanu Prasad, editor. *Proceedings of the 2nd Indian International Conference on Artificial Intelligence, Pune, India, December 20-22, 2005*. IICAI, 2005.

[73] P. Pyreddy and W. Croft. Tintin: A system for retrieval in text tables. In *In Proceedings of the Second International Conference on Digital Libraries*, pages 193–200, 1997.

[74] R. Kumar R. Fagin and D. Sivakumar. Comparing top k lists. In *SIAM Journal on Discrete Mathematics*, pages 134–160, 2003.

[75] D. Blostein R. Zanibbi and J.R. Cordy. A survey of table recognition: Models, observations, transformations, and inferences. In *Int'l J. Document Analysis and Recognition, Vol. 7, No.1*, pages 1–16, 2004.

[76] T. Gandhi A. Prasad R. Kasturi A. Chhabra S. Chandran, S. Balasubramanian. Structure recognition and information extraction from tabular documents. In *In Int'l Journal of Imaging Systems and Technology, Vol 7*, pages 289–303, 1998.

[77] S.Walker S. E. Robertson and M. Beaulieu. Okapi at trecc7: automatic ad hoc, filtering, vlc and filtering tracks. In *In Voorhees and Harman, NIST Special Publication SP 500-242*, page 253C261, 1998.

[78] A. K. Das Bhabatosh Chanda S. Mandal, S. P. Chowdhury. Detection and segmentation of tables and math-zones from document images. In *Symposium on Applied Computing archive, Proceedings of the 2006 ACM symposium on Applied computing table of contents, Dijon, France*, pages 841–846, 2006.

[79] S.R. Safavian and D.A. Landgrebe. A survey of decision tree classifier methodology. In *SMC(21), No. 3, May 1991, pp. 660-674.*

[80] C. Casado M.M. Sandusky, R.J. Tenopir. Uses of figures and tables from scholarly journal articles in teaching and research. In *In Proceedings of the 70th Annual Meeting of the American Society for Information Science & Technology*, pages 1–13, Milwaukee, WI, 2007.

[81] Lawrence Page Sergey Brin. The anatomy of a large-scale hypertextual web search engine. In *Proceedings of the seventh international conference on World Wide Web 7*, pages 107–117, 1999.

[82] F. Sha and F. Pereira. Shallow parsing with conditional random fields, 2003.

[83] J. Shamilian, H. Baird, and T. Wood. A retargetable table reader, 1997.

[84] L. Shih and D. Karger. Using urls and table layout for web classification tasks, 2004.

[85] A. Dengel T. Kieninger. Applying the t-rec table recognition system to the business letter domain. In *In Proc. of the 6th Int'l Conf. on Document Analysis and Recognition*, pages 518–522, September 2001.

[86] Shinji Tsuruoka, Toru Tanaka, Tomohiro Yoshikawa, Tsuyoshi Shinogi, and Kensuke Takao. Region segmentation for table image with unknown complex structure. In *ICDAR*, pages 709–, 2001.

[87] S. Tupaj, Z. Shi, and D. Chang. Extracting tabular information from text files, 1996.

[88] J. Wang and J. Hu. A machine learning based approach for table detection on the web. In *Proceedings of the 11th Int'l Conf. on World Wide Web (WWW'02)*, pages 242–250, Nov 2002.

[89] X. Wang. Tabular abstraction, editing, and formatting. In *Ph.D. Thesis, Dept. of Computer Science, University of Waterloo*, 1996.

[90] Y. Wang and J. Hu. Detecting tables in html documents, 2002.

[91] Yalin Wang, Robert M. Haralick, and Ihsin T. Phillips. Statistical-based approach to word segmentation. In *ICPR*, pages 4555–4558, 2000.

[92] Yalin Wang and Jianying Hu. A machine learning based approach for table detection on the web. In *WWW*, pages 242–250, 2002.

[93] Howard C. Wasserman, Keitaro Yukawa, Bon K. Sy, Kui-Lam Kwok, and Ihsin T. Phillips. A theoretical foundation and a method for document table structure extraction and decompositon. In *Document Analysis Systems*, pages 291–294, 2002.

[94] Toyohide Watanabe, Qin Luo, and Noboru Sugie. Layout recognition of multi-kinds of table-form documents. *IEEE Trans. Pattern Anal. Mach. Intell.*, 17(4):432–445, 1995.

[95] Claudia Wenzel and Wolfgang Tersteegen. Precise table recognition by making use of reference tables. In *Document Analysis Systems*, pages 283–294, 1998.

[96] J. Hu Y. Wang. Detecting tables in html documents. In *In Proc. of the 5th IAPR Int'l Workshop on Document Analysis Systems, Princeton, NJ*, 2002.

[97] R. Haralick Y. Wang, I.T. Philips. Automatic table ground truth generation and a background-analysis-based table structure extraction method. In *In Proc. of the 6th Int'l Conference on Document Analysis and Recognition*, page 528, September 2001.

[98] R. M. Haralick Y. Wang, L.T. Phillips. Table structure understanding and its performance evaluation. In *Pattern Recognition, 37(7)*, pages 1479–1497, July 2004.

[99] B. Yildiz, K. Kaiser, and S. Miksch. pdf2table: A method to extract table information from pdf files. *IICAI05*, (Pune, India), 2005.

[100] M. Yoshida and H. Nakagawa. Web document parsing: A new approach to modeling layout-language relations. pages 203–207, 2007.

[101] M. Yoshida, K. Torisawa, and J. Tsujii. A method to integrate tables of the world wide web, 2001.

[102] Minoru Yoshida and Kentaro Torisawa. A method to integrate tables of the world wide web. In *In Proceedings of the International Workshop on Web Document Analysis (WDA 2001*, pages 31–34, 2001.

[103] Yefeng Zheng, Changsong Liu, Xiaoqing Ding, and Shiyan Pan. Form frame line detection with directional single-connected chain. In *ICDAR*, pages 699–703, 2001.

[104] Zijian Zheng. Naive bayesian classifier committees. In *European Conference on Machine Learning*, pages 196–207, 1998.

[105] Konstantin Zuyev. Table image segmentation. In *ICDAR*, pages 705–708, 1997.

Vita

Ying Liu

Education

The Pennsylvania State University State College, Pennsylvania 2005–Present

Ph.D. in Information Sciences and Technology, expected in Dec. 2009

Area of Specialization: Information Retrieval

University of Alberta Edmonton, Alberta, Canada 2001–2005

M.S. in Computer Science

Area of Specialization: Software Engineering

Hefei University of Technology Hefei, Anhui, P. R. China 1994–1998

B.S. in Computer Science

Research Experience

Doctoral Research The Pennsylvania State University 2005–Present

Thesis Advisor: Prof. C. Lee Giles, Prasenjit Mitra

Graduate Research The University of Alberta, Canada 2001–2005

Research Advisor: Prof. Eleni Stroulia, Kenny Wong

Teaching Experience

Teaching Assistant The Pennsylvania State University 2005, 2006

I taught labs and graded assignments and exams.

Teaching Assistant University of Alberta, Canada 2001–2004

I taught programming courses and software engineering courses.