TABLE RECOGNITION IN MATHEMATICAL DOCUMENTS

by

MOHAMED A. ALKALAI

A thesis submitted to
The University of Birmingham
for the degree of
DOCTOR OF PHILOSOPHY

School of Computer Science
College of Engineering and Physical Sciences
The University of Birmingham
November 2015
Abstract

While a number of techniques have been developed for table recognition in ordinary text documents, when dealing with tables in mathematical documents these techniques are often ineffective as tables containing mathematical structures can differ quite significantly from ordinary text tables. In fact, it is even difficult to clearly distinguish table recognition in mathematics from layout analysis of mathematical formulas. Again, it is not straightforward to adapt general layout analysis techniques for mathematical formulas. However, a reliable understanding of formula layout is often a necessary prerequisite to further semantic interpretation of the represented formulae.

In this thesis, we present the necessary preprocessing steps towards a table recognition technique that specialises on tables in mathematical documents. It is based on our novel robust line recognition technique for mathematical expressions, which is fully independent of understanding the content or specialist fonts of expressions.

We also present a graph representation for complex mathematical table structures. A set of rewriting rules applied to the graph allows for reliable re-composition of cells in order to identify several valid table interpretations. We demonstrate the effectiveness of our technique by applying them to a set of mathematical tables from standard text book that has been manually ground-truthed.
ACKNOWLEDGEMENTS

I first would like to express my appreciation to my supervisors, Dr. Mark Lee and Prof. Jonathan Rowe for supporting and encouraging me. Completing this thesis would not have been possible without their guidance.

I also wish to thank my parents for believing in me and encouraging me reach my goals. Indeed, life would be very difficult without their unlimited support.

I would also like to express the honest appreciation to my wife, Ghada, who stands behind me and whose her help allowed me to accomplish this task, already has my heart so I will just give her a heartfelt “thanks.”

Finally, to my sons and daughter, Ali, Omar, Abdalla and Amna who make my life easier by their innocent smiles.
CONTENTS

1 Introduction ................................................. 1
   1.1 Interpretations of Mathematical Table ................. 2
   1.2 Research questions .................................... 4
   1.3 Contributions ......................................... 5
   1.4 Publications .......................................... 6
   1.5 Overview of Thesis .................................... 7

I Background .................................................. 9

2 Literature Review ........................................... 10
   2.1 Issues in table structure recognition ................. 11
   2.2 Document structural analysis ........................... 11
   2.3 Table analysis ......................................... 12
   2.4 Methods used in table syntactic recognition ....... 13
   2.5 Standard line/cell segmentation techniques ........ 14
      2.5.1 Smearing approaches ............................. 14
      2.5.2 Projection Profile Cuts (PPC) .................... 15
      2.5.3 Grouping approaches ............................. 18
      2.5.4 Hough transform approaches .................... 19
   2.6 Mathematical expressions segmentation ............. 20
      2.6.1 Mathematical expression segmentation in images 20
      2.6.2 Mathematical expression segmentation in PDF files 21
2.7 Unsolved issues concerning the detection of lines containing mathematical expressions ................................................. 22
2.8 Existing table segmentation methods .......................................................... 22
2.8.1 Projection-profile approaches .............................................................. 22
2.8.2 Heuristic approaches ........................................................................... 23
2.8.3 Grouping approaches ........................................................................... 24
2.8.4 Advanced approaches ......................................................................... 25
2.9 Existing table segmentation techniques for PDF/Postscript ..................... 26
2.10 Unsolved issues concerning the segmentation of tables into cells ............. 27
2.11 Table representation techniques ............................................................... 28
2.12 Models to represent tables ....................................................................... 29
2.12.1 Tree models ...................................................................................... 29
2.12.2 Dual hierarchy models ...................................................................... 30
2.12.3 The CALS model ............................................................................. 31
2.12.4 Graph model: Amano and Asada’s model ........................................... 32
2.13 Unsolved issues concerning the representation of table structure ........... 35
2.14 Summary ................................................................................................. 36

II Preprocessing Steps Toward A Robust Mathematical Table Recognition 37

3 A Novel Histogrammatic Approach to Line Segmentation 38
3.1 The approach at a glance ......................................................................... 39
3.2 Experimental Results and Discussion ...................................................... 55
3.2.1 Incorrect Non-principal Lines: ............................................................ 61
3.2.2 Incorrect Principal Lines: ................................................................. 61
3.3 Evaluation of integrating my line segmentation technique with MaxTract 62
3.4 Summary ................................................................................................. 64
4 Towards Full Cell Recognition

4.1 Table Column Extraction
4.1.1 Example
4.1.2 Determining $V$ value

4.2 Evaluation and Experimental Results

4.3 Summary

III Table Recognition

5 Recognising Tabular Mathematical Expressions using Graph Rewriting

5.1 Multi-Interpretations of Table’s Re-composition
5.1.1 Preprocessing Steps
5.1.2 Tabular Representation Using Graph Model
5.1.3 Construct Initial Graph (Example)

6 First Method: Direct rewriting of the initial graph

6.1 Graph rewriting system
6.1.1 Table Production Rules
6.1.2 Number of rules needed to cover any node combinations
6.1.3 Recognizing Rows Using Virtual Nodes

6.2 A Framework For Choosing Column Combinations
6.2.1 Number of possible column combinations

6.3 Summary

7 Evaluation of the first tabular recognition method

7.1 Applying the rewriting system on a table

7.2 Experimental Results
7.2.1 Analysis of table interpretations that are partially extracted or missed
7.2.2 Table Re-composition: within documents
7.2.3 Table Re-composition: combining tables
8 Second Method: Pre-process rewriting on the initial graph

8.1 Introduction

8.2 Types of spatial relationships between cells

8.2.1 Experiments

8.3 Graph Rewriting System

8.3.1 Graph representation model

8.3.2 Graph rewriting rules

8.4 Table structure analysis

8.4.1 Apply these rules (example)

8.5 Summary

9 Evaluation of the second tabular recognition method

9.1 More experiments

9.1.1 No need of constraints

9.1.2 Experimental results

9.2 Summary

10 Conclusions and future work

10.1 Contributions

10.1.1 Mathematical line segmentation

10.1.2 Cell segmentation

10.1.3 First table interpretation method

10.1.4 Second table interpretation method

10.2 Future Work

10.2.1 Improving cells segmentation technique

10.2.2 Automatic selecting of rewriting rules for different interpretations of table structure
IV  Appendixes  143

.0.3  An example of constraints associate with each rule  . . . . . . . . 155

List of References  178
1.1 Cell identification with tables (taken from [Grad 07]) containing multi-line expressions. ................................................................. 3
1.2 Two different interpretations of a single table (taken from [Grad 07]). . . . 4

2.1 An example of Smearing’s drawback. .................................................. 16
2.2 A Statistical table. .............................................................................. 17
2.3 An example of PPC applied to the table in figure 2.2 ......................... 18
2.4 An example of PPC’s drawback. ........................................................... 19
2.5 Examples of displayed and embedded mathematical expressions which is taken from [Stro 11] ................................................................. 21
2.6 A table (taken from [Silv 06]) most segmentation algorithms would not process accurately ........................................................................... 28
2.7 Table used for describing the work of the x-y cut algorithm ................. 30
2.8 The table cells in Figure 2.7 represented by x-y tree .......................... 30
2.9 Example of table used in illustrating CALS representation model .......... 32
2.10 Cells that are totally and vertically overlapped ................................. 33
2.11 Cell’s y-axis borders within other cell’s y-axis borders ...................... 33
2.12 Cells that are totally and horizontally overlapped ............................. 34
2.13 Cell’s x-axis borders within other cell’s x-axis borders .................... 34
2.14 Cells that are partially and horizontally overlapped ........................... 35
2.15 Example for table representation model (taken from [Grad 07]) .......... 35
2.16 The resulted graph representation of figure 2.15 ............................. 36
<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Examples (taken from [Grad 07]) of glyph’s bounding boxes and vertical and horizontal overlaps.</td>
<td>41</td>
</tr>
<tr>
<td>3.2</td>
<td>Examples of a line (taken from [Grad 07])</td>
<td>42</td>
</tr>
<tr>
<td>3.3</td>
<td>Examples of different types of lines overlapping (taken from [Grad 07])</td>
<td>43</td>
</tr>
<tr>
<td>3.4</td>
<td>Example of horizontal overlap glyphs that both vertically overlap with one glyph</td>
<td>44</td>
</tr>
<tr>
<td>3.5</td>
<td>Example of horizontal overlap glyphs that are conveyed one symbol</td>
<td>44</td>
</tr>
<tr>
<td>3.6</td>
<td>(a) illustrates an example (taken from [Grad 07]) of compound line contains two horizontal overlap lines and a separator. (b) shows the resulting two lines</td>
<td>46</td>
</tr>
<tr>
<td>3.7</td>
<td>Examples (taken from [Deni 11]) of pages and their histogram of the gap between glyphs</td>
<td>49</td>
</tr>
<tr>
<td>3.8</td>
<td>Examples of principal and non-principal lines (taken from [Grad 07])</td>
<td>49</td>
</tr>
<tr>
<td>3.9</td>
<td>An example of lines (taken from [Deni 11]) are wrongly classified as non-principal</td>
<td>51</td>
</tr>
<tr>
<td>3.10</td>
<td>An example of lines (taken from [Deni 11]) are wrongly classified as principal</td>
<td>51</td>
</tr>
<tr>
<td>3.11</td>
<td>An example of merging lines process (taken from [Deni 11])</td>
<td>55</td>
</tr>
<tr>
<td>3.12</td>
<td>The results of running our technique over 200 pages with different (T)</td>
<td>58</td>
</tr>
<tr>
<td>4.1</td>
<td>Tables (taken from [Grad 07]) with mathematical expressions before applying the Kieninger’s algorithm.</td>
<td>66</td>
</tr>
<tr>
<td>4.2</td>
<td>The results of applying Kieninger’s algorithm.</td>
<td>67</td>
</tr>
<tr>
<td>4.3</td>
<td>The results of running our technique over 200 pages with different (V)</td>
<td>70</td>
</tr>
<tr>
<td>4.4</td>
<td>Ideal segmentation output of both table 1 and table 2 in figure (4.1)</td>
<td>72</td>
</tr>
<tr>
<td>4.5</td>
<td>An example of a classified page from [Grad 07].</td>
<td>73</td>
</tr>
<tr>
<td>4.6</td>
<td>examples of a case of errors in (a) and a case of correctness in (b)</td>
<td>75</td>
</tr>
<tr>
<td>5.1</td>
<td>Cell identification with table containing multiline expressions that is taken from [Grad 07].</td>
<td>80</td>
</tr>
</tbody>
</table>
7.9 (a) Isolated table (taken from [Grad 07]), (b) The same table but combined with other table

8.1 Different arrows represent different relationships between cells

8.2 Cells that are internally and vertically overlapped

8.3 Cells that are totally and vertically overlapped

8.4 Cells that are centrally and vertically overlapped

8.5 Cells that are partially and vertically overlapped

8.6 Cells that are sided and vertically overlapped

8.7 Cells that are internally and horizontally overlapped

8.8 Cells that are fully and horizontally overlapped

8.9 Cells that are centrally and horizontally overlapped

8.10 Cells that are partially and horizontally overlapped

8.11 Cells that are sided and horizontally overlapped

8.12 Example for table representation model

8.13 The graph representation of the table in figure 8.12

8.14 Rule One

8.15 Rule Two

8.16 Rule Three

8.17 An example of columns that have not been correctly split

8.18 How the rewriting Process starts

8.19 The targeted nodes for rewriting

8.20 After the first rewriting took place

8.21 The targeted nodes for the second rewriting

8.22 After the second rewriting took place

8.23 After the final rewriting took place

9.1 First method: After the final rewriting took place

9.2 Second method: After the final rewriting took place
10.1 An example of table which is taken from [Grad 07] shows failure of cells.

2 Rule Four ................................................................. 145
3 Rule Five ................................................................. 147
4 Rule Six ................................................................. 149
5 Rule Seven .............................................................. 151
6 Rule Eight ............................................................... 152
7 Rule Nine ............................................................... 154
8 Table which is taken from [Grad 07]: example ........................................ 157
9 A graph represents table which is taken from [Grad 07] .................................. 158
10 Possible interpretation of a table which is taken from [Grad 07] .................. 158
11 First rewriting of the graph that represents a table which is taken from [Grad 07] 159
12 Final result of rewriting the graph that represents a table which is taken from [Grad 07] 160
# LIST OF TABLES

3.1 Results for vertical splitting. ........................................ 57
3.2 Experimental results for line recognition. ............................ 59
3.3 Experimental results of 1000 pages. .................................. 60
3.4 Evaluation results of 1000 pages. .................................... 60
3.5 Comparison of previously generated and new \LaTeX .................. 63

4.1 Experimental results for cell recognition. ............................ 74
4.2 Experimental results of running our technique over 994 pages .... 74

5.1 Type of nodes. .......................................................... 85

6.1 The binary representation and their corresponding $V^*$ and $R^*$. 92
6.2 All possible table column combinations .............................. 95
6.3 Final result of all possible table column combinations ............ 96

7.1 Results of applying the proposed table interpretation technique on 150 tables105
7.2 Results of applying the proposed table interpretation technique on 110 tables105
7.3 Table Re-composition: within documents ........................... 109
7.4 Table Re-composition: combining tables ............................ 109

8.1 Statistics of horizontal overlap relationships ....................... 117
8.2 Statistics of vertical overlap relationships ........................ 117

9.1 Results of applying the proposed table interpretation technique on 150 tables134
9.2 Results of applying the proposed table interpretation technique on 110 tables134
9.3 Results of applying the proposed table interpretation technique on 150 ICDAR tables...
CHAPTER 1

INTRODUCTION

Table recognition is a research case within document analysis that has attracted much attention over the years. Our aim is to develop a table recognition algorithm that is particularly good for tables containing mathematical expressions. As the distinction between tables and complex typeset mathematical formulas spanning multiple lines is often difficult, we forgo a narrow definition of table and instead consider a far wider range of expressions as tables than is usually the case in the literature [Grad 07].

There is no a standard convention of composing table. As a consequence, various structures of table are noticed in wide range of documents which include fully-bordered table, partially-bordered table and no-bordered table. Based on this fact, the proposed table recognition technique designed to deals with tables that have all of their ruling lines (if any) removed. So that, it can be applied on vast types of table.

The tabular form in which many mathematical expressions are presented can often lead to ambiguities in the interpretation of what essentially constitutes a table component (i.e., a column, row or cell). While in understanding of ordinary text tables the goal is generally to restrict a result to a single valid interpretation, for mathematical tables these ambiguities can lead to several perfectly acceptable interpretations. Therefore, the aim of our recognition procedure is to produce as a result a number of possible interpretations.
Our approach to the problem is in two steps: First we design an algorithm that enables the separation and detection of lines in mathematical documents. This is a non-trivial task since, contrary to ordinary text, simple line separation by vertical whitespace does not suffice, due to the two-dimensional nature of mathematical formulas. The procedure uses simple yet effective spatial means and serves as a basis of a cell detection algorithm. This algorithm builds on some of the same spatial measures as line detection.

In the second step, these two methods work as a basis of a table representation algorithm which ultimately lead to the full-fledged table recognition algorithm. In the next section we will discuss some examples to further motivate our problem. It is worth to mention that we do not use any character information or OCR information other than bounding boxes of connected components in any of the proposed techniques.

1.1 Interpretations of Mathematical Table

The boundary of what constitutes a mathematical table and what is a complex formula given in a tabular presentation is quite fluid. While some tables, for example Cayley tables in abstract algebra, are quite straightforward to recognise due to their easy tabular composition and sometimes clear separation of rows and columns with bars, this is generally not the case. In fact, the common absence of any vertical or horizontal bars as well as the complexity of formulas often spanning multiple lines make it not only difficult to identify the cell structure but can lead to a number of different interpretations for the same table, which are often equally valid.

Figure 1.1 presents two tables taken from [Grad 07] with a fairly conservative column and row layout. There is indeed a unique ideal interpretation for Table 1, consisting of two columns and three rows, where the cell in the lower right hand corner contains a
If $R(x)$ is A particular solution to $y'' + b^2 y = R(x)$ is

\[
\begin{align*}
\sin bx & \quad - \frac{x \cos bx}{2b} \\
P(x) \sin bx & \quad \frac{\sin bx}{(2b)^2} \left[ P(x) - \frac{P''(x)}{(2b)^2} + \frac{P^{(4)}(x)}{(2b)^4} + \ldots \right] \\
& \quad - \frac{\cos bx}{2b} \int \left[ P(x) - \frac{P''(x)}{(2b)^2} + \ldots \right] dx.
\end{align*}
\]

Table 1

| 1  | $p^m(x) = (-1)^m \left(1 - \frac{x^2}{2b^2}\right)^{\frac{m}{2}} \frac{d^m}{dx^m} P(x)$ | [IH, MO 84, EH I 143(6)] |
| 2  | $p^m(x) = (-1)^m \frac{\Gamma(n+1)}{\Gamma(n-m+1)} \left(1 - x^2\right)^{-\frac{n}{2}} \int_0^1 \ldots \int_0^1 P_n(x)(dx)^n$ | [m \geq 1] |
| 3  | $p^{-m}(x) = \left(z^2 - 1\right)^{-\frac{m}{2}} \int_0^1 \ldots \int_0^1 P_n(x)(dz)^m$ | [m \geq 1] |
| 4  | $Q^m(z) = \left(z^2 - 1\right)^{-\frac{m}{2}} \frac{d^m}{dz^m} Q_n(z)$ | [IH, MO 85, EH I 149(6)] |
| 5  | $Q^{-m}(z) = \left(z^2 - 1\right)^{-\frac{m}{2}} \int_0^\infty \ldots \int_0^\infty Q_n(z)(dz)^m$ | [m \geq 1] |

Table 2

Figure 1.1: Cell identification with tables (taken from [Grad 07]) containing multi-line expressions.

mathematical expression spanning two lines. And given the difference in font weights, one could even interpret the first line as a clear header row.

Table 2 in Fig. 1.1 is slightly less straightforward given the overlapping expressions in the second line. However, one can still come up with a unique interpretation of five rows and three columns. But it already becomes obvious that due to the overlap for formulas as well as the condition formulas which are effectively in a column of their own this interpretation is not so easy to obtain automatically.

Figure 1.2 presents a clipping from a more complex table also taken from [Grad 07]. Here we can already see two different interpretations, both with their own merits. While
both interpretations regard the basic table as consisting of four columns, the first interpretation results in three rows, using the formula names on the right hand side as a header column. The second interpretation on the other hand uses the enumeration in the first column as the header. Obviously there are still more interpretations: for example, one with three columns with the middle column containing complex formulas or even one with only two columns, where the right column contains named multi-line formulas that possibly can even be considered as subtables.

## 1.2 Research questions

Recognizing tables with mathematical expressions imposes several interesting research questions. For instance, mathematical expressions usually have a 2-dimensional structure which is sometimes spread over more than one line. This can lead current table segmentation methods for regular text to erroneous results by mistaking casual vertical space that appears within one formula as row delimiters. Therefore, observing to what extent the adaptation of existing table extraction techniques could handle tables with mathemati-
cal expressions is an interesting research question. However, due to the lack of sufficient formal definitions of the different components that compose these techniques, it is very difficult if not impossible to implement them [Zani 05]. Therefore, an analysis of these methods and inferring their shortcomings when they are to be applied on tables contain mathematical expressions and then develop techniques to deal with these shortcomings would be also an interesting research question on its own.

Another important research question concerns the fact that tables which can contain cells spanning multiple rows or columns as well might even contain a nested table in one of its cells. In tables containing mathematical expressions these distinctions can easily become blurred, which can lead to having more than one possible, equally legitimate, interpretation of the table form. Figure 1.2 shows two different interpretations of one table. This not only gives rise to the problem of finding a method that can yield a number of equally acceptable interpretations, but also the need to find an adequate grid structure to represent such tables holding the different interpretations and to give a means to re-compose the recognised cells. Therefore, developing a framework that is able to tackle this issue as well as the misaligned cells problem which commonly appears in such tables is another interesting question.

1.3 Contributions

A summary of the contributions of this thesis is as follows;

1. We describe a novel technique that reliably identifies lines in mathematical documents. This is challenging as standard line detection techniques for regular text using vertical cuts along consecutive whitespace break down in the light of mathematical expressions (e.g., limits of sums, integrals).
2. This forms the basis of a row and column segmentation method to produce the maximum number of identifiable cells.

3. We also describe two alternative novel methods to identify several acceptable table interpretations and a representation that will allow for the recomposition of cells.

4. We show an evaluation of the effectiveness of these techniques by putting together a ground truth set of representative documents for case studies.

1.4 Publications

This thesis is based partly upon the following conference and workshop publications:


1.5 Overview of Thesis

Chapter 1 gives an introduction to this thesis which contains a list of research questions, brief description of contributions and list of publications.

Part I is an overview of the techniques, algorithms and research related to the work in this thesis. In particular; chapter 2 is a review of the traditional approaches to the detection of plain text line/cell and also a discussion of current syntactic and semantic table recognition techniques and their limitations.

Part II presents a pre-processing approach toward a recognition technique of tables that contain mathematical structure. Chapter 3 details a novel histogrammatic algorithm for the detection of mathematical lines and also describes how this technique integrates with the Maxtract tool to improve its performance. Chapter 4 presents a technique to segment tables into cells. This technique, unlike other methods, has the ability to correctly segment tables containing mathematical expressions.

Part III presents two novel table representation techniques. Chapters 5 and 6 detail the first one which uses a graph, that represents cells and columns of table which are gained by applying heuristic technique, to interpret the table structure. Chapter 7 gives an evaluation of the first method. Chapter 8 presents the second table representation technique that uses the same graph, illustrated in chapter 5, but then extract some cells or columns (if any) that the heuristic technique of the first method fails to do so. In
chapter 9, a discussion about the results of running the technique from chapter 8 over two datasets, one of these is from a book [Grad 07] and the other is available online, is given.

Chapter 10, sums up the contributions of my research and suggests some future work on my findings.
Part I

Background
CHAPTER 2

LITERATURE REVIEW

Tables are very common objects for expressing information within documents in a concise manner. Recognising these tables is not a trivial problem in document structural analysis. As a result, a lot of techniques for understanding tables have been proposed by document analysis researchers. There are various formats of documents, for instance, document images, accessible text documents, PDF documents and web documents. However, we mainly discuss table structure understanding in image and PDF documents.

In order to extract information from tables, one should first come up with an approach for finding the table within a document. Although, it is easy for any human to recognise tables, robust methods for automatic recognition is difficult. Several observed features like symbols and layout might all be utilised in attempting to recognise tables. However, most of existing techniques suffer from the degradation in the accuracy rate of results when they are applied to various table datasets. Indeed, extracting the table’s structure as well as its context, is essential. This would contribute to have better markup and also give extra information which, in turn, pave the way to better document accessibility.
2.1 Issues in table structure recognition

With the rapid spread of electronic documents on the internet and the consequent increasing speed in the production of new documents in general, designing efficient algorithms for retrieving and storing the content of these documents becomes essential. In the last decades, document analysis experts have proposed many techniques for symbol recognition, page segmentation and recognising of documents containing regular texts. The majority of these algorithms were not developed to deal with documents containing complex components, for instance, tables and mathematical expressions. Tables can efficiently and concisely present information that have relations and many of today’s documents have different types of tables. As a consequence, recognising tables is an interesting topic for research. The application of this area can be realized in information retrieval and classification of documents, etc. Tables usually have no standard format. The aims of table recognition are to detect tables on pages and then try to extract their structural and content information.

2.2 Document structural analysis

The main function of document structural analysis is to detect the borders of different components and the spatial relationships between these components in documents. The demands in this area specifically encompass the following:

1. Segmentation of text word.
2. Segmentation of text line.
4. Table structure recognising.

Much work on these issues has been conducted. Several schemes and techniques are published in [Jain 98, Lee 00]. An example document analysis survey is reported
The majority of existing document structural analysis methods presume the existence of some knowledge about document properties. The information needed for these properties can be very precise and specific, for instance, space width between documents components or shallow like how many columns a document is divided into.

2.3 Table analysis

By observing the current research work, it is found that some particular problems regarding table recognition, have not been given the deserved attention and even the researchers who work in this area have mainly used only heuristic rules [Silv 06], which are usually inferred from local analysis, to recognise tables. To understand the existing limitations of current table recognition methods, firstly the stages of table analysis are discussed:

1. **Table detection:** the table is usually located using empirical rules extracted from local analysis. As a consequence, current table detection methods fail in many cases. To avoid premature commitment to errors, some research works just pass by this task and focus on recognising clipped tables [Zani 04].

2. **Table syntactic recognition:** here tables are segmented into cells or blocks. However, no detailed logical relationships between cells is given (i.e, remains unspecified).

3. **Table semantic recognition:** unlike the last task, the semantic relationships between table’s cells are formally determined.

The majority of current table recognition methods focus on the first two tasks (i.e., table detection and table syntactic recognition). These techniques often use, for recognising tables, heuristic rules and not statistic rules that would be, by experience [Wang 04], more reliable. The third task does not attract the researchers because, it is almost impossible
to have obvious, sufficient and automatic way of assigning precise semantic information
to the relationships between table cells. However, an example where attempting to do
this is in [Taus 07], though the approach was for on-line recognition of matrices not for
off-line recognition technique.

Tables have a graphical structure which can be expressed by visual languages. Due
to the two-dimensional structure of table, two dimensional graph grammars are a valid
method for representing table elements. If the content of cells are also targeted, OCR
technologies are often engaged in the recognition process.

2.4 Methods used in table syntactic recognition

Since the mechanism used in our vision is still not fully known, no one can claim that
his/her method has the ability to perfectly recognise any tables. Tables have many styles
and conventions, therefore, it is very difficult, if not impossible, to create a method that
can cope with any eventuality, without the aid of human intervention. Therefore, in try-
ing to solve this problem, several methods have used syntactic information to decompose
table.

Text line and cell segmentation are often considered as a prerequisite step for table
structure recognition either for printed character or handwriting recognition. Much work
has been conducted for text line and cell segmentation of printed documents (scientific
journals, books, reports). In section 2.5, a brief description of standard text line/cell
segmentation methods is presented which includes Projection Profile Cutting techniques,
Smearing techniques, Grouping techniques and techniques based on Hough transform is
given. In section 2.6, some information about segmenting mathematical expressions is
presented. Then, in section 2.8, several cell segmentation techniques which use some of
the standard line/cell segmentation approaches and other heuristic features are discussed.
2.5 Standard line/cell segmentation techniques

Extracting line/cell from different components (paragraphs, mathematical expressions and tables, ..., etc.) is one of the first steps towards understanding them. Much work has been done on this area in trying to handle this non-trivial task. Next, the standard approaches, that are widely used to segment line/cell, are described.

2.5.1 Smearing approaches

Run-Length Smoothing Algorithm is one example of these techniques which reasonably works on printed documents [Wong 82]. Consecutive black pixels in this technique that are vertically overlapped are smeared. As a consequence, the white space between these black pixels is blotted as well to form one block if this space’s width is less than a particular threshold. Eventually, text lines are extracted by clustering the connected components found in the smeared image.

In [Wong 82], documents are scanned and converted into black and white images. Then, run-length smearing algorithm (RLSA) is used. An example is given to demonstrate how this method works. Let us have the following:

1. Black and white pixels that are denoted by (1’s) and (0’s) respectively.
2. An input of 0001100001100100000.

Then, the output of running the (RLSA) over this input would be: 1111100001111100000.
The RLSA often starts by smearing lines and then columns. The results of these smears are then merged. RLSA process smears all the material that exists in the original image. As a result, an image contains blocks corresponding to components (such as, paragraph, figure, etc.) that compose the original image is created.

Tables have mostly the same layout structure as documents. Therefore, it is possible, by minor modifications to RLSA method, to achieve segmentation of tables. However, as mentioned before, tables have no standard convention of composing, therefore, it is almost impossible to develop a technique that has good performance over all cases. For example, tables which have spaces between their columns that as wide as the spaces between their words, would not be correctly segmented by these kind of methods.

Figure 2.1(a) illustrates the application of this technique which succeeds in separating the columns. In contrast, figure 2.1(b), due to the tiny space between the columns, the technique fails to work.

### 2.5.2 Projection Profile Cuts (PPC)

These techniques are often used to segment printed documents. This method could also be adjusted to work with handwritten documents that have slight overlap. The vertical projection-profile can be obtained by gathering pixel values across the x-axis for each y value. After calculating the vertical profile, the text lines in the vertical direction can be extracted.

There is more than one way for obtaining a profile trend which include observing the transitions of black/white pixels such as in Marti and Bunke [Mart 01] or by assembling a number of connected components according to the vertical overlapping between them. This profile trend is usually smoothed, for instance, by a Gaussian or median filter to get rid of elusive local maxima [Bous 10]. The profile trend is then analysed to retrieve its
peaks and valleys.

As it can be observed, printed documents primarily contain blocks. Therefore, a document can be cut recursively into blocks. These document’s blocks can be represented by several models (such as tree, graph...etc that will be discussed later). The cut application can be based on the observation of pixels or connected components density. A detector is used to point out local peaks in the horizontal and vertical profiles that represent row and column separators. The cuts process are placed on these local peaks.

Similar to the previous technique, PPC could be used to decompose tables and indeed it is a fast simple way to effectively perform more complex layout analysis of tables.
However, it has some significant drawbacks. For instance, tables are likely to have some spanning cells, which in turn, makes the PPC over-segment or under-segment the columns of the table. This can be illustrated by the next example.

Considering the table in figure 2.2, The first projection would lead to a vertical cut that separates the table into two columns. First column headed by \textit{Col. N.} and the second one headed by \textit{Characteristics} (see figure 2.3(a)). This corresponds to the first layer in the resulting parse tree (see figure 2.3(c)). Then horizontal cuts are followed to separate the first column to nine rows and the second to ten rows (see figure 2.3(b)). This indeed forms the outer structure of the table. Nevertheless, this is not the only acceptable scenario. Figure 2.4 illustrates the other possible vertical and horizontal cuts.

It can be inferred from the above example that there is some drawbacks of using PPC. Below is a list of these drawbacks:

1. PPC separates columns or rows (over-segment), which is generally undesirable, as they have to be reassembled.

2. PPC would fail if it is used for multi-line rows. Specially, if there is no a distinguishable spaces between them.

3. Degraded images can also impose considerable problems, skew can blur the rela-

\begin{center}
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
\textbf{Col. N.} & \textbf{A} & \textbf{B} & \textbf{C} & \textbf{D} & \textbf{E} & \textbf{F} & \textbf{G} & \textbf{H} & \textbf{I} & \textbf{J} \\
\hline
1 & .0005 & .03 & -.02 & -.04 & .02 & -.08 & .04 & -.01 & -.04
\hline
2 & .0003 & -.02 & -.07 & -.05 & -.05 & -.05 & -.06 & .04 & -.02 & .02
\hline
3 & -.0003 & -.10 & -.08 & -.11 & -.04 & -.07 & -.07 & .01 & -.03 & 0
\hline
4 & .0001 & 0 & -.02 & -.04 & -.01 & -.02 & -.03 & -.05 & -.05 & .02
\hline
5 & .0013 & -.18 & 0 & -.13 & -.01 & -.04 & -.04 & -.04 & -.03 & .02
\hline
6 & -.0005 & -.07 & -.01 & -.01 & 0 & .03 & -.02 & -.15 & 0 & .01
\hline
7 & -.0013 & -.07 & .03 & -.07 & .02 & .05 & .02 & .04 & .01 & 0
\hline
8 & .0015 & .07 & .03 & .09 & .02 & .01 & -.02 & .02 & .01 & .02
\hline
\end{tabular}
\end{center}
Figure 2.3: An example of PPC applied to the table in figure 2.2

tionships between what is horizontal and vertical, touching symbols might wrongly segment as atomic blocks that can cause under-segmentation of rows and columns.

2.5.3 Grouping approaches

These techniques at first construct alignments by clustering units using a bottom-up approach [Breu 02]. These units could be simple like pixels or complex such as connected components and blocks. The joining process depends on both local and global constraints. These constraints are used for testing consistency.
The document structure method of O’Gorman [OGor 93] is an example of the grouping technique using a bottom-up approach. It assembles consecutive connected components using heuristic rules which are built using the geometric relationship that exists between adjacent units. This technique works on printed documents [Bake 09] and also on hand-written documents with little overlapping [Bous 10].

### 2.5.4 Hough transform approaches

The method described in [Houg 62] and [Illi 88] is the typical approach for retrieving the line’s parameters in a binary image. It was introduced to detect parametrised regions in binary images and also to compare each image point with all points that exist in the parameter space that could likely produce the image point. Therefore, each image point gives a vote for the region parameters that could produce it. The most likely to have produced a specific shape in the true image is the point(s) in the parameter space gathering the greatest number of votes. The main disadvantage of implementing such techniques is its need of big storage and other computational requirements.
2.6 Mathematical expressions segmentation

There are several types of input that are used in mathematical recognition techniques which encompasses vector graphics (such as PDF) or a document images. Each of these input types imposes some demands on detecting expressions. As for document images, several techniques apply, for instance, OCR techniques, before recognising mathematical expressions in documents, to locate mathematical expressions. For PDF documents, a match technique (like the one in [Bake 10]) is used to overcome the inaccurate bounding boxes of symbols extracted from PDF files.

2.6.1 Mathematical expression segmentation in images

In this type of mathematical recognition method, the mathematical expressions are usually detected using attributes of connected components. One can categorise these expressions into two: 1) displayed expressions which are stand alone apart from text paragraphs 2) embedded expressions that occur within text lines. Figure 2.5 shows some examples of displayed and embedded mathematical expressions.

Detecting displayed expressions are often easier than the expressions embedded in text lines. This is because there are distinguishable differences between regular text lines and displayed expressions properties (this encompasses geometric features like width, space separator and symbol sizes [Gara 07, Kace 01]).

In [Gara 07] another technique for locating embedded expressions is developed. Firstly ordinary text lines are detected. Then, some statistical facts about pure ordinary text lines and texts that have embedded expressions are obtained by observing neighbouring symbols. An accuracy rate of 97% is stated.

An extension work of [Gara 07] is presented in [Gara 09]. This is accomplished by
Figure 2.5: Examples of displayed and embedded mathematical expressions which is taken from [Stro 11]

calculating the average of sophisticated parameters values for detecting the two expression types. Using recall measure for evaluation, an accuracy rate of 88.3% and 97.2% for embedded and displayed expressions respectively are obtained. These rates are considered to be high due to the comparison with the rates in [Chu 13, Lin 11, Chow 03]

2.6.2 Mathematical expression segmentation in PDF files

For PDF documents, the work has been so far focused on how to extract symbols and recognise manually clipped expressions. However, to the best of our knowledge, a dedicated work about automatic segmenting of mathematical expressions is not yet accomplished, since the information currently available in PDF files does not provide any advantages in detecting mathematical expression positions [Zani 12]. Therefore, conducting
research on finding possible ways to tackle this issue is important for several applications of retrieving mathematical information.

2.7 Unsolved issues concerning the detection of lines containing mathematical expressions

While in regular text documents generally lines can be clearly separated by simply detecting consecutive whitespace between lines and applying vertical cuts or by using one of the techniques mentioned earlier, for documents containing mathematical expressions the current techniques do not suffice due to the occurrence of particular artifacts of mathematical notations (e.g., mathematical accents, the limits of sum symbols). And while there exists a body of work on the segmentation of mathematical documents, this work is generally more concerned with the identification and separation of mathematical structures from surrounding text and their subsequent layout analysis [Zani 12].

2.8 Existing table segmentation methods

The main segmentation goal is to describe the physical structure of a table, i.e. detect its cells and their border information, and also its columns and rows. In this section, we sum up the segmentation methods proposed by different authors. These techniques are categorised based on the approaches used in detecting a table’s cells.

2.8.1 Projection-profile approaches

In Green and Krishnamoorthy [Gree 95], the user supplies a table prototype. This prototype includes the features of the separators that split up the table into different areas, encompassing its cells, rows and columns. Based on these features, the table is consecutively sub-split into the smaller levels of the prototype and each one is labelled by combining the name of a new subdivision with name of the previous “above level”divisions. The
output is represented in an XY tree. Experimental results on a small number of tables are reported.

In Wang et al. [Wang 01], a vertical projection is made on the word boundaries and columns are extracted based on the space corresponding to every peak between two valleys. The same authors in [Wang 02] imply that they have the intention to build a probabilistic algorithm for this task. For the evaluation of this technique, a dataset of 1125 document pages which contains 518 table entities and 10941 cell entities is utilised. The method uses segmented line and word as input. 90% accuracy rate for correctly detecting cells was obtained.

Kieninger [Kien 98] builds a network of block segmentation based on words that are horizontally overlapped in consecutive lines. It is found that, unlike other document components, blocks in a table are only horizontally overlapped to form columns. A claim from the author stated that this algorithm has a reasonable performance on ASCII files and can be adapted to work with scanned documents. The issues with this method are that it has too many heuristic rules and there is no guidance for how to use the parameters in the post processing steps.

2.8.2 Heuristic approaches

Mixed rule

Ferguson [Ferg 97], uses column headers usually appearing in the first row of the table to roughly detect column bounders, then, lines are extracted until the detection of the end of table bounders. Based on this information, the table is segmented such that lines contain number of cells, ranged from one to the maximum columns of the table.

In [Janu 97], the model used to manipulate tabular data utilizes font features, positional information and some heuristic rules extracted using OCR results to detect the cell
boundaries. In case information is lacking, vertical and horizontal white space are utilized.

**White spaces rule**

In Shamillian et al. [Sham 97], the user is provided with a model that is able to retrieve table row and column separators using the white space; those separators are then used to segment the table.

In Tupaj et al. [Tupa 96], vertical white space paths existing in the table are retrieved; the lines intersecting a great number of such paths are eliminated from the table; when there is a line with narrow paths, the centre of the path assigns to the column separators. Unfortunately, cells spread over columns are not correctly extracted; and as a consequences, ghost columns containing a few cells are wrongly detected.

### 2.8.3 Grouping approaches

Kornfeld and Wattecamps [Korn 98] split up the lines of the table into cells. Each line would have number of cells not less than one cell and no more than the maximum number of columns that possibly exist in the table. A method then clusters the cells that are consecutive in the horizontal direction where the space between them is too small for the horizontal gap usually available between the columns.

Chao [Chao 03] applies a hierarchical grouping algorithm to cluster the words inside the table (to form cells) based on how close their beginning and end positions are. Then the best clustering levels that might compose a column are chosen according several heuristics. The author assumes that the column in the leftmost position of the table is often a header. Then, a method is utilized to detect table rows. The lines that have no data are assembled with the above line. This basic heuristic leads the cells which represent titles in lower position to be gathered with lines that have no data. The method does not consider data cells which are spread over lines.
Mandal et al. [Mand 06] have proposed a simple method for table detection. It relies on observing spaces between table cells in an image (this gap is supposed to be larger than the gap between words in a regular text line). This technique claims an accuracy rate of 97.21%. One drawback of this approach as the author stated is that the algorithm does not work properly when a page contains tables only or those with narrow columns with single word/numbers.

2.8.4 Advanced approaches

In Ng et al. [Ng 99], they expected column delimiters to be whenever the change from an alphanumeric to a non-alphanumeric character takes place. Every column which has a width of one character is compared with the following and the previous and checked whether there are changes between different character groups. A neural networks algorithm (decision tree induction algorithm together with a backpropagation algorithm) was utilized to infer a classifier that judges whether a particular vertical line in the document is a column delimiter or not.

Handley [Hand 00], starts by observing if the table cells are completely extracted using graph line or not. If not, the position of line is combined against the relative location of the word boundaries and white spaces around them. An initial guess of lines is constructed using the histogram of the horizontal projection of the word boundaries in the table; then words that are vertically overlapped and are close to each other are clustered into cells. A histogram of the vertical projection on cells is constructed to extract columns. However, header cells of the table would give less weight, because the likelihood of such cells to be spanning is high. A more precise heuristic using the same histogram approach is used for detecting spanning rows and cells.

In Hurst [Hurs 01, Hurs 03], the proposed algorithm uses a language model in analysing
the document. The model encompasses statistics including how many two words occur together in the training set. The document is retrieved to collect the points needed for a decision maker to decide about what the next to text components in documents should be. The method splits, using the model of language, the candidate cells existing in a table. The output of this technique is segmentation of the table into cells [Hurs 01], spanning cells are also included. In [Hurs 03] Hurst develops an approach that clusters these cells into rows and columns. This is accomplished by concatenating together cells that horizontally or vertically overlap with each other. Several conditions must be satisfied when constructing these links. This step might correct the mistakes made in the segmentation of cells phase described in [Hurs 01].

2.9 Existing table segmentation techniques for PDF/Postscript

Yildiz et al. [Yild 05] proposed a scheme called PDF2table which is a syntactic approach that encompasses two techniques: table recognition, where components arranged in tabular form is segmented, and table re-composition in which extracted elements are passed to a table model. Experimental results of running the approach over a dataset of 150 documents are presented. However, comparing this method with other technique is impossible, due to the fact that the dataset is not available.

Hassan and Baumgartner [Hass 06] design a method that extracts components from PDF files (such as paragraphs, tables and graphics). These components are then tagged with a appropriate labels. This work is accomplished by observing the features of all text fragments structure and also by considering the white gap between different components. Authors mentioned that their work can not ensure a reasonable performance when it runs over PDF files that contain tables. The output can sometimes contain an object that represent a whole column, also sometimes cells of tables are interpreted as single objects.
Lawrence et al [Lawr 99] have designed methods that aim to improve the accessibility of scientific documents. This work is sponsored by Citeseer project. This research mainly focused on indexing all texts that are within scientific articles which have Postscript and PDF formats. However, the necessary attention for tables structure and their contents is not given in this research.

Anjewierden [Anje 01] works with information directly extracted from PDF files. He claims that this would allow us to have the same components as they appear in original documents, as well as extra fonts and style information. He states that his grammars effectively work with several type of documents. This system, according to the author, performed on three different document domains which are:

1. Military equipment.

2. Car repair.

3. Electronic control systems for traffic lights.

### 2.10 Unsolved issues concerning the segmentation of tables into cells

Silva et al. [Silv 06] state that most approaches to table recognition are still not general enough to derive physical models of complex tables with overlapping cells if no grid-lines are available. As an example, they present the table in Fig. 2.6 where the top left cell spans two columns, but can usually not be detected by table recognition approaches if the cell contains more than one distinguishable word (e.g., [Tupa 96, Kien 98]). Unfortunately, tabular layout of this nature is very common for mathematical formulas.
An exceptional approach that can deal with these types of cells is presented by Hurst [Hurs 01, Hurs 03]. Rather than using abstract words and their position to identify columns, Hurst uses a language model in order to group words into cells, before constructing rows and columns. While this makes the method more robust to the presence of multicolour or multirow cells as well as to some extend to cell overlap, one needs to have a sufficiently robust language model first, which to our knowledge does not exist for arbitrary mathematical expressions.

### 2.11 Table representation techniques

Tables are usually represented by a tree or graph at the logical level. The role of this table recognition stage is to bridge the tables at abstract and physical levels by defining the table cells and the proper way of how they can be recomposed to form a table. Examples are in [Hara 94, Nagy 00, Wang 96, Wang 93]. Before going further, a brief description of the table at abstract and physical levels is given.

The table encompasses and expresses the intention of the author. This could be statistical data of a particular type of job seekers, or a comparison of different data that the composer tries to convey in a concise manner. *Abstract tables* label can be given to tables in this level where a description of table structure is manually presented by the composer. Examples of such a level of tables are provided in [Wang 93].

Tables at the physical level might use pixels, glyphs and lines within documents to
represent them. Physical tables mean here the description of layout structures of the tables [Hara 94, Hurs 99, Nagy 00]. Physical tables are often constructed automatically; these kind of tables are usually composed by tools which are built on some high level description of the tables. For instance, tables in XML documents are constructed by browsers using mark-up tags. In the same manner, tables in PDF documents could be composed from strings of table description that obey the \LaTeX\ syntax rules.

2.12 Models to represent tables

As mentioned above, a logical tables role specifies how table cells should be re-arranged using the relationship between cells. Next, various models are illustrated which include Tree, Dual Hierarchy, CALS and Graph Models.

2.12.1 Tree models

Trees are a well-known model for representing the structures of table. Nagy and Seth [Nagy 84] were the first authors to use x-y trees for representing tables. Since then, such models have been widely utilized. Examples of applications encompassing the representation of the hierarchy of form documents, for instance, [Duyg 00, Duyg 98], table structures recognition, for instance, [Ha 95, Sylw 01, Zou 07] and extraction data from tables, for instance, [Kr 05, Tari 98].

To extract a table’s structure, the table is split into rectangular blocks using the x-y cut algorithm [Zani 00]. The algorithm recursively cuts along row and column delimiters until nothing remains to be cut. For example, a vertical cut can split the table in Figure 2.7 into two parts: the right part that represents cell A and the left part that represents cells B and C which could be segmented again by projecting a horizontal cut to the left part. The resulting cells are represented by an x-y tree. The whole table is represented by the
Figure 2.7: Table used for describing the work of the x-y cut algorithm

Figure 2.8: The table cells in Figure 2.7 represented by x-y tree

root of the tree where a leaf node denotes a single cell and a non-leaf node indicates a part that can be further segmented. Figure 2.8 illustrates the x-y tree for the table structure in Figure 2.7. The x-y tree structure matches with the table structure that it represents. The x-y tree hierarchy matches with the table block hierarchy and the re-composition of a table can be accomplished by traversing the tree.

2.12.2 Dual hierarchy models

In this type of representation, row and column hierarchies are simultaneously used to represent a table. Examples of this kind of representation are in [Bigg 84, Beac 85, Came 89]. A linked list of rows and a linked list of columns are utilized to represent a table. Each cell in the table corresponds to an element in both the row and column lists. If a cell was to be deleted, then the row and column lists must be updated.
The authors in [Bigg 84] introduced a system for recognizing tables called TABLE. However, prior to the system implementation, a comparison between the effectiveness of the tree and dual hierarchy representations was accomplished. The authors concluded that there are advantages and disadvantages for both representations, functions which perform well on one representation were usually shown to degrade in accuracy at the other one. For instance, functions which deal with cells spanning more than one row or column were much easier to be implemented using the tree representation. On the other hand, functions that are used to observe the pointer movements between cells of tables were much easier to perform using the dual hierarchy representation. The authors overall decided to use the dual hierarchy representation. Since there are similarities between this type of representation and graph representation which, as we will be discussed later, outweighs the advantages of using a tree representation, one can say that authors was right to work with this representation.

2.12.3 The CALS model

This model was designed by the Organization for the Advancement of Structured Information Standards (OASIS). Tables in SGML/XML use this model as a standard to represent them. It is designed to deal with a wide range of complex tables found in technical documents of military organizations. The CALS modal allows encoding the geometric features that are extracted from a table as well as several basic formatting styles, for instance, borders and cell alignment. XML strings are used to define tables in the CALS model. These strings must meet the CALS DTD, see [OASI 95, OASI 96, OASI 99]. The CALS DTD contains many detail but structurally, this table model establishes that any table must include a head title.

Among the characteristics of this model is the one that gives the ability to authors to assign name strings to columns. Rather than pointing to columns using their references, such as column no 1 and column no 2, columns could be retrieved by some strings, such
as “student names column” and “student ID column” in Figure 2.9. Despite the fact that a set of characters can be assigned to columns, this CALS model cannot semantically interpret these column names.

### 2.12.4 Graph model: Amano and Asada’s model

A table graph representation usually encodes the geometric relationships between table cells such that each cell is denoted by a node and that the relationships between these cells are denoted by edges.

In [Aman 02] a graph representation system which directly implements the geometric relationship between table cells is introduced. These cells defined by projecting on the x axis and y axis to extract vertical and horizontal lines positions that are then used in detecting the borders of table’s cells. Every cell in the table is denoted by a node. An edge in the graph is used to represent the relationship between pairs of nodes. Different edges are considered to represent a relationship between any two cells. Here is an illustration of some of these edges.

![Table](image)

Figure 2.9: Example of table used in illustrating CALS representation model
Figure 2.10: Cells that are totally and vertically overlapped

1. Two cells that are vertically overlapped where they have the same start and end y-axis border values. Figure 2.10 shows Cell A and Cell B and the unique geometric relationships between them (edge).

2. Two cells that are vertically overlapped where the start and end y-axis borders of one cell is within the start and end y-axis borders of other cell. Figure 2.11 shows Cell A and Cell B and the unique geometric relationships between them (edge).

3. Two cells that are horizontally overlapped where they have the same start and end x-axis border values. Figure 2.12 shows Cell A and Cell B and the unique geometric relationships between them (edge).

4. Two cells that are horizontally overlapped where the start and end x-axis borders of one cell is within the start and end x-axis borders of other cell. Figure 2.13 shows Cell A and Cell B and the unique geometric relationships between them (edge).
5. Two cells that are horizontally overlapped where the end x-axis border of one cell is greater than the start x-axis border and less than the end x-axis border of other cell. Figure 2.14 shows Cell A and Cell B and the unique geometric relationships between them (edge).

In this representation model, the geometric relations that occur between table cells are represented by the edge types in the graph. For example, figure 2.16 is the graph representation of figure 2.15 (Note: only a sub set of edges are shown in this example not all edges). This representation opens the gate for expressing table layouts using context-free grammars by detecting all layout relations any two cells can have.
2.13 Unsolved issues concerning the representation of table structure

The authors of [Rame 03] analyse the two most well-known table representation systems (which are introduced by the World Wide Web Consortium (W3C) and Advancement of Structured Information Standards (OASIS)) that are used to represent tables and find that they share the same deficiencies. First, the representation of irregular physical layouts are difficult. Poorly aligned borders of cells are not allowed and improvised solutions are provided for the spanning cells. Finally, limited means are supplied for the description of the logical structure of a table. Therefore, the first issue is to find a general model for representing the information of table structure. Table representation models which are domain-dependent do exist, however, usually the performance of these methods hugely decreases when one tries to generalise. Scalability, extensibility are among the preferable
properties of any general representation methods.

### 2.14 Summary

In this chapter, table recognition literature review, which encompasses a discussion of different techniques and approaches that are used to tackle several problems facing the recognition of table structure, is given. Firstly, current line/cell segmentation methods are reviewed and classified based on the standard approaches that are widely utilised in segmenting lines/cell. In addition, some issues regarding detecting mathematical expression is mentioned. Secondly, several existing cell segmentation techniques which utilise various features, that can be observed within tables, to detect table cells, as well as, unsolved problems concerning cell segmentation are discussed. Finally, different algorithms that are currently used to represent table structure and some limitations concern these representation algorithms are illustrated.
Part II

Preprocessing Steps Toward A Robust Mathematical Table Recognition
CHAPTER 3

A NOVEL HISTOGRAMMATIC APPROACH TO LINE SEGMENTATION

The basis of our table recognition system is a robust algorithm for the separation of mathematical lines. Line segmentation is a prerequisite step for structural analysis of both printed and handwritten documents. Much work has been done for text line segmentation of documents containing text only. The developed techniques such as projection profile cutting [Mart 01, Bous 10], smearing [Wong 82], grouping [OGor 93] or seam carving [Saab 11], rely to some extent on the fact that in regular text documents generally lines can be clearly separated by detecting consecutive whitespace between them.

For documents containing mathematical expressions, however, these techniques do not suffice due to the occurrence of particular artifacts of mathematical notations such as mathematical accents, the limits of sum symbols, etc. that, while actually constituting a single line, can appear spatially laid out over more than one separable line. And while there exists quite a body of work on the segmentation of mathematical documents, this work is generally more concerned with the identification and separation of mathematical structures from surrounding text and their subsequent layout analysis [Zani 12].

In this chapter we present a mathematical line recognition algorithm that is independent of knowledge on any peculiarities of mathematical expressions. It is based on spatial
considerations only, thus avoiding committing to premature errors that stem from considering actual content such as symbols or fonts. In particular, we use a histogram-based approach, considering horizontal spaces between glyphs in lines of a page, in order to classify lines into two types: principal and non-principal, where the former are lines in their own right, while the latter are only parts of mathematical expressions and should be merged with neighbouring lines. In addition to this technique we have developed a set of heuristics using simple yet effective measures for the correction of classification errors as well as to separate lines that share vertically overlapping characters but that belong to distinct mathematical expressions. We demonstrate the effectiveness of our approach by presenting experiments on two distinct data sets containing 200 and 1000 pages from mathematical documents, where we achieve an accuracy rate of 96.9% and 98.6%, respectively for line detection (Sec. 3.2).

The basic idea of our approach is to detect all possible individual lines first and then merge neighbouring lines into single lines likely to contain mathematical expressions. Thereby we rely neither on knowledge of the content of lines, font information nor vertical distance. Instead we use a histogrammatic measure on space within a single line. In a final step we then employ simple height considerations to detect lines that have not been merged or wrongly merged.

3.1 The approach at a glance

In summary our procedure consists of the following six steps:

1. Compute bounding boxes of all glyphs on a page.

2. Construct lines by separating sets of glyphs with respect to vertical whitespace.

   That is, we use vertical cuts similar to projection profile cutting [Zani 00]. These lines will get refined later.
3. *Detect and split lines* with vertically overlapping characters.

4. We classify lines as principal or non-principal — where the former constitute a proper line while the latter should be merged with a principal line above or below (see figure 3.8)— by comparing the horizontal distances of neighbouring glyphs. A line is classified as non-principal if no pairs of its glyphs have a separation distance within a certain threshold, where the threshold is computed from the histogram of all horizontal distances between neighbouring glyphs for the entire page.

5. The classification is then corrected by comparing non-principal to neighbouring principal lines with respect to their *maximum character height*. If a non-principal line is not within a ratio of the character height of its neighbouring principal lines it is promoted to principal line.

6. In a second corrective step we compare all lines identified as principal on a page with the maximum height of all existing non-principal lines on that page. If any principal line is less than or equal to that maximum, it is demoted to non-principal line.

7. Finally non-principal lines are *merged recursively* with their closest neighbouring principal lines.

More formally we define the following concepts for our algorithms. First we render our notion of bounding boxes (See figure 3.1) more precise:

**Definition 1** (Bounding Box). Let $g$ be a glyph, then the limits of its *bounding box* are defined by $l(g), r(g), t(g), b(g)$ representing left, right, top and bottom limit respectively. We also have $l < r$ and $t < b$.

**Definition 2** (Vertical and Horizontal Overlap). Let $g_1, g_2$ be two glyphs. We say $g_1$ overlaps vertically with $g_2$ if we have $[t(g_1), b(g_1)] \cap [t(g_2), b(g_2)] \neq \emptyset$, where $[t(g), b(g)]$ is the interval defined by the top and bottom limit of the glyph $g$. 

40
Similarly we define horizontal overlap of two glyphs $g_1, g_2$ by $[l(g_1), r(g_1)] \cap [l(g_2), r(g_2)] \neq \emptyset$.

**Algorithm 1:** Composing initial lines algorithm

```
input : Non-empty list of bounding boxes of page's glyphs that is in ascending order according to $t(g)$
output: Initial lines contain mathematical structure

1 begin
2 let list = list of glyphs $g$
3 let lines = emptylist
4 let listTemp = emptylist
5 let tallestGlyphs = b(head(list))
6 while list $\neq$ emptylist do
7     if $t(head(list)) < tallestGlyphs$ then
8         add head(list) to listTemp
9         tallestGlyphs = max$_{g \in listTemp} b(g)$
10        remove head(list) from list
11     else
12         add listTemp to lines
13         let tallestGlyphs = b(head(list))
14         let listTemp = emptylist
15      add listTemp to lines
16 return lines
```

We can now define a line using the vertical overlap on a set of glyphs. The goal of this step is to extract initial lines that exist in a page. See figure 3.2.

**Definition 3** (Line). Let $G = \{g_1, ..., g_n\}$ be the glyphs of a page ascendant ordered by $t(g)$. Then $L = \{g_1, ..., g_m\}$ is the first line of this page if it is the maximal prefix of $g_1, ..., g_n$ such that for all $j = 1, ..., m$, $b(g_j) \geq t(g_m)$. 

Figure 3.1: Examples (taken from [Grad 07]) of glyph’s bounding boxes and vertical and horizontal overlaps.
To extract the rest of lines existed in this page, repeat the following two steps while \( m + 1 \leq n \):

(i) Remove all \( g \in L \) from \( G \), i.e. \( G = G \setminus L \).

(ii) Identify the new line utilizing the same condition that was used to extract the first line.

Algorithm 1 presents a pseudocode which shows how to extract lines from a page’s glyphs.

The intuition of the next definition is to separate lines that share vertically overlapping characters as one line. For examples of different types of overlaps see Fig. 3.3. Therefore, we perform a post-processing step to detect and split those lines, which is formalised in the next three definitions:

**Definition 4 (Detect Overlapping Line).** Let \( L = \{g_1, \ldots, g_m\} \) be a line where the glyphs are sorted in ascending order according to \( l(g) \). We split \( L \) if ALL of the following conditions are satisfied:

(i) Neighbouring glyphs \( g_i, g_{i+1} \in L \) horizontally overlap such that \([l(g_i), r(g_i)] \cap [l(g_{i+1}), r(g_{i+1})] \neq \emptyset \) and \( b(g_i) < t(g_{i+1}) \)

This condition is insufficient by its own, it just gives a list of lines that contain horizontal overlap glyphs. This list usually include lines with horizontal overlap glyphs but which do not overlap with other lines. Therefore, the next three conditions
Figure 3.3: Examples of different types of lines overlapping (taken from [Grad 07])

(i) On a une décomposition $H_s(A+G,M) = \oplus_{i,j} H_s(A+G,M)_{[ij]}$ ainsi qu’une suite spectrale $E_{\nu}^{\tau,s} = H_s(\mathcal{Z}(g),H_s(A,M_g)) \Rightarrow HH_{r+s}(A+G,M)_{[ij]}$.

(a) Lines contain embedded math expressions that share vertically overlapping characters.

\[
\left\{ \begin{array}{l}
\frac{\partial}{\partial \xi} (x,s,\xi) \leq h_2(x) + b_3(x) |s|^{\frac{1}{2}} + |\xi|^2 \\
\frac{\partial}{\partial \xi} (x,s,\xi) \leq h_2(x) + b_3(x) |s|^{\frac{1}{2}} + |\xi|^2 
\end{array} \right.
\]

(b) Lines with displayed math expressions that share vertically overlapping characters.

(iii) $\psi'$ is non-increasing

\[
\sum_{j=1}^{\infty} \frac{\partial}{\partial s} a_j(x,s) \xi_i \xi_j \leq 2e^{-1K} \psi'(|s|) a_j(x,s) \xi_i \xi_j \text{ for all } s \in \mathbb{R}^m
\]

(c) Lines contain embedded math expressions that are overlapped because a part of these lines is misaligned

\[
\varphi(t+h) - \varphi(t) \leq \int_{-\infty}^{t} (S(h) - t) A^\alpha S(t - \sigma) f(\sigma, A^{-\alpha} \varphi(\sigma)) d\sigma \\
+ \int_{t+h}^{\infty} A^\alpha S(t + h - \sigma) f(\sigma, A^{-\alpha} \varphi(\sigma)) d\sigma
\]

(d) Lines with displayed math expressions that are overlapped because a part of these lines is misaligned

Figure 3.3: Examples of different types of lines overlapping (taken from [Grad 07])

attempt to eliminate these kind of lines to end up with the correct overlapping lines list.

(ii) The same two neighbouring glyphs $g_i, g_{i+1} \in L$ do not both vertically overlap with any $g \in L$ where $g \neq g_i$ and $g \neq g_{i+1}$ such that $t(g) > b(g_i)$ or $b(g) < t(g_{i+1})$.

This condition remove lines from the list that contain horizontal overlap glyphs. However, they both vertically overlap with one glyph. For instance, the limits associated with integral symbol in figure 3.4

(iii) $h(g_i) < (t(g_{i+1}) - b(g_i))$ and $h(g_{i+1}) < (t(g_{i+1}) - b(g_i))$ where $h$ is the height of glyphs such that $h = b(g) - t(g)$.

In this stage, the gap between $b(g_i)$ and $t(g_{i+1})$ must be greater than the height of each glyphs. This would eliminate lines with horizontal overlapping glyphs, that are conveyed one symbol, that usually are wrongly classified as overlapping lines.

Figure 3.5 shows a case of these lines.

(iv) $h(g_i) > (w(g_i)/2)$ and $h(g_{i+1}) > (w(g_{i+1})/2)$ where $w$ is the width of glyphs such
Figure 3.4: Example of horizontal overlap glyphs that both vertically overlap with one glyph

\[
\int_0^1 r \, P_n (1 - 2x^2) \, K_0 (xy) \, dx = y^{-1} \left[ (-1)^{n+1} \, K_{2n+1}(y) + \frac{i}{2} \, S_{2n+1}(iy) \right]
\]

Figure 3.5: Example of horizontal overlap glyphs that are conveyed one symbol that \( w = r(g) - l(g) \).

This final condition is mainly for equal symbol where it composes of two horizontal overlap glyphs. Lines that are wrongly classified as overlapping lines based on this symbol are removed.

The intention of these cases is to find lines, that are each wrongly encompasses of more than one line. Then, split them into lines that correspond to the correct lines. Each of these conditions eliminate, from the candidate lines, some type of the elusive lines that might have horizontal overlapping glyphs but there are still only a single line.

We then split the line into two lines by using a threshold that is determined by projecting onto the vertical axis across the whole vertical distance between the two overlapping glyphs, using the \( y \)-coordinate value of the line that crosses the least number of glyphs (See figure 3.6(a)).

**Definition 5** (Separator Value). Let \( L \) be a line that can be split according to definition 4. We apply a vertical projection profile on \( L \) such that separator value \( S \) is defined as the minimum value in the valley curve.

We then cluster glyphs into two lines using the separator value \( S \) as a threshold (See figure 3.6(b)). The intuition of this step is to extract the correct lines from the lines that are actually composed by overlapping lines. This step is repeatedly performed until no overlapping lines in the page. The algorithm 2 describes how to detect and split overlap-
Algorithm 2: Detecting and splitting overlapping lines algorithm

**input**: lines a list of the initial lines

**output**: Correct lines contain mathematical structure

1 begin
2     let isOverlap=true
3     let correctLines = emptylist
4     while lines ≠ emptylist do
5         let lineTemp, line = head(lines)
6         sort lineTemp in an ascending order according to \( l(g) \)
7         let firstItem = head(lineTemp)
8         remove head(lineTemp) from lineTemp
9         while lineTemp ≠ emptylist do
10            if \( [l(firstItem), r(firstItem)] \cap [l(head(lineTemp)), r(head(lineTemp))] \neq emptylist \)
11               and \( t(g \in line \setminus \{firstItem, (head(lineTemp))\}) > b(firstItem) \) or
12               \( b(g \in line \setminus \{firstItem, (head(lineTemp))\}) < t(head(lineTemp)) \)
13               and \( h(firstItem) < (t(head(lineTemp)) - b(firstItem)) \) and
14               \( h(head(lineTemp)) < (t(head(lineTemp)) - b(firstItem)) \)
15               and \( h(firstItem) > (w(firstItem)/2) \) and
16               \( h(head(lineTemp)) > (w(head(lineTemp))/2) \) then
17                 calculate the separator \( S \) using vertical projection profile
18                 split line using \( S \) such that
19                 \( L_{above} = \{g \in line | t(g) + (h(g)/2) < S \} \) and \( L_{below} = L \setminus L_{above} \).
20                 add the two split lines \( L_{above} \) and \( L_{below} \) to correctLines
21                 assign false to isOverlap
22             if isOverlap then
23               firstItem = head(lineTemp)
24             remove head(lineTemp) from lineTemp
25         else
26             assign emptylist to lineTemp
27         if isOverlap then
28             add line to correctLines
29             isOverlap=true
30     remove head(lines) from lines
31 return correctLines
Figure 3.6: (a) illustrates an example (taken from [Grad 07]) of compound line contains two horizontal overlap lines and a separator. (b) shows the resulting two lines ping lines more formally.

We can now define the distance measure with respect to which we will consider histograms. The x-axis in this histogram presents horizontal distance values between split-pair glyphs in a page and y-axis presents how many time each of these values occur.

**Definition 6** (Split-pair, Horizontal Distance). Let \( L = \{g_1, ..., g_n\} \) be a line where the glyphs are sorted in ascending order according to \( l(g) \). We call two glyphs \( g, g' \in L \) split-pair if \( r(g) < l(g') \). We then define the horizontal distance \( d \) between two split-pair glyphs \( g, g' \) as \( d(g, g') = l(g') - r(g) \).

The intuition here is to define that any two glyphs can only be split-pair if there are not horizontally overlapped. For instance, in \( \sum_{i \in \mathbb{Z}} \frac{q_i}{2} \), the split-pairs are \((\sum, i), (i, \_\_\_), (b, a), (i, c)\). However, “b” and “\_\_\_” are not split-pair.

Algorithm 3 presents a pseudocode that shows how to calculate horizontal distances between a page’s glyphs.
Algorithm 3: Calculating horizontal distance algorithm

\[\text{input} : \text{correctLines} \text{ a list of the correct lines} \]
\[\text{output}: \text{A list of horizontal distances between page's glyphs} \]

begin
  let distances = emptylist
  while correctLines ≠ emptylist do
    let line = head(correctLines)
    sort line in ascending order according to \(l(g)\)
    let firstItem = head(line)
    remove head(line) from line
    while line ≠ emptylist do
      if \(r(firstItem) < l(head(line))\) then
        add \(l(head(line)) - r(firstItem)\) to distances
        firstItem = head(line)
      remove head(line) from line
    remove head(correctLines) from correctLines
  return distances

The distance measure from the previous definition allows us now to compute a histogram that captures the horizontal distances between glyphs in lines for the entire page. Figure 3.7 shows an example for the histograms, where the x-axis denotes the values for the distance measure \(d\) in pixels and the y-axis the number of occurrences of a particular distance. Note, that when building the histogram, we deliberately omit all the values where no distance occurs or in other words, where the y value is equal to 0.

We can observe a general pattern in these histograms: They can be split into two parts by a global minimum that is roughly in the middle of the x-axis. This leaves two parts, each with a global maximum. Furthermore in the right part one can identify a further global minimum. While this can be at the very end of the x-axis it usually is not. We call these two minimal points \(v_1\) and \(v_2\), respectively. Algorithm 4 presents a pseudocode that shows how to calculate these values. We use them to define classification of lines as follows:

Definition 7 (Principal Lines). Let \(L\) be a line which its glyphs ascendantly ordered
Algorithm 4: Calculating \((v_1, v_2)\) algorithm

input: \(distances\) a list of the horizontal distances between a page’s glyphs
output: two minimal points \(v_1\) and \(v_2\)

begin
\[\text{let times}=0\]
\[\text{let } first_{max}=0\]
\[\text{let } second_{max}=0\]
\[\text{let distancesPair = emptylist}\]
\[\text{sort } distances\text{ in ascending order according to the horizontal distance values}\]

while \(distances \neq \text{emptylist}\) do
\[\text{let firstDist} = \text{head}(distances)\]
while \(firstDist = \text{head}(distances)\) do
\[\text{times}=\text{times}+1\]
remove \text{head}(distances) from \(distances\)
\[\text{add } (firstDist, \text{times}) \text{ to } \text{distancesPair}\]
\[\text{times}=0\]
split \(distancesPair\) into two halves such that
\[\text{let } \text{Temp}_{first-half}, \text{first-half} = \{(d, \text{times}) \in \text{distancesPair}|(\text{The position of (d, times)} < = (\text{length}(\text{distancesPair})/2))\}\]
and
\[\text{let } \text{Temp}_{second-half}, \text{second-half} = \text{distancesPair} \setminus \text{first-half}\]

while \(\text{Temp}_{first-half} \neq \text{emptylist}\) do
\[\text{if } \text{head}(\text{Temp}_{first-half}).\text{times} > first_{max} \text{ then}\]
\[\text{first}_{max} = \text{head}(\text{Temp}_{first-half}).\text{times}\]
remove \text{head}(\text{Temp}_{first-half}) from \(\text{Temp}_{first-half}\)

while \(\text{Temp}_{second-half} \neq \text{emptylist}\) do
\[\text{if } \text{head}(\text{Temp}_{second-half}).\text{times} > second_{max} \text{ then}\]
\[\text{second}_{max} = \text{head}(\text{Temp}_{second-half}).\text{times}\]
remove \text{head}(\text{Temp}_{second-half}) from \(\text{Temp}_{second-half}\)

while \(\text{first-half} \neq \text{emptylist}\) do
\[\text{if } \text{head}(\text{first-half}).\text{times} = first_{max} \text{ then}\]
\[\text{Let } V1_{list} = \text{first-half}\]
\[\text{Let } \text{first-half} = \text{emptylist}\]
remove \text{head}(\text{first-half}) from \(\text{first-half}\)

while \(\text{second-half} \neq \text{emptylist}\) do
\[\text{if } \text{head}(\text{second-half}).\text{times} = second_{max} \text{ then}\]
\[\text{Let } V2_{list} = \text{second-half}\]
\[\text{Let } \text{second-half} = \text{emptylist}\]
remove \text{head}(\text{second-half}) from \(\text{second-half}\)

assign \(d\) to \(v_1 \in V1_{list}\) and its pair \(\text{times} = \min_{\text{times} \in V1_{list}}(\text{times values})\)
assign \(d\) to \(v_2 \in V2_{list}\) and its pair \(\text{times} = \min_{\text{times} \in V2_{list}}(\text{times values})\)
return \((v_1, v_2)\)
by \( l(g) \). We call \( L \) a principal line if there exists two split-pair glyphs \( g, h \in L \) with \( v_1 \leq d(g, h) \leq v_2 \). Otherwise \( L \) is a non-principal line.

Algorithm 5 presents a pseudocode that shows how to classify lines into principal and non-principal lines.

The intuition behind this definition is that the values in the histogram less than \( v_1 \) represent distances between single characters in a word or a mathematical expression, whereas the area between \( v_1 \) and \( v_2 \) represent the distance between single words, which generally do not occur in lines that only constitute part of a mathematical formula, for
Algorithm 5: Classifying lines into principal and non-principal algorithm

input: correctLines a list of the correct lines
input: two minimal points \((v_1, v_2)\)
output: A list of lines marked with principal and non-principal

begin
let isPrincipal = false
let allLines = emptylist
while correctLines ≠ emptylist do
let line, lineTemp = head(correctLines)
sort \(g \in lineTemp\) in ascending order according to \(l(g)\)
let firstItem = head(lineTemp)
while lineTemp ≠ emptylist do
if \(r(firstItem) < l(head(lineTemp))\) and \(l(head(lineTemp)) - r(firstItem) > v_1\) and \(l(head(lineTemp)) - r(firstItem) < v_2\) then
mark line with principal
add line to allLines
isPrincipal = true
lineTemp = emptylist
else
firstItem = head(lineTemp)
remove head(lineTemp) from lineTemp
if Not isPrincipal then
mark line with non-principal
add line to allLines
isPrincipal = false
remove head(correctLines) from correctLines
return allLines
Figure 3.9: An example of lines (taken from [Deni 11]) are wrongly classified as non-principal

\[ E \left[ u(x + S(\nu_n)) ; \tau_x > \nu_n, \nu_n \leq n^{1-\varepsilon} \right] = E \left[ Y_{\nu_n} ; \tau_x > \nu_n, \nu_n \leq n^{1-\varepsilon} \right] \]

Consequently

\[ E \left[ u(x + S(\nu_n)) ; \tau_x > \nu_n, \nu_n \leq n^{1-\varepsilon} \right] = E \left[ Y_{\nu_n} ; \tau_x > \nu_n, \nu_n \leq n^{1-\varepsilon} \right] \]

Figure 3.10: An example of lines (taken from [Deni 11]) are wrongly classified as principal

While this measure alone already yields good results, it can be improved upon by considering a simple ratio between glyph heights of principal and non-principal lines.

**Definition 8** (line’s borders). Let \( \ln \) be a line, then the limits of its bounding box are defined by \( l'(\ln), r'(\ln), t'(\ln), b'(\ln) \) representing left, right, top and bottom limit respectively. We also have \( l' < r' \) and \( t' < b' \), such that \( l'(\ln) = \min_{g \in \ln} [l(g)], r'(\ln) = \max_{g \in \ln} [r(g)], t'(\ln) = \min_{g \in \ln} [t(g)] \) and \( b'(\ln) = \max_{g \in \ln} [b(g)] \)

**Definition 9** (Height Ratio). Let \( L_1 \) and \( L_2 \) be two consecutive lines, such that \( L_1 \) is a non-principal line and \( L_2 \) is the nearest principal line to \( L_1 \). If \( b'(L_1) - t'(L_1) > \frac{1}{T}(b'(L_2) - t'(L_2)) \), where \( 1 \leq T \leq 2 \) then \( L_1 \) is converted into a principal line.

Observe that the value for the parameter \( T \) is fixed and determined empirically by experiments in on a small sample set.

Algorithm 8 presents a pseudocode that shows how to detect wrongly classified as non-principal lines and convert them into principal lines.
Algorithm 6: Checking spaces between lines algorithm

1: function IsLess(currentLine, previousLine, nextLine)
2:   return \((t'(currentLine) - b'(previousLine)) < (t'(nextLine) - b'(CurrentLine))\)
3: end function

Algorithm 7: Checking the correctness of the line classification algorithm

1: function IsTruePrincipal(line1, line2)
2:   return \((b'(line1) - t'(line1)) > \frac{1}{T}(b'(line2) - t'(line2)))\)
3: end function

Since the previous step tackles only the problem of wrongly classified principal lines (See figure 3.9), we also need to define a corrective instrument to detect non-principal lines that have wrongly been classified as principal lines (see figure 3.10). This is achieved as follows:

**Definition 10** (Non-principal Height Bound). Let \(L_n = \{n_1, n_2, \ldots, n_l\}\) be the set of non-principal lines of a page and \(L_p = \{p_1, p_2, \ldots, p_k\}\) be the principal lines of the same page.

Then we define the non-principal height bound as the maximum height of all non-principal lines \(M\) as

\[ M = \max_{n \in L_n} |b'(n) - t'(n)|, \]

where \(t'\) and \(b'\) are the top and bottom limits of \(L\) respectively, such that \(t'(n) = \min_{g \in n} t(g)\) and \(b'(n) = \max_{g \in n} b(g)\).

Any \(p \in L_p\) is converted to a non-principal line, if and only if, \(\max[b'(p) - t'(p)] \leq M\).

Algorithm 9 presents pseudocode that describes how to re-classify principal lines to non-principal lines based on non-principal height bound.

Once the classification of lines is finished, in the final step we merge non-principal lines with their horizontally closest neighbouring principal line, but only if there exists horizontal overlapping between them (See figure 3.11). If not, the non – principal line is converted to principal line. This can be formally defined as follows:
**Algorithm 8:** Correcting wrongly classified as non-principal lines algorithm

**input:** allLines a list of lines marked with principal and non-principal

**output:** A list of lines after detecting some wrongly classified as non-principal lines and remake them as principal lines based on their relative height ratios

```
begin
let T between 1 and 2
let hrLines = emptylist
let previousLine = head(allLines)
remove head(allLines) from allLines
if is previousLine marked with non-principal
and IsTruePrincipal(previousLine, head(allLines))
remark previousLine with principal
add previousLine to hrLines
while length(allLines) > 1 do
if is head(allLines) marked with non-principal
and IsLess(head(allLines), previousLine, head(allLines \ head(allLines)))
then
if IsTruePrincipal(head(allLines), previousLine) then
remark head(allLines) with principal
else
if is head(allLines) marked with non-principal
and IsTruePrincipal(head(allLines), head(allLines \ head(allLines)))
then
remark head(allLines) with principal
previousLine = head(allLines)
add head(allLines) to hrLines
remove head(allLines) from allLines
if is head(allLines) marked with non-principal
and IsTruePrincipal(head(allLines), previousLine) then
remark head(allLines) with principal
add head(allLines) to hrLines
return hrLines
```
Algorithm 9: Correcting wrongly classified as principal lines algorithm

**input**: \( hrLines \) a list of lines marked with principal and non-principal

**output**: A list of lines after detecting some wrongly classified as principal lines and remake them with non-principal base on non-principal height bound

1 begin
2   let maximum = 0
3   let finalLines = emptylist
4   let hrLinesTemp = hrLines
5   while hrLinesTemp ≠ emptylist do
6       if is head(hrLinesTemp) marked with non-principal
7           and
8               maximum < \( b'(head(hrLinesTemp)) - t'(head(hrLinesTemp)) \) then
9               maximum = \( b'(head(hrLinesTemp)) - t'(head(hrLinesTemp)) \)
10              remove head(hrLinesTemp) from hrLinesTemp
11   while hrLines ≠ emptylist do
12       if is head(hrLines) marked with principal
13           and
14               \( b'(head(hrLines)) - t'(head(hrLines)) \) ≤ maximum then
15               remark head(hrLines) with non-principal
16               add head(hrLines) to finalLines
17               remove head(hrLines) from hrLines
18   return finalLines
Figure 3.11: An example of merging lines process (taken from [Deni 11])

Definition 11 (Merging Lines). Let $N$ and $P$ be non-principal and principal lines respectively, such that $P$ is the nearest neighbour of $N$. For any line $P$ or $N$, let $l'$ and $r'$ be the left and right limits of $L$ respectively, such that, $l' = \min_{g \in L} l(g)$ and $r' = \max_{g \in L} r(g)$. If $[l'(P), r'(P)] \cap [l'(N), r'(N)] \neq \emptyset$ then $N$ and $P$ are merged. Otherwise, $N$ is converted to $P$.

Algorithm 11 presents a pseudocode that shows how to merge each non-principal line with its closest principal line.

\begin{algorithm}
\begin{algorithmic}
    \STATE \textbf{function} Is-HO(line1, line2)
    \STATE \hspace{1em} \textbf{return} $([l'(line1), r'(line2)] \cap [l'(line2), r'(line2)] \neq \emptyset)$
    \STATE \textbf{end function}
\end{algorithmic}
\end{algorithm}

3.2 Experimental Results and Discussion

We have run experiments on two datasets of 200 and 1000 pages, respectively, taken from a wide variety of mathematical documents. Before discussing the results of our overall
Algorithm 11: Merging non-principal lines with closest principal lines algorithm

input : finalLines a list of lines marked with principal and non-principal
output: A lines list after merging non-principal with its closest principal lines

begin
  if Length(finalLines) > 1 then
    let mergeLines = emptylist
  if Is head(finalLines) a non-principal line and
    Is-HO((head(finalLines \ head(finalLines)), (head(finalLines)))) then
    merge head(finalLines) to head(finalLines\ head(finalLines))
    put the result in mergeLines then remove head(finalLines) and
    head(finalLines\ head(finalLines)) from finalLines
  else
    add head(finalLines) to mergeLines
    remove head(finalLines) from finalLines
  while Length(finalLines) > 1 do
    if Is head(finalLines) a non-principal line and IsLess(head(finalLines), lastline(mergeLines), head(finalLines\ head(finalLines))) and Is-HO(lastline(mergeLines), head(finalLines)) then
      merge head(finalLines) to last line(mergeLines)
      remove last line(mergeLines) from mergeLines
      put the result of 12 in mergeLines
      remove head(finalLines) from finalLines
    else
      if Is head(finalLines) a non-principal line and Is-HO((head(finalLines \ head(finalLines)), (head(finalLines)))) then
        merge head(finalLines) to head(finalLines\ head(finalLines))
        put the result in mergeLines then remove head(finalLines) and
        head(finalLines\ head(finalLines)) from finalLines
      else
        add head(finalLines) to mergeLines
        remove head(finalLines) from finalLines
      if Length(finalLines) = 1 and Is head(finalLines) a non-principal line and Is-HO((lastline(mergeLines)), (head(finalLines))) then
        merge head(finalLines) to last line(mergeLines)
        remove last line(mergeLines) from mergeLines
        put the result of 24 in mergeLines
      else
        if Length(finalLines) = 1 then
          add head(finalLines) to mergeLines
          return mergeLines
        else
          return finalLines
end
In terms of experiments of the overall procedure we have carried out initial experiments on 200 pages. These pages are taken from 13 documents comprising a mixture of books [Juds 09, Melj 03b, Wilk 98, Deni 11, Ster 05, Stro 11] and journal articles [Melj 03a, Mate 06, Zhan 07, Take 96, Guid 98, Jinz 99, Chui 01]. Table 3.2 presents the experimental results for this dataset. We have compared using simple vertical cuts, with our techniques of using the horizontal distance measure introduced in Def. 6 as well as using additionally the height ratio defined in Def. 9 and the non-principal height bound defined in Def. 10. As height ratio parameter we have set $T = 1.7$, a value that was experimentally determined on a 200 pages dataset.
Figure 3.12 shows a histogram that presents in its x-axis different values of $T$ (1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2.0) and in its y-axis the sensitivity and specificity rates that are obtained by running our technique over 200 pages (the training dataset) each time with different $T$. These sensitivity and specificity rates are extracted automatically.

As it can be seen, there is no big differences between rates using the set of $T$ mentioned above. However, the corresponding sensitivity and specificity rates of $t=1.7$ is relatively in their highest values. Thus, we use $t=1.7$ on our experiments of 1000 pages (the testing dataset).

![Figure 3.12: The results of running our technique over 200 pages with different ($T$)](image)

Altogether we manually identified 5801 lines in the 200 pages of the dataset. We compare this number with the number of lines found altogether and the number of lines identified correctly, that is, those lines corresponding to the actual line as manually identified in the dataset.

Not surprisingly simple vertical cuts results in a larger number of lines and, as there
Table 3.2: Experimental results for line recognition.

is no subsequent merging of lines, in a relatively low accuracy of 86.4%. Using the horizontal distance measure improves this accuracy, however, in general merging too many lines. This is corrected by the addition of the height ratio that re-classifies some of the lines incorrectly assumed to be non-principal as principal lines. As a consequence we get a slightly higher number of lines but also a higher accuracy of 96.3%. A further slight improvement in this accuracy to 96.9% is obtained using the height bound.

To further examine the robustness of our technique and in particular to rule out that it was overfitted to our original data set we have experimented with a second independent and larger data set. The data set contains 1000 pages composed from more than 60 mathematical papers different from our original set.(Some of the papers included are [Aass 02, Ahma 02, Aiss 02, Dads 02, Akdi 01, Akhm 02, Ghou 04, Alka 01, Alla 01].)

We ran our technique on this second larger data set and then manually checked the results by painstakingly going through every page line by line. Consequently we have done this comparison only for the full classification including both height ratio and height bound correction. And while we can not rule out some classification mistakes due to human error we are very confident that the experimental results given in Table 3.3 are accurate.

Table 3.3 demonstrates that although, the data set is five times the size of the previous one our classification results remain stable. In fact, one can see that in comparison with Table 3.2 we have even a increase of recognition rate by approximately 2%. This result
Table 3.3: Experimental results of 1000 pages.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pages</th>
<th>Total lines</th>
<th>Lines found</th>
<th>Correct lines</th>
<th>Precision</th>
<th>Recall</th>
<th>Balanced F</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height Bound</td>
<td>1000</td>
<td>32985</td>
<td>33374</td>
<td>32504</td>
<td>98.5%</td>
<td>97.3%</td>
<td>97.9%</td>
</tr>
</tbody>
</table>

Table 3.4: Evaluation results of 1000 pages.

<table>
<thead>
<tr>
<th>Line Type</th>
<th>correctly classified</th>
<th>incorrectly classified</th>
<th>Sensitivity</th>
<th>Specificity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principal Line</td>
<td>32904</td>
<td>201</td>
<td>99.39%</td>
<td></td>
</tr>
<tr>
<td>Non-principal Line</td>
<td>1242</td>
<td>81</td>
<td>93.87%</td>
<td></td>
</tr>
</tbody>
</table>

gives us confidence about the effectiveness of our technique even on large datasets and documents.

Further evaluation is shown in Table 3.4. Sensitivity (Sens) and Specificity (Spec) measurements are used. The front one concerns with the technique’s ability to detect a target object correctly. In our case, Sensitivity of our technique is the proportion of lines that are correctly detected as principal. The latter one concerns with the technique’s ability to exclude a target object correctly. In our case, Specificity of our technique is the proportion of lines that are correctly extracted as non-principal. Mathematically, these measurements can be written as:

Sensitivity (Sens) For P lines = \[ \frac{\text{correct P lines}}{\text{correct P lines} + \text{incorrect N lines}} \]

Specificity (Spec) For N lines = \[ \frac{\text{correct N lines}}{\text{correct N lines} + \text{incorrect P lines}} \]

As can be seen, the (Sens) percentage for principal line is high since there are a great deal of correct principal lines and a very small number of incorrect ones. For non-principal lines, the (Spec) percentage is not as high as the former lines. However, one can still claim that these results are very promising in comparison with the Vertical Cuts results where all non-principal lines are not recognized.

For the lines that were not identified, it is possible to categorise the recognition er-
3.2.1 Incorrect Non-principal Lines:

The most common error stems from classifying a line with respect to the horizontal distance measure as a principal line that should actually be non-principal. This is the case when there is a gap between two neighbouring glyphs that satisfies the horizontal distance condition. Below are some examples show several cases of errors taken directly from our dataset.

\[ \leq C \sum_{i=1}^{\infty} \sum_{j=1}^{\sqrt{i}} E[x + S(l)] \ldots \ e_{i} \bar{B}_{i+1} \bar{B}_{i} = 2\bar{M}_{i} + \ldots \]

Although, the first expression should be detected as a single line, the limits under the two summation symbols are at a distance that coincides with the distance identified by the histogram for the entire page. Likewise, in the second expression, also taken from our dataset, the tilde accents have a similar distance.

3.2.2 Incorrect Principal Lines:

This error occurs when a line is initially classified as non-principal line as it does not contain any glyph gaps that coincide with the distance measure derived from the histogram. Examples of these lines are those with single words, page numbers, single expressions etc. While these can be corrected by the height ratio, sometimes they are not as they do not satisfy the ratio condition. Below is an example taken from our dataset.

\[ + \frac{3}{5} \left( V^{k-1,k,2}_{1}(n; (1)) \cdot \left( I_{n-3}^{k-1,k,2} - I_{3}^{k-1,k,2} \right) \right) \]
Here the page number 12 is merged as a non-principal line to the expression above, as firstly it does not exhibit a glyph gap satisfying the distance measure and secondly its height is significantly smaller as the height of the open parenthesis in the mathematical expression.

### 3.3 Evaluation of integrating my line segmentation technique with MaxTract

My technique for mathematical line identification along with formula identification method that was developed by our group student visitor Xiaoyan [Lin 11] were integrated with MaxTract tool [Bake 12] to improve the output of its structural analysis of mathematical formula phase.

Table 3.5 shows two examples of how the output of Maxtract has improved after implementing the new techniques. The first column is a screenshot of the original formula, the second column shows the \LaTeX{} and its rendering generated, by the original Maxtract, and the third column shows the \LaTeX{} and its rendering generated after the modifications.

The first example shows how originally, a text line with an embedded formula was incorrectly recognised entirely as mathematics, and wrapped in a mathematical `align` environment. Whilst this only made a small difference to the rendering, with a different font and lack of space between $F$ and `and`, it showed that the semantics of this fragment had been incorrectly analysed by Maxtract. After the embedded formula had been correctly identified using the machine learning method in [Lin 13] (that is based on three modules which are classification, feature extraction and text line merging), the fragment was correctly recognised as a text line containing inline mathematical.

The second example shows where a formula with a lower limit was originally incorrectly
\begin{align*}
\text{and } F(-\infty) < \infty.
\end{align*}

$\text{and } F(-\infty) < \infty.$

\[N_i = \sum_{j} W_{ji} X_j.\]

$N_i = \sum_{j} W_{ji} X_j.$

<table>
<thead>
<tr>
<th>Original Image</th>
<th>Previous Output</th>
<th>New Output</th>
</tr>
</thead>
</table>
| and $F(-\infty) < \infty.$ | \begin{align*}
\text{and } F(-\text{infty}) < \text{infty}.
\end{align*} | and $F(-\infty) < \infty.$ |
| $N_i = \sum_{j} W_{ji} X_j.$ | \begin{align*}
N_{i} &= \sum_{j} W_{ji} X_{j}.
\end{align*} & \begin{align*}
N_{i} &= \sum_{j} W_{ji} X_{j}.
\end{align*} |

Table 3.5: Comparison of previously generated and new \LaTeX

63
split into two separate lines via Project Profile Cutting. This error was then propagated through to the formula recognition stage where it was erroneously treated as a multiline formula. When converted into \texttt{\LaTeX} the component was wrapped in an \texttt{aligned} environment, resulting not only in an incorrect structural analysis, but also when rendered, a formula that is no longer understandable. The histogramatic approach correctly identified that the limit was in fact a non-principal line and merged it with the previous principal line. This allowed the formula analysis to correctly identify the structure of an equation with a limit.

### 3.4 Summary

A description of a line detection method, that tackles the issues concern segmenting lines contain mathematical structures, is given. Lines are classified, using histogrammatic information, into two types (\textit{Principal} and \textit{Non–principal} lines). Then, post-processing steps are taken to ensure correct classification. Finally, each \textit{Non–principal} line is merged to its closest \textit{Principle} line. Experiments on 200 and 1000 pages from two datasets show an accuracy rate of 96.9% and 98.6% respectively. A brief discussion of the results that are obtained by integrating this technique with MaxTract tool [Bake 12] is also provided. The output indicates that our method helps in improving the MaxTract tool performance.
Having a robust line finding algorithm available for documents containing large numbers of mathematical expressions gives us a good basis to tackle our actual goal, identifying tabular layout of mathematical expressions. This chapter presents a cell segmentation method that attempts to handle the unsolved issues discussed in chapter 2.

To appreciate the obstacles encountered in segmenting such tables we have applied an implementation of Kieninger’s algorithm [Kien 98], which was described in chapter 2, to the two tables with mathematical expressions presented in Fig. 4.1. The results of the algorithm is given in Fig. 4.2.

The cell recognition errors are indicated with red and blue stars in Fig. 4.2, respectively. These errors can be classified into typical ones that occur during table segmentation, namely splitting and merging errors. A merging error occurs when two blocks of cells are wrongly transformed into one block. A splitting error occurs if the converse case happens. Splitting errors can be found in table 1 whereas merging errors can be found in table 2.

**Splitting Errors Analysis**

The splitting errors in table 1 in figure 4.2 result in fake columns. The cause of this is the structural nature of mathematical expression that usually have an inconsistent space
If \( R(x) \) is a particular solution to \( y'' + b^2 y = R(x) \) is

\[
\begin{align*}
\sin bx & \quad - \frac{x \cos bx}{2b}, \\
P(x) \sin bx & \quad \frac{\sin bx}{(2b)^2} \left[ P(x) - \frac{P'(x)}{(2b)^2} + \frac{P''(x)}{(2b)^4} + \ldots \right] \\
& \quad - \frac{\cos bx}{2b} \int \left[ P(x) - \frac{P'(x)}{(2b)^2} + \ldots \right] dz.
\end{align*}
\]

Table 1

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( F^{m}(x) = (-1)^{m} (1 - x^2)^{m} \frac{d^n}{dx^n} P(x) )</td>
</tr>
<tr>
<td>2</td>
<td>( P^{m}(x) = (-1)^{m} \frac{\Gamma(n + m + 1)}{\Gamma(n + m + 1)} P^{m}(x) = (1 - x^2)^{m} \int_{0}^{1} \ldots \int_{0}^{1} P_{m}(x)(dx)^{m} )</td>
</tr>
<tr>
<td>3</td>
<td>( P^{m}(x) = (x^2 - 1)^{-m} \int_{1}^{x} \ldots \int_{1}^{x} P_{m}(x)(dz)^{m} )</td>
</tr>
<tr>
<td>4</td>
<td>( Q^{m}(x) = (x^2 - 1)^{-m} \frac{d^n}{dx^n} Q_{m}(x) )</td>
</tr>
<tr>
<td>5</td>
<td>( Q^{m}(x) = (-1)^{m} (x^2 - 1)^{-m} \int_{0}^{\infty} \ldots \int_{0}^{\infty} Q_{m}(x)(dx)^{m} )</td>
</tr>
</tbody>
</table>

Table 2

Figure 4.1: Tables (taken from [Grad 07]) with mathematical expressions before applying the Kieninger’s algorithm.

value distribution between their symbols which in turn makes Kieninger’s algorithm more likely to mistake casual space character alignment with a column delimiter.

Merging Errors Analysis

The merging errors in table 2 in figure 4.2 result in too few cells and more importantly do not detect the correct rows. And while some overlapping cells are detected, it is probably not the overlap one would like to have. The method primarily fails to split columns because there is no discernible space between them relative to the other rows. That is, the consecutive cells in one row obscure the column structure in the next row.
If $P(x)$ is

\[ \sin bx \quad \frac{x \cos bx}{2b} \]

\[ P(x) = \frac{P^0(x) + \int^x P^0(x) dz}{2b^2} \]

\[ \frac{- \cos bx}{2b} \int P(x) = \int P^0(x) + \ldots \ dz. \]

**Table 1**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
</table>
| 1 | $P''(x) = (-1)^m (1 - x^2)^n \frac{d^n P(x)}{dx^n}$ | WH, MO 84, EH I 148[6] *
| 2 | $P_m^0(x) = (-1)^m \Gamma(v + m + 1) P_m^0(x) - (1 - x^2)^n \int P_m^0(x) dz$ | MO 85, EH I 149[11] *
| 3 | $P_m^0(x) = (z^2 - 1)^{\frac{n}{2}} \int P_m^0(z) dz$ | MO 85, EH I 149[10] *
| 4 | $Q_m^0(z) = (z^2 - 1)^{\frac{n}{2}} \frac{dz}{d^m} Q_m^0(z)$ | WH, MO 85, EH I 149[5] *
| 5 | $Q_m^0(z) = (-1)^m (z^2 - 1)^{-\frac{n}{2}} \int Q_m^0(z) dz$ | MO 85, EH I 149[9] *

Figure 4.2: The results of applying Kieninger’s algorithm.

### 4.1 Table Column Extraction

To avoid errors of the above nature, in our approach we identify cells first before combining them to rows and columns. In the absence of a language model we have to rely on purely spatial techniques to identify enough features to determine a cell boundary.

We first use our line finding algorithm in chapter 3 to identify the single lines in the page or area of the table we want to recognise. We then exploit again the histogrammatic information that we have gathered during the line finding algorithm in the following way:

We consider the distribution of horizontal distance values $d$ between consecutive glyphs as given by the histogram derived in the line finding algorithm. However, we now
consider the gaps between the values on the x-axis. As mentioned previously we do not
denote any x-value for which no horizontal distance exists in the page. As a consequence
not all values are given along the axis. We now look for the first difference between neigh-
bouring values that exceeds a particular threshold (in our case we generally take the value
10). We assign the greater of the two difference values to be our cell separation distance
$T$. This procedure can be formally defined as follows:

**Definition 12** (Cell Separation Distance). Let $D = \{d_1, \ldots, d_n\}$ be the ascending-sorted
set of all horizontal distances (as defined in definition 6) that do exist in a page. Then
let $V = \{v_1, \ldots, v_n\}$ be the set of differences between neighbouring elements of $D$, that
is, $v_i = d_i - d_{i-1}$.

The cell separation distance $T$ is then assigned to the smallest $d_i$ such that $v_i \geq \theta$
(this value of $\theta$ is determined empirically), where $i = 1, \ldots, n$.

---

**Algorithm 12**: Calculating cell separation distance $T$ algorithm

```
input : distances a list of the horizontal distances between a page’s glyphs $d$
output: cell separation distance $T$

begin
  sort distances in ascending order according to the horizontal distance values
  let firstItem = head(distances)
  remove head(distances) from distances
  while distances $\neq$ emptylist do
    if head(distances) - firstItem $\geq \theta$ then
      T = head(distances)
      distances = emptylist
    else
      firstItem = head(distances)
      remove head(distances) from distances
  return T
```

---

68
4.1.1 Example

To exemplify the idea assume that we have the following horizontal distances occurring between all glyphs of a given page: 12, 15, 17, 21, 23, 28, 35, 50, 56, 75, 120. Using our threshold parameter of 10 the first gap exceeding this value is between 35 and 50. Consequently, we will choose $T = 50$ and separate every line into columns whenever the horizontal distance between two neighbouring glyphs is greater than or equal 50.

4.1.2 Determining $V$ value

Figure 4.3 shows a histogram that presents in its x-axis different values of $V$ (5, 8, 10, 13, 15, 17) and in its y-axis the precision and recall rates that are obtained by running our technique over 200 pages (the training dataset) each time with different $V$. These precision and recall rates are extracted automatically.

As it can be seen, the corresponding precision and recall rates of $v=10$ is in the highest value. Thus, we use $v=10$ to get the results of running our technique over 994 pages (the testing dataset).

One can notice later, that there is a slight difference between the precision and recall rates with $v=10$ in this histogram, and those in table 4.1. This is because the latter are calculated manually.

The intuition of the definition 12 is that small values $v_i$ are intra-columnar distances, A sudden leap (of 10 pixels) is deemed to mark the transition between intra-columnar and inter-columnar distances, and hence the value $d_i$ at which this occurs $T$ is the correct splitting point. Algorithm 12 presents a pseudo code that shows how to calculate $T$.

It is worth to state that the unit of value 10 in definition 12 is in pixels and the reso-
We then separate lines into columns in the places where the horizontal distance between two neighbouring glyphs is larger than $T$.

**Definition 13** (Cells). Let $L = \{g_1, \ldots, g_n\}$ be a line. We impose a partition on $L$ as follows:

1. two neighbouring glyphs $g_i, g_{i+1}$ belong to the same set if $l(g_{i+1}) - r(g_i) < T$,
2. two neighbouring glyphs $g_i, g_{i+1}$ belong to different sets if $l(g_{i+1}) - r(g_i) \geq T$.

We call the subsets of $L$ given by this partition cells. Algorithm 13 presents a pseudo code that shows how to extract cells from table.

While the choice of this distance measure is fairly simple, it has the advantage that it does not rely on any additional knowledge on the content or elaborate statistical consideration. Nevertheless, in practice we have observed very good results. For example, when running our algorithm over table 1 and table 2 from Fig. 4.1 we obtain the results
Algorithm 13: Extracting cells of table algorithm

input : mergeLines a list of lines
input : cell separation distance T
output: cells of a table

1 begin
2 let clusterG = emptylist
3 let cellOfTable = emptylist
4 while mergeLines ≠ emptylist do
5   let line = head(mergeLines)
6   let maxi = head(line)
7   add maxi to clusterG
8   remove head(line) from line
9   while line ≠ emptylist do
10      if l(head(line)) - r(maxi) >= T then
11         add clusterG to cellOfTable
12         clusterG = (head(line))
13      else
14         add head(line) to clusterG
15         if (r(maxi) < r(head(line))) then
16            maxi= head(line)
17            remove head(line) from line
18      add clusterG to cellOfTable
19      clusterG = emptylist
20     remove head(mergeLines) from mergeLines
21 return cellOfTable
presented in Fig. 4.4, where we can see that we indeed obtain the cells that we would ideally identify before putting together the table.

\[
\begin{align*}
\text{If } R(x) \text{ is } \sin b x & \quad \text{A particular solution to } y' + b^2 y - R(x) \text{ is } \sin b x
\\
\frac{x \cos b x}{2b} & \\

P(x) \text{ sin } b x & \quad \frac{\sin b x}{(2b)^2} \left\{ P(x) - \frac{P'(x)}{(2b)^2} + \frac{P''(x)}{(2b)^4} + \ldots \right\}
\\
& \quad - \frac{\cos b x}{2b} \int [P(x) - \frac{P'(x)}{(2b)^2} + \ldots] \, dx.
\end{align*}
\]

Table 1

\[
\begin{align*}
1 & \quad P_{\nu}^m(x) = (-1)^m (1 - x^2)^{\nu/2} \frac{d^n}{dx^n} P_{\nu}(x) \\
2 & \quad P_{\nu}^{-m}(x) = (-1)^m \Gamma(\nu - m + 1) \frac{d^n}{dx^n} P_{\nu}(x) = (1 - x^2)^{-\nu/2} \int_1^1 \ldots \int_1^1 P_{\nu}(x) (dx)^n \\
3 & \quad P_{\nu}^{-m}(z) = (z^2 - 1)^{-\nu/2} \int_1^1 \ldots \int_1^1 P_{\nu}(z) (dz)^n \\
4 & \quad Q_{\nu}^m(x) = (z^2 - 1)^{\nu} \frac{d^n}{dz^n} Q_{\nu}(z) \quad \text{for } \nu \geq 1 \\
5 & \quad Q_{\nu}^{-m}(z) = (-1)^m (z^2 - 1)^{-\nu} \int_1^1 \ldots \int_1^1 Q_{\nu}(z) (dz)^n
\end{align*}
\]

Table 2

Figure 4.4: Ideal segmentation output of both table 1 and table 2 in figure (4.1)

4.2 Evaluation and Experimental Results

As a means to both evaluate the performance of our approach as well as to later be able to locate tables in a page we adapt a technique from the literature. For example, in [Mand 06, Oro 09] lines on a page are classified as tabular lines, if they can be split into two or more segments, or otherwise as text lines. This classification is then used to identify table cells and consecutive tabular lines are clustered and the top and bottom
boundaries of table calculated based on the first and last.

Although for mathematical documents this technique needs to be modified as not all tabular lines should be considered part of a table, the classification has still a valid use in evaluating our algorithm’s performance where we are only interested in extracting the correct table cells. Consequently we classify lines into two categories: either text or table together with the number of recognised cells. Fig. 4.5 gives an example of a fully classified page.

Figure 4.5: An example of a classified page from [Grad 07].

For our evaluation we have manually ground truthed the first 200 pages of [Grad 07], a document that contains many tabular components. We then ran our technique over the 200 pages, which produces a corresponding classified page for each page in the test set, that contains a mixture of Tab and Txt label annotations.
Table 4.1: Experimental results for cell recognition.

Although our experimental results — presented in Table 4.1 — show already a promisingly high accuracy rate. Running this technique over other dataset is necessary to test the technique’s robustness. To achieve this, the implementation of this techniques is run over 994 pages (taken from [Grad 07]). A manual check of the results is accomplished by going through every page line by line. A rectangle is drawn around the borders of the lines to make the visual checking process easier. Figure 4.6 shows in (a) what it is interpreted as a case of errors and in (b) what it is interpreted as a case of correctness. Table 4.2 shows the output of this step.

Table 4.2: Experimental results of running our technique over 994 pages

Table 4.2 gives a concise information about the obtained results of re-running our technique again over the 994 pages. Although, the data set is almost five times the size of the previous one our detection results remain stable. In fact, one can see that in comparison with Table 4.1 we have even a slight increase of recognition rate. This result gives us confidence about the effectiveness of our technique even on large datasets and documents.

4.3 Summary

In this chapter, a technique for cell segmentation is described. We utilised the histogramatic information presented in chapter 3 to determine where to cut on the lines to obtain cells. Experiments on two datasets show promising results.
Figure 4.6: examples of a case of errors in (a) and a case of correctness in (b)
Part III

Table Recognition
While much work has been done on table recognition this research has been primarily concerned with tables in ordinary text. However, tables containing mathematical structures can differ quite significantly from ordinary text tables and therefore specialist treatment is often necessary. In fact, it is even difficult to clearly distinguish table recognition in mathematics from layout analysis of mathematical formulas. This blurring often leads to a number of possible, equally valid interpretations. However, a reliable understanding of the layout of a formula is often a necessary prerequisite to further semantic interpretation.

The aim of this work is to develop a table recognition algorithm that is particularly good for tables containing mathematical expressions. As the distinction between tables and complex typeset mathematical formulae spanning multiple lines is often difficult, the narrow definition of tables is forgone and instead we consider a far wider range of expressions as tables as is usually the case in the literature.

Since there is no standard convention for composing tables, there is a need of building table representation framework which is flexible enough to deals with tables from various domains. Therefore, the framework, illustrated in section 5.1, is constructed based on abstract concepts that allow for producing the maximum cells, columns and rows which
can be extracted from a table form.

5.1 Multi-Interpretations of Table’s Re-composition

Now, a description of this table interpretation method is given. The input is the bounding box of the table cells. Using these cells, the algorithm first produces the maximum columns and also the maximum cells in each column that can be extracted from a table. This is defined precisely in 5.1.1. Then an initial graph that represents the table grid and the relationship between their nodes (cells) are defined and built. Also a new set of graph rewriting rules that are used to recompose the table cells is illustrated. Finally, a method that produces all possible column combinations is described which works as a basis of finding a set of possible table structure interpretations.

5.1.1 Preprocessing Steps

Based on the definitions concerning cell extraction that are given in chapter 4, we now formally define the basic concepts necessary for the procedure of extracting maximum columns and cells from tables that will form our initial graph.

Definition 14 (Cell’s borders). Let $c$ be a cell, then the limits of its bounding box are defined by $l(c), r(c), t(c), b(c)$ representing left, right, top and bottom limit respectively. We also have $l < r$ and $t < b$.

Definition 15 (Vertical and horizontal overlap between cells). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps vertically with $c_2$ ($c_1 \leftrightarrow c_1$) if we have $[t(c_1), b(c_1)] \cap [t(c_2), b(c_2)] \neq \emptyset$ where $[t(c), b(c)]$ is the interval defined by the top and bottom limit of the cell $c$. Similarly we
define horizontal overlap of two cells $c_1, c_2$ ($c_1 \triangleleft c_1$) by $[l(c_1), r(c_1)] \cap [l(c_2), r(c_2)] \neq \emptyset$.

Before building a graph to represent a table structure, all cells $C$ are first sorted ascendantly with respect to $l(c)$. Then, initial columns are constructed by splitting $C$ on the cell that does not horizontally overlap with all cells which are above it. Algorithm 14 presents a pseudo code that shows how to construct these initial columns.

**Algorithm 14:** Extracting initial columns algorithm

```
input : cellOfTable a list of cells after sorting it in ascending ordering according to \( l(c) \)
output: initial columns of a table
1 begin
2 let tempList = head(cellOfTable)
3 let columnList = emptyList
4 remove head(cellOfTable) from cellOfTable
5 while cellOfTable $\neq$ emptylist do
6     if \( l(head(cellOfTable)) < (\min_{c\in tempList} r(c)) \) then
7       add head(cellOfTable) to tempList
8       remove head(cellOfTable) from cellOfTable
9     else
10       add tempList to columnList
11       tempList = emptyList
12       tempList =head(cellOfTable)
13       remove head(cellOfTable) from cellOfTable
14 add tempList to columnList
15 return columnList
```

**Definition 16** (Initial Columns). Let $C = \{c_1, c_2, ..., c_n\}$ be all cells ascendantly ordered by $l(c)$ and $col$ be a column of table. Then $col = \{c_1, c_2, ..., c_m\}$ if one of the set $[r(c_1), r(c_2), r(c_3), ..., r(c_m)] < l(c_{m+1})$ where $n = 1, 2, ...$ and $m < n$

The table in figure 5.1 is utilised as an example to clarify how to extract these initial columns. First, the cells are sorted ascendantly with respect to $l(c)$. Figure 5.2 shows the
Figure 5.1: Cell identification with table containing multiline expressions that is taken from [Grad 07].

Now, this cells list is split into columns, based on the horizontal overlap between their borders, as follows:

- Take the first cell in the cells list and put it in the first column list of the table.
- Check whether the following cell in the cells list horizontally overlaps with ALL cells in the first column list. If yes, put it into the same column list otherwise add it to a new column list.
- Repeat step 2 until there is no cell left in the cells list.

After applying the above steps on the cells list, we obtain the initial columns that are presented in figure 5.4.

In case there is an absence of a cell which should be beneath or above a cell that is being checked using the step described above, a virtual cell $c'$ is added. When the graph is built later, these virtual cells are represented as nodes. The goals of adding such nodes to the graph are firstly to avoid the complex relationship between nodes and secondly to use such nodes later to detect actual rows. Some examples are shown in Figure 5.3.
Figure 5.2: Sorted cells of the table in figure 5.1

\[ p_c^v(x) = (-1)^n \left( 1 - x^2 \right)^{\frac{n}{2}} \frac{d^n}{dx^n} P_c(x) \]

\[ p_c^v(x) = (-1)^n \frac{\Gamma(n - m + 1)}{\Gamma(n + m + 1)} p_c^v(x) = \left( 1 - x^2 \right)^{-\frac{n}{2}} \int_{x}^{\infty} P_c(x)(dx)^n \]

\[ q_c^v(x) = (x^2 - 1)^{-n} \frac{d^n}{dx^n} Q_c(x) \]

\[ q_c^v(x) = (-1)^n (x^2 - 1)^{-\frac{n}{2}} \int_{x}^{\infty} Q_c(x)(dx)^n \]

\[ [m \geq 1] \]

\[ [m \geq 1] \]

\[ [m \geq 1] \]

Figure 5.3: Examples of virtual cells which their borders appear to be bigger

In order to locate the position of virtual cells, the line definition 3 is recalled to calculate the borders of lines and then use them to detect if there is an existence of a line which has no corresponding cell (belong to a particular column) vertically overlapped with its borders. If so, a virtual cell is added. Algorithm 15 presents a pseudo code that shows how to add these visual cells.

**Definition 17 (Virtual cells).** Let \( col = \{c_1, c_2, ..., c_n\} \) be a column and \( l = \{g_1, g_2, ..., g_n\} \) be a line such that if \( b(c_1) \leq b'(l_1) \) \&\& \( t(c_1) > t'(l_1) \) where \( t'(l_1) = \min_{g \in l_1} t(g) \) and \( b'(l_1) = \max_{g \in l_1} b(g) \) then add the real \( c_1 \) otherwise add a virtual cell \( c'_1 \).
### 5.1.2 Tabular Representation Using Graph Model

All cells produced by the procedure in section 5.1.1 are used to build an initial graph \( G = (N, E) \) to represent the table structure. Each cell \( c \) corresponds to a node in \( N \), with four edges \( E \) for directions west, east, north and south, labelled \( w, e, n \) and \( s \), respectively (with exception of border nodes that have only two or three edges). Furthermore, all possible connections between adjacent nodes must exist.

---

Table: Initial columns of the table in figure 5.1 after adding the virtual cells.

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>[ P^m_n(x) = (-1)^m (1 - x^n)^{\frac{m}{n}} \frac{d^m}{dx^m} P_n(x) ]</td>
<td>[ P^m_n(x) = (-1)^m \frac{\Gamma(n-m+1)}{\Gamma(n+1)} \frac{d^m}{dx^m} P_n(x) = (1 - x^n)^{\frac{m}{n}} \int \cdots \int P_n(x) dx^{m} ]</td>
<td>[ Q^m_n(x) = (x^2 - 1)^{\frac{m}{2}} \frac{d^m}{dx^m} Q_n(x) ]</td>
<td>[ Q^m_n(x) = (-1)^m (x^2-1)^{\frac{m}{2}} \int \cdots \int Q_n(x) dx^{m} ]</td>
</tr>
</tbody>
</table>

---

Figure 5.5: Initial columns of the table in figure 5.1 after adding the virtual cells.

Figure 5.5 shows the columns in figure 5.4 after adding the virtual cells to them. Empty boxes in the figure represent the virtual cells.
Algorithm 15: Adding virtual cells algorithm

input : columnList a list of initial columns
input : mergeLines a list of lines
output: columns of a table after adding virtual cells

begin

let tempLineList = emptyList
let colList = emptyList
let colsList = emptyList

while columnList ≠ emptylist do

    tempLineList = mergeLines

    let column = head (columnList)

    while tempLineList ≠ emptylist do

        if \( b(\text{head(column)}) \leq b'(\text{head(tempLineList)}) \) and
        \( t(\text{head(column)}) \geq t'(\text{head(tempLineList)}) \) then

            add head(column) to colList

            remove head(tempLineList) from tempLineList

        else

            add \( c' \) to colList

            remove head(tempLineList) from tempLineList

    remove head(columnList) from columnList

    add colList to colsList

    colList = emptyList

return colsList
**Definition 18** (Graph Specifications). Let \( n \) be a node which represents a cell \( c \), then the directions of its outgoing edges are defined by \( w(n), e(n), n(n), s(n) \) representing west, east, north and south directions respectively. Such that every \( n \) has at least one outgoing edge at each described directions.

1) **Type of Nodes**: When constructing the initial graph, one can divide the nodes into four types. The classification process is done by checking whether there is a horizontal overlap between columns or not. The motivation of this classification is to distinguish *completely separated* columns that their cells have no horizontal overlap with the cells in other columns that all form a table. One can use this step later to eliminate such columns from the set of possible column combinations. The cells are classified into four types using the following conditions:

**IF** (each of \([l(c), r(c)] \in col1 \cap \text{any of } [l(c), r(c)] \in col2 = \emptyset\) **Then** mark all \( c \in col1 \) with \( R \) and mark all \( c' \in col1 \) with \( V \)

**IF** (at least one of \([l(c), r(c)] \in col1 \cap \text{any of } [l(c), r(c)] \in col2 \neq \emptyset\) **Then** mark all \( c \in col1 \) with \( R^* \) and mark all \( c' \in col1 \) with \( V^* \)

These conditions are applied on all columns of any target table. Such that \( col1 \) on the left of \( col2 \). Table 5.1 shows the node types and how they are treated by the interpreter.

2) **Type of Relationships between nodes (Edges)**: To avoid having complex relationships between nodes, a graph which represents the maximum number of nodes for a table is constructed. This provides us with simple relationship between nodes which are horizontal and vertical edges.
<table>
<thead>
<tr>
<th>Node Types</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>is a real node which must not be merged with other nodes from other columns</td>
</tr>
<tr>
<td>$V$</td>
<td>is a virtual node which must not be merged with other nodes from other columns</td>
</tr>
<tr>
<td>$R^*$</td>
<td>is a real node which can be merged with other nodes to form one of the possible valid table interpretations</td>
</tr>
<tr>
<td>$V^*$</td>
<td>is a virtual node which can be merged with other nodes to form one of the possible valid table interpretations</td>
</tr>
</tbody>
</table>

Table 5.1: Type of nodes.

5.1.3 Construct Initial Graph (Example)

The graph in Figure 5.6 represents the table in Figure 5.1 which illustrates one possibly valid interpretation of the table. As it can be seen, the proposed algorithm succeeded in distinguishing the second column that represents equations from the misaligned third column that represents the conditions associated with these equations and eventually splits them into two columns.

The following chapters introduce two methods for rewriting tables which the first, that described in chapter 6, directly rewrites the initial graph using graph rewriting rules, also illustrated in chapter 6, to infer the table structure. The evaluation of this method is given in chapter 7.

However, the second method, presented in chapter 8, first utilises some other graph rewriting rules to check and fix the initial graph and then rewrites it to infer the table structure. The goal of doing this, is to overcome the issues that the first method can not resolve. The evaluation of this method is given in chapter 9.
Figure 5.6: A graph represents one of the possible interpretations of the table’s columns shown in the table in Fig 5.1
FIRST METHOD: DIRECT REWRITING OF THE INITIAL GRAPH

In this method, a direct rewriting of the initial graph, that is obtained using the steps mentioned in section 5.1, is preformed. To accomplish this, a graph rewriting system is used. The description of this system is given in the following sections.

6.1 Graph rewriting system

The system is introduced to utilise in automatically interpreting the structure of table. The graph defined above is used to represent them. Although, in [Rahg 96] and [Aman 02] the authors have used graph rewriting system before to analyse table layout, due to the complex structure of the table domain that are used in the experiments, a new set of rewriting production rules are produced.

Let $N$ and $E$ represent a specific set of nodes and a specific set of relationships between nodes called edges respectively. Then, a graph rewriting system can be represented by the following tuple $g = \{N, E, P, ER\}$ where $P$ are rewriting production rules which have the form $lhs \rightarrow rhs$. This specifies two graphs where the subgraph $lhs$ in a host graph $(G)$ can be replaced with a graph $rhs$. The embedding relations $ER$ associated with each rewrite rule $lhs \rightarrow rhs$ specify how the new subgraph $rhs$ is connected to the remainder.
Figure 6.1: Example of production rule

graph of the host graph $G$, after $lhs$ is removed. The notation containing a tuple of the form $\{(n_1,e_1,n_2,n_3,e_2,n_4): n_1,n_2,n_3,n_4 \in N; e_1,e_2 \in E\}$ is used to represent embedding relations $ER$ where $n_1,n_2$ and $e_1$ represent two nodes and an edge between them in $lhs$ that must be replaced by the $n_3,n_4$ nodes and $e_2$ in $rhs$ to maintain the integrity of the graph. Figure 6.1 shows an example of production rule.

Embedding rule $ER$ which tells edge label conversion from $lhs$ to $rhs$ for the production rule shown in Figure 6.1 is expressed as follows:

$$
ER = ((1V^*,l,7N,5V^*,l,7N),(7N^*,r,1V^*,7N,r,5V^*),(1V^*,t,14N,5V^*,t,14N),
(14N,b,1V^*,14N,b,5V^*),(2V^*,t,13N,5V^*,t,13N),(13N,b,2V^*,13N,b,5V^*),
(2V^*,r,12N,5V^*,r,12N),(12N,l,2V^*,12N,l,5V^*),(3R^*,l,8N,6R^*,l,8N),
(8N^*,r,3R^*,8N,r,6R^*),(3R^*,b,9N,6R^*,b,9N),(9N,t,3R^*,9N,t,6R^*),
(4V^*,b,10N,6R^*,b,10N),(10N,t,4V^*,10N,t,6R^*),(4V^*,r,11N,6R^*,r,11N),
(11N,l,4V^*,11N,l,6R^*))
$$
6.1.1 Table Production Rules

A set of production rules that are needed to recompose all possible table interpretations are shown in Figure 6.2. This figure illustrates these rules which should cover the different cases of node combinations. By using these rules, one can produce a set of possible table interpretations.
6.1.2 Number of rules needed to cover any node combinations

Since the nodes, that should be gathered in different combinations, are classified into two types $R^*$ and $V^*$, binomial coefficients are the right method for finding how many rules are needed to completely cover all different cases of node combinations. Binomial coefficients, in mathematics, are a group of positive integer numbers that arise as coefficients in the binomial theorem. These coefficients are indexed using two non-negative numbers. The binomial coefficient is often written $\binom{n}{k}$.

This group of positive numbers occurs in several fields of mathematics beside algebra, especially in combinatorics. For any set comprising $n$ elements, the number of independent $k$-element sub-sets of this set that can be composed (in other words, the $k$-combinations of $n$ elements) is expressed using the binomial coefficient $\binom{n}{k}$. Therefore $\binom{n}{k}$ is commonly read as (n choose k).

**Theorem 1** (The Binomial Theorem). Let $a$ and $b$ be any numbers and $n$ be any integer number that is greater than 0. Then

$$(x + y)^n = \sum_{k=0}^{n} \binom{n}{k} x^k y^{n-k}$$

To find out how many rules are needed for our framework to cover all cell combinations. Put $x = 1$ and $y = 1$ in the equation above where 1 in both variables $x$ and $y$ represents the two types of nodes mentioned above and $n$ represents number of nodes involved in composing the rules. We get

$$(1 + 1)^n = \binom{n}{0}1^0.1^{n-0} + \binom{n}{1}.1^1.1^{n-1} + ... + \binom{n}{n-1}.1^{n-1}.1^{n-(n-1)} + \binom{n}{n}.1^n.1^{n-n}$$

hence

$$2^n = \binom{n}{0} + \binom{n}{1} + \binom{n}{2} + ... + \binom{n}{n-2} + \binom{n}{n-1} + \binom{n}{n}.$$
Figure 6.3: Some examples of how to applying the rewriting rules

If \( n \) in our case is equal to 4. Then, \( 2^4 = 16 \) rules, which each contains four nodes, are enough for covering all possible cell combinations in a table.

**Case-analysis: The specification of rules that can rewrite a whole graph representation of table**

As it is described above in section 5.1, the proposed framework deals with a graph represents the maximum cells that can be found in a table. This means that all rows have the same number of cells (nodes) and this also means that for each two nodes in a row, there is two nodes in its adjacent row. As a consequence, square rules where each one contains four nodes can rewrite any table contains even rows. We achieve this by applying these rules on the first two columns then on the resulted column with a new column. Figure 6.3 shows some examples of how to applying such rules.

Since there is four nodes for each rule and these nodes are of two types \( V^* \), \( R^* \), the binary representation of the number 0 to 15 (which equal to the needed number of rules for covering all possible cell combinations) is feasible method to work out what type of nodes each rule should contain. This is accomplished by donating 0 and 1 to \( V^* \) and \( R^* \) respectively. Table 6.1 presents the number from 0 to 15, their binary representation and
their corresponding $V^*$ and $R^*$.

<table>
<thead>
<tr>
<th>No</th>
<th>Binary Representation</th>
<th>Corresponding $V^<em>$ and $R^</em>$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0 0 0 0</td>
<td>$V^<em>$ $V^</em>$ $V^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>1</td>
<td>0 0 0 1</td>
<td>$V^<em>$ $V^</em>$ $V^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>2</td>
<td>0 0 1 0</td>
<td>$V^<em>$ $V^</em>$ $R^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>3</td>
<td>0 0 1 1</td>
<td>$V^<em>$ $V^</em>$ $R^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>4</td>
<td>0 1 0 0</td>
<td>$V^<em>$ $R^</em>$ $V^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>5</td>
<td>0 1 0 1</td>
<td>$V^<em>$ $R^</em>$ $V^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>6</td>
<td>0 1 1 0</td>
<td>$V^<em>$ $R^</em>$ $R^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>7</td>
<td>0 1 1 1</td>
<td>$V^<em>$ $R^</em>$ $R^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>8</td>
<td>1 0 0 0</td>
<td>$R^<em>$ $V^</em>$ $V^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>9</td>
<td>1 0 0 1</td>
<td>$R^<em>$ $V^</em>$ $V^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>10</td>
<td>1 0 1 0</td>
<td>$R^<em>$ $V^</em>$ $R^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>11</td>
<td>1 0 1 1</td>
<td>$R^<em>$ $V^</em>$ $R^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>12</td>
<td>1 1 0 0</td>
<td>$R^<em>$ $R^</em>$ $V^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>13</td>
<td>1 1 0 1</td>
<td>$R^<em>$ $R^</em>$ $R^<em>$ $V^</em>$</td>
</tr>
<tr>
<td>14</td>
<td>1 1 1 0</td>
<td>$R^<em>$ $R^</em>$ $R^<em>$ $R^</em>$</td>
</tr>
<tr>
<td>15</td>
<td>1 1 1 1</td>
<td>$R^<em>$ $R^</em>$ $R^<em>$ $R^</em>$</td>
</tr>
</tbody>
</table>

Table 6.1: The binary representation and their corresponding $V^*$ and $R^*$.

The rules in Figure 6.2 represent all combinations that show in Table 6.1. One can note that there are, in figure 6.4, four rules with two nodes where $n = 2$, these are to use, for instance, in tables that the number of their rows are odd. Figure 6.5 shows an example of how to apply these rules on a table.

### 6.1.3 Recognizing Rows Using Virtual Nodes

After constructing the initial graph that contains the virtual nodes, inferring possible interpretations for a table’s rows becomes feasible. One of these interpretations can be obtained by observing the virtual nodes in the first column and merging their lines with the line above like in [Chao 03]. We illustrate this step by utilizing the graph in Figure 5.6. By looking at Figure 6.6 it is clear that the rows are correctly clustered and gives one possible valid interpretation for table rows.
Since almost all the tables in the dataset we used have a common feature which states that each row starts with a real node \( R \) or \( R^* \) in its first column and end by the appearing of the next real node \( R \) or \( R^* \) at the first column of any row. This means that using this fact in recognising the rows would give mainly perfect results. However, this cannot be generalised to all kinds of table structures. One can still use this fact in extracting tables from other datasets but since the common feature in the used dataset might not be that common in tables of other datasets, the results are expected to be humble.
6.2 A Framework For Choosing Column Combinations

Before the rewriting rules can be applied on a table, an approach for determining which table columns are to be combined is applied on the table, that contains the maximum columns as described in section 5.1, to obtain the set of possible tables that should be rewritten. The framework produces all possible column combinations. Since, we can divide the type of columns into two which are either column which contains cells that are horizontally overlapped with its adjacent column’s cells “indicated by 1” or column which has no cells that are horizontally overlapped with its adjacent column’s cells “indicated by 0”, using binomial coefficient theorem (discussed in Sec. 6.1.2) would be a feasible solution for representing all of possible column combinations. To illustrate this technique in more detail and how to avoid replicating some of column combinations results, an example is showed along with the steps of this method.

6.2.1 Number of possible column combinations

Two to the power of \( n \), written as \( 2^n \), is the number of ways the bits in a binary word with length \( n \) can be arranged. This approach can be adapted to achieve the goal of obtaining
Figure 6.7: Example (taken from [Grad 07]) to show how the framework for choosing column combinations works

all possible column combinations. For representing this framework, columns containing some cells that are horizontally overlapped with some of their adjacent columns cells marked by “1” and columns that are stand alone without any overlapping marked by “0”.

Figure 6.7 shows an example of table contains four columns. The first column does not have any overlapping with other columns where the second, third and fourth are horizontally overlapped with each others. Applying the proposed framework on the table in figure 6.7, the result will be 0111. Based on this number, the all possible table column combinations are equal to $2^3 = 8$ which show in table 6.2:

Note: in this example $n=3$ indicates the number of columns that their cells are horizontally overlapped with each others.

<table>
<thead>
<tr>
<th>No of Possible Combination</th>
<th>Possible Column Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>0001</td>
</tr>
<tr>
<td>3</td>
<td>0010</td>
</tr>
<tr>
<td>4</td>
<td>0011</td>
</tr>
<tr>
<td>5</td>
<td>0100</td>
</tr>
<tr>
<td>6</td>
<td>0101</td>
</tr>
<tr>
<td>7</td>
<td>0110</td>
</tr>
<tr>
<td>8</td>
<td>0111</td>
</tr>
</tbody>
</table>

Table 6.2: All possible table column combinations
The possible combinations that produce the same output are eliminated. In table 6.2 the possible combinations of 0000, 0001, 0010, 0100 and 0101 can be replaced by one of them, for instance, 0000. This is because all of them give us the same possible combination of table contains four columns. Table 6.3 shows the final result.

<table>
<thead>
<tr>
<th>No of Possible Combination</th>
<th>Possible Column Combination</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0000</td>
</tr>
<tr>
<td>2</td>
<td>0011</td>
</tr>
<tr>
<td>3</td>
<td>0110</td>
</tr>
<tr>
<td>4</td>
<td>0111</td>
</tr>
</tbody>
</table>

Table 6.3: Final result of all possible table column combinations

The output of applying the possible column combination of 0000 on the table in figure 6.7 is illustrated in figure 6.8(a). As it can be seen, the table contains four columns. Each column is given a different colour. Likewise, figure 6.8(b) and figure 6.9(c)(d) illustrate the result of applying 0011, 0110 and 0111 on the table in figure 6.7 which shows tables contain three, three and two columns respectively.

6.3 Summary

In this chapter, a table representation technique is described, which attempts to tackle the problems mentioned in section 2.13. Firstly, preprocessing steps are precisely defined to prepare and arrange the cells (that are extracted using the method in chapter 4). These steps would cluster the misaligned cells to their correct columns. Then, virtual cells are added to these columns for obtaining the maximum cells that can be found in a table and also for simplifying the relationships that can be found between cells. Secondly, a graph model is specified for representing the cells of table (obtained from the previous steps). The nodes (that represents the cells) are classified into four types. Then, a set of graph rewriting rules are composed to utilise in automatically interpreting the structure of a table. a description of a way to recognise table rows using virtual nodes are
Figure 6.8: Examples of table (taken from [Grad 07]) output-part1

also given. Finally, a framework for finding a set of possible cell combinations is explained

In the next chapter, an evaluation of the technique that was presented in this chapter is given.
(c)

\[ \int_0^a x P_n (1-2x^2) Y_\nu (xg) \, dx = x^{-1} g^{-1} [S_{2n+1}(g) + \pi Y_{2n+1}(g)] \]
\[ n = 0, 1, \ldots; \quad g > 0, \quad \nu > 0 \]
\[ \text{ET II 109(1)} \]

\[ \int_0^a x P_n (1-2x^2) K_\nu (xg) \, dx = g^{-1} \left( (-1)^{n+1} K_{2n+1}(g) + \frac{1}{2} S_{2n+1}(g) \right) \]
\[ g > 0 \]
\[ \text{ET II 130(1)} \]

\[ \int_0^a x P_n (1-2x^2) J_\nu (xg) \, dx = g^{-1} J_{2n+1}(g) \]
\[ g > 0 \]
\[ \text{ET II 13(1)} \]

\[ \int_0^a x P_n (1-2x^2) |J_\nu (ax)|^2 \, dx = \frac{1}{2(2n+1)} \left( |J_n(a)|^2 + |J_{n+1}(a)|^2 \right) \]
\[ \text{ET II 338(30)} \]

(d)

Figure 6.9: Examples of table (taken from [Grad 07]) output-part2
CHAPTER 7

EVALUATION OF THE FIRST TABULAR RECOGNITION METHOD

In this chapter, the effectiveness of the technique in chapters 5 and 6 is demonstrated by first showing an example of how to apply this technique on a table, then applying it to a set of mathematical tables from standard textbook [Grad 07] that has been manually ground-truthed. Finally, new experimental approaches to improve the initial output of the technique is introduced which show promising results.

7.1 Applying the rewriting system on a table

In this section, a way of applying this system on a table is illustrated to show how one can get a set of interpretations of this table structure. Since there is no semantic information associated with the framework proposed in this chapter, it is very hard if it is not possible to correctly decide the final spatial relationships that should be between a set of nodes after rewriting. This is because of the following handicaps:

- **Mis-aligned cells**: this happens when either two cells from different columns are horizontally overlapped where they should not (For instance see figure 6.8(a)) or two cells are not horizontally overlapped where they should (For instance see first row in figure 7.1 where there is a spanning cell).

- **Fake virtual nodes**: since there is no way of distinguishing a real visual node
Figure 7.1: Examples of cells are not horizontally overlapped where they should be.

Figure 7.2: Examples of elusive empty cells.

from a node that represents an empty cell, there would be no feasible approach for inferring spanning cells as well as empty cells (Figure 7.2 shows an example of this case).

For these reasons, an intermediate form is created using the framework proposed. This is accomplished by putting some constraints on interpreting a set of nodes along with applying the production rewriting rules on them. Since these constraints are extracted using a local analysis of tables structure in particular dataset, it is worth mentioning that, for each different table dataset, these constraints would usually be modified or even some
new ones are added to them.

**Constraints for our dataset**

By observing our dataset of 150 tables that are taken from [Grad 07]. It is found that:

- There are no obvious empty cells at all.
- Since there are many mis-aligned cells, there is not clear-cut feature that tell apart spanning cells.

Therefore, to get the intermediate form for this dataset, the following constraints are utilised:

- Remove virtual cells from columns that contains more than one cell in their rows.
- If a row in column contains only virtual cells, replace them with one empty cell.

**An example**

A table in figure 6.7 is used to illustrate the rewriting system steps. First, we apply the framework of choosing column combinations, that is explained in the chapter 5, on this table. As it can be seen in figures 6.8 and 6.9, there are four possible column combinations of this table. We use the table in figure 6.9(c), which contains 3 columns and second column contains two sub-columns, to show how to apply the production rewriting rules with the described constraints. The initial graph in figure 7.3 represents this table structure.

The rewriting process starts form the Top-left cell and goes through the rows. Columns that contain one cell in its rows remain the same, only the virtual cells interpreted as empty cells. The production rewriting rules are applied on columns that contain more than one cell in its rows. In this example, the only column with this type is the second column.
Therefore, cells (1,2), (1,3), (2,2), (2,3) (see figure 7.4) will be the first nodes to be rewritten.

Since these nodes match the nodes in \textit{lhs} of rewriting rule number 8 shown in figure 6.2, the \textit{rhs} of this rule replace these nodes. After the replacing process, some conversions in the edges is required. If we suppose that the four nodes are 1\(R^\ast\), 2\(V^\ast\), 3\(V^\ast\), 4\(R^\ast\), the two nodes in the \textit{rhs} are 5\(R^\ast\), 6\(R^\ast\) and 7\(R\), 8\(V\), 9\(V^\ast\), 10\(V^\ast\), 11\(V\), 12\(V\) are the adjacent nodes of the four nodes start from the left and go anticlockwise (see figure 7.4), then the required edge conversions can be expressed by \(ER\), that described in section 6.1, as the
$ER = ((1R^*, l, 7R, 5R^*, l, 7R), (7R, r, 1R^*, 7R, r, 5R^*),
(2V^*, r, 12V, 5R^*, r, 12N), (12V, l, 2V^*, 12V, l, 5R^*), (4R^*, l, 8V, 6R^*, l, 8V),
(8V^*, r, 4R^*, 8V, r, 6R^*), (4R^*, b, 9V^*, 6R^*, b, 9V^*), (9V^*, t, 4R^*, 9V^*, t, 6R^*),
(4R^*, b, 10V^*, 6R^*, b, 10V^*), (10V^*, t, 4R^*, 10V^*, t, 6R^*), (4R^*, r, 11V, 6R^*, r, 11V),
(11V, l, 4R^*, 11V, l, 6R^*)$
After completing these conversions, we obtain the result that is illustrated in the table in figure 7.5. Likewise the above first rewriting, the rewriting process continues over the other nodes of the second column. The final result of this table can be seen in figure 7.6. Next, an evaluation of this rewriting system and experimental results on 150 tables are discussed in detail.

### 7.2 Experimental Results

To accomplish the table recognition evaluation, preparing table ground-truthing is usually needed. In [Hu 01] the author stated that, in some cases, the researchers who are ground-truthing tables might have different opinions about the right interpretation of a table. Sometimes, several interpretations seem to be justifiable and appear to be equally valid. Taking into account this fact and for evaluation purposes, 150 tables were manually ground-truthed, such that, each table has all possible interpretations that can be extracted from it. To facilitate the comparison procedure, a visual technique is designed which allows us to visually assess the table recognition output. The technique draws rectangles around table cells. Each column’s cells are given a unique colour to their rectangles. Experiments are done using 100 pages taken from [Grad 07] which contains 150 tables. Table 7.1 illustrates concise information about the experimental results. In this table, the 150 tables are categorised to three groups based on the number of possible interpretations that can be obtained from a table. This can be accomplished using the ground-truth tables. A comparison between outputted possible table interpretations and the corresponding ground-truth table is then manually done. The results of this comparison are classified into three categories. This is determined by observing how far one possible interpretation of the table from the proposed system matches one possible interpretation of the table according to the ground truth set. These three categories are:

1) Table interpretations that completely and correctly extracted (TICE). An output table is classified under this category if it 100% matches.

2) Table interpretations partially
extracted (TIPE). Here the matching rate is greater than or equal 75%. 3) Tables interpretations that are considered as missed (TIM). In this cases, the matching rate is less than 74%.

<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpretations</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TIPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TIM</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>68</td>
</tr>
<tr>
<td></td>
<td></td>
<td>66</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>127</td>
<td>8</td>
<td>1016</td>
</tr>
<tr>
<td></td>
<td></td>
<td>692</td>
</tr>
<tr>
<td></td>
<td></td>
<td>192</td>
</tr>
<tr>
<td></td>
<td></td>
<td>132</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>48</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
</tr>
</tbody>
</table>

Table 7.1: Results of applying the proposed table interpretation technique on 150 tables

Although our experimental results presented in Table 7.1 show already a promisingly high accuracy rate, running this technique over other dataset is necessary to test the technique robustness. To achieve this, the implementation of this technique is run over 110 tables (also taken from [Grad 07]). Table 7.2 shows the output of this step.

<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpretations</td>
<td>Total</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TIPE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TIM</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>800</td>
</tr>
<tr>
<td></td>
<td></td>
<td>656</td>
</tr>
<tr>
<td></td>
<td></td>
<td>90</td>
</tr>
<tr>
<td></td>
<td></td>
<td>54</td>
</tr>
<tr>
<td>9</td>
<td>16</td>
<td>144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>96</td>
</tr>
<tr>
<td></td>
<td></td>
<td>32</td>
</tr>
<tr>
<td></td>
<td></td>
<td>16</td>
</tr>
</tbody>
</table>

Table 7.2: Results of applying the proposed table interpretation technique on 110 tables

7.2.1 Analysis of table interpretations that are partially extracted or missed

One can see that there is still a considerable problem with the mis-clustering of some cells to the wrong column (in the case of table interpretations that are partially extracted) and the failure of splitting two columns (in the case of table interpretations that are missed). An analysis of these cases yields that the majority of mis-clustering and failure cases are
Figure 7.7: Example of table (taken from [Grad 07]) re-composition (a) before manual intervention (b) after manual intervention due to the preprocessing step when it fails to assemble the cells into their proper columns. One possible approach to tackle this problem and eventually improve the accuracy of the proposed approach is to manually intervene by adding marks on tables which assist the proposed method in inferring the correct column and as a result, extract the all possible valid table interpretations. Figure 7.7 illustrates how adding marks on tables improves the accuracy rate.

Figure 7.7 shows tables before and after the manual intervention. Each column in this table is bordered and given a number. By observing table (a) it can be clearly seen that the last cell in the first row is wrongly clustered to the third column where it should have been gathered with the cells in fourth column. Table (b) shows a solution of the problem by adding an empty rectangle over the third column to tell the proposed method that the last cell in the first row is not overlapped with all cells in third column and therefore, it should be gathered with cells in fourth column.
Figure 7.8: (a) Isolated table (taken from [Grad 07]), (b) The same table but maintained in its page

In addition to the manual intervention approach that used to improve the proposed method accuracy rate, some case studies have been proposed to represent two automatic approaches that might improve the initial performance of the proposed method represented in table 7.1.

### 7.2.2 Table Re-composition: within documents

This case study checks the ability of the proposed method to correctly recompose tables which exist within its other page components. Table 7.3 concisely represents the results of running the method over the 150 tables that remain within their pages.

Figure 7.8 shows a table taken from [Grad 07]. The table in (a) is isolated from its page. However, the same table in (b) is maintained in its page. One can see that in Figure 7.8 (a), there are misaligned cells which cause the column contains mathematical
7.2.3 Table Re-composition: combining tables

This time, tables are first clipped and then merged in pages to observe how this would increase or decrease the accuracy rate of the method output. Table 7.4 concisely represents the results of running the method over the 150 tables.

Figure 7.9: (a) Isolated table (taken from [Grad 07]), (b) The same table but combined with other table equations and the one contains conditions to be wrongly clustered together and form one column. However, in Figure 7.8 (b) this problem is resolved by other cells in other components of the page.
<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpretations Total</td>
<td>No. of TICE No. of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIPE TIM</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>68 58 4</td>
</tr>
<tr>
<td>127</td>
<td>8</td>
<td>1016 738 243</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>96 60 12</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.3: Table Re-composition: within documents

<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpretations Total</td>
<td>No. of TICE No. of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIPE TIM</td>
</tr>
<tr>
<td>17</td>
<td>4</td>
<td>68 56 6</td>
</tr>
<tr>
<td>127</td>
<td>8</td>
<td>1016 708 256</td>
</tr>
<tr>
<td>6</td>
<td>16</td>
<td>96 55 14</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 7.4: Table Re-composition: combining tables

it can be seen that in Figure 7.9 (a), there are misaligned cells which cause the column contains mathematical equations and the one contains conditions to be wrongly clustered together and form one column. However, in Figure 7.9 (b) this problem is resolved by other cells in other components of the page.

The results in the tables 7.3 and 7.4 show that using the approaches represent in the case studies 7.2.2 and 7.2.3 improves the results in table 7.1. However, more experiments on different table domains are needed to ensure the robustness of the proposed method.

### 7.3 Summary

In this chapter, experimental evaluation on the framework, that is described in chapters 5 and 6, is accomplished. 150 tables (taken from [Grad 07]) were ground-truthed to use it in assessing the performance of the proposed technique. Initial results are obtained which are then improved by using three new cases-studies approaches.
CHAPTER 8

SECOND METHOD: PRE-PROCESS REWRITING ON THE INITIAL GRAPH

8.1 Introduction

As in, chapter 5, a new table recognition technique is described. However, in this time, we exploit the multi spatial relationships between a table’s cells that were observed on a wide range of tables. The technique uses the same graph, illustrates in chapter 5, but then cleverly extract some cells or columns (if any) that the heuristic technique of the first method, in chapter 5 fails to do so. This method has some similarities with the technique introduced in [Aman 02]. However, we are working on a different table domain that contains misaligned cells which, in turn, may have more than one possible interpretation of table structure [Alka 13a]. As a consequence, a graph representation model as well as a new set of graph rewriting rules are proposed to deal with the requirements needed for our table interpretation process.

8.2 Types of spatial relationships between cells

In order to precisely define these multi spatial relationships [Aiel 09, Gt 94], we first have to agree on how to express some concepts regarding cell’s borders, vertical and horizontal overlaps between table cells.
Figure 8.1: Different arrows represent different relationships between cells

**Arrow representation**

We define some arrow drawings in figure 8.1 that represent the different relationships between nodes to use them later in expressing the production rules. Each of these arrow drawings illustrates one of the relationships formally defined in this section.

We can now formally define the multiple spatial relationships that can be found between cells. These relationships are ordered, based on our observations of the 150 table dataset mentioned in chapter 7, from most significant to least significant. We also observed that there are eleven relationships between any two cells of the tables in the targeted dataset which are:

1. Relationships between adjacent cells are observed between every cell $c_1$ and its neighbouring cell $c_2$ such that projecting the limits of one cell on the other must not cross any other cells of $C$ which denotes all cells in a table.

**Definition 19** (Adjacent cells). For any cells $c_1, c_2 \in C$ with $c_1 \neq c_2$, we say $c_1$ is adjacent to $c_2$ if for all $c_3 \in C \setminus \{c_1, c_2\}$ it holds that:

   (i) $[\min(b(c_1), b(c_2)), \max(t(c_1), t(c_2))] \cap [t(c_3), b(c_3)] = \emptyset$ in the case $c_1$ overlaps horizontally with $c_2$. 

111
2. Two cells that are vertically overlapped where the start and end y-axis borders of one cell is within the start and end y-axis borders of other cell. Figure 8.2 shows an example of these two cells and the unique geometric relationships between them (edge).

**Definition 20** (Interior vertical Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps internally and vertically with $c_2$ or $c_2$ overlaps internally and vertically with $c_1$ if we have $(t(c_1) > t(c_2) \text{ and } b(c_1) < b(c_2)) \text{ OR } (t(c_2) > t(c_1) \text{ and } b(c_2) < b(c_1))$ respectively.

a. Two cells that are vertically overlapped and have the same start and end y-axis borders values. Figure 8.3 shows an example of these two cells and the unique geometric relationships between them (edge).

**Definition 21** (Fully matched vertical Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps fully and vertically with $c_2$ if we have $t(c_1) = t(c_2)$ and $b(c_1) = b(c_2)$

b. Two cells that are vertically overlapped where the interval of start and end y-axis borders of one cell is in the middle of the start and end y-axis borders of other cell. Figure 8.4 shows an example of these two cells and the unique geometric relationships between them (edge).
Definition 22 (Central vertical Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps centrally and vertically with $c_2$ or $c_2$ overlaps centrally and vertically with $c_1$ if we have $\left( t(c_1) - t(c_2) = b(c_2) - b(c_1) \right)$ OR $\left( t(c_2) - t(c_1) = b(c_1) - b(c_2) \right)$ respectively.

3. Two cells that are vertically overlapped where the end y-axis border of one cell is greater than the start y-axis border and less than the end y-axis border of other cell. Figure 8.5 shows an example of these two cells and the unique geometric relationships between them (edge).

Definition 23 (Partial vertical Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps partially and vertically with $c_2$ or $c_2$ overlaps partially and vertically with $c_1$ if we have $\left( b(c_1) > t(c_2) \right)$ and $\left( b(c_1) < b(c_2) \right)$ and $\left( t(c_1) < t(c_2) \right)$ OR $\left( b(c_2) > t(c_1) \right)$ and $\left( b(c_2) < b(c_1) \right)$ and $\left( t(c_2) < t(c_1) \right)$

4. Two cells that are vertically overlapped and have the same start or end y-axis border values but not both. Figure 8.6 shows an example of these two cells and the unique geometric relationships between them (edge).

Definition 24 (Sided vertical Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps one-sidedly and vertically with $c_2$ or $c_2$ overlaps one-sidedly and vertically with $c_1$ if we have $\left( t(c_1) = t(c_2) \right)$ and $\left( b(c_1) \neq b(c_2) \right)$ OR $\left( t(c_1) \neq t(c_2) \right)$ and $\left( b(c_1) = b(c_2) \right)$
5. Two cells that are horizontally overlapped where the start and end x-axis borders of one cell is within the start and end x-axis borders of other cell. Figure 8.7 shows an example of these two cells and the unique geometric relationships between them (edge).

**Definition 25** (Interior horizontal Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps internally and horizontally with $c_2$ or $c_2$ overlaps internally and horizontally with $c_1$ if we have $(l(c_1) > l(c_2) \text{ and } r(c_1) < r(c_2)) \text{ OR } (l(c_2) > l(c_1) \text{ and } r(c_2) < r(c_1))$ respectively.

a. Two cells that are horizontally overlapped and have the same start and end x-axis border values. Figure 8.8 shows an example of these two cells and the unique geometric relationships between them (edge).

**Definition 26** (Fully matched horizontal Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps fully and horizontally with $c_2$ if we have $l(c_1) = l(c_2)$ and $r(c_1) = r(c_2)$

b. Two cells that are horizontally overlapped where the interval of start and end x-axis borders of one cell is in the middle of the start and end x-axis borders of other cell. Figure 8.9 shows an example of these two cells and the unique geometric relationships between them (edge).
Definition 27 (Central horizontal Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps centrally and horizontally with $c_2$ or $c_2$ overlaps centrally and horizontally with $c_1$ if we have $(l(c_1) - l(c_2) = r(c_2) - r(c_1))$ OR $(l(c_2) - l(c_1) = r(c_1) - r(c_2))$ respectively.

6. Two cells that are horizontally overlapped where the end x-axis border of one cell is greater than the start x-axis border and less than the end x-axis border of other cell. Figure 8.10 shows an example of these two cells and the unique geometric relationships between them (edge).

Definition 28 (Partial horizontal Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ overlaps partially and horizontally with $c_2$ or $c_2$ overlaps partially and horizontally with $c_1$ if we have $(r(c_1) > l(c_2) \text{ and } r(c_1) < r(c_2) \text{ and } l(c_1) < l(c_2))$ OR $(r(c_2) > l(c_1) \text{ and } r(c_2) < r(c_1) \text{ and } l(c_2) < l(c_1))$

7. Two cells that are horizontally overlapped and have the same start or end x-axis border values but not both. Figure 8.11 shows an example of these two cells and the unique geometric relationships between them (edge).
**Definition 29** (Sided horizontal Overlap). Let $c_1, c_2$ be two cells. We say $c_1$ *one-sidedly and horizontally overlaps* with $c_2$ or $c_2$ *one-sidedly and horizontally overlaps* with $c_1$ if we have $l(c_1) = l(c_2)$ and $r(c_1) \neq r(c_2)$ OR $l(c_1) \neq l(c_2)$ and $r(c_1) = r(c_2)$.

### 8.2.1 Experiments

The following tables 8.1 and 8.2 show statistical numbers that correspond to the total occurrence of every relationship between cells mentioned above. A dataset of 110 tables that are taken from [Grad 07] is used for testing and obtaining these numbers. One can infer from these tables that, although some types of relationship between cells appear a small number of times, all of these relationships do occur between table cells and therefore we have to consider them when we attempt to correctly recompose table cells. Next, putting into account these spatial relationships illustrated in section 8.2, we introduce an approach that uses a graph model to represent table structure. Then, we produce graph rewriting rules that contribute to automatically interpreting the possible table structure.
8.3 Graph Rewriting System

After determining the possible relationships between table cells, a graph rewriting approach of recomposing these cells to form a possible table layout structure is described in the next sections. First of all, a graph representation model of table structure is illustrated. Then, a description of three components of graph rewriting system, that are utilised to rewrite a table graph, is given. After that, a set of rewriting rules, which are inferred by observing tables in [Grad 07], are formally expressed. Finally, table structure analysis and an example of how to apply some of the graph rewriting rules are discussed.

8.3.1 Graph representation model

We use a graph model $G$ to represent table layout structure where the geometric relationships that occur between table cells are represented by the edges and the nodes represent the cells themselves. For example, figure 8.13 is the graph representation of the table in figure 8.12. Due to the narrow spaces between nodes, one can notice that not all relationships found between nodes (cells) are represented in the figure 8.13. This representation model opens the gate for expressing table layouts using grammars by representing all layout relationships that any two cells can have.

8.3.2 Graph rewriting rules

Several rewriting rules are composed to assist in interpreting the layout structure of tables. The purpose of constructing these rules is to rewrite the graph that represents a table so
that we have a possible interpretation of the table layout structure. Before representing these rules, we first state what the graph rewriting approach consists of. In [Rahg 96] graph rewriting rules are encompassed of three components which are:

**Production Rules**: These rules have a form of $g_l \rightarrow g_r$ where $g_l$ denotes the subgraph of an initial graph of table $G$ and $g_r$ denotes the subgraph that might replace $g_l$. Next, several production rules are illustrated which help in coming up with a set of possible table structure interpretations.

**Embedded Notations**: The role of this component is to monitor and preserve the integrity of the graph $G$ by showing the required conversions on the edges within $G$ when a production rule $g_l \rightarrow g_r$ is applied. A six tuple $(n_1, e_1, n_2, n_3, e_2, n_4)$ is used to expressed this notation where $n_1, n_2$ and $e_1$ represent two nodes and an edge from $g_l$ respectively, which can be replaced by $n_3, n_4$ and $e_2$ that represent two nodes and an edge from $g_r$ respectively.

**Application Conditions**: These conditions are associated with each production rule. They determine when a production rule might be applied. They are typically expressed as constraints or predicates on the node and edge attributes. The conditions on a rule

\[
P_{\nu}^{\alpha}(x) = (-1)^{m} (1 - x^2)^{-\frac{\alpha}{2}} \frac{d^{m}}{dx^{m}} P_{\nu}(x) \]

\[
P_{\nu}^{-m}(x) = (-1)^{m} \frac{\Gamma(\nu - m + 1)}{\Gamma(\nu + m + 1)} P_{\nu}^{m}(x) - (1 - x^2)^{-\frac{\alpha}{2}} \int_{x}^{1} \cdots \int_{x}^{1} P_{\nu}(x)(dx)^m \]

\[
Q_{\nu}^{m}(z) = \left( z^2 - 1 \right)^{-\frac{m}{2}} \frac{d^{m}}{dz^{m}} Q_{\nu}(z) \]

\[
Q_{\nu}^{-m}(z) = (-1)^{m} \left( z^2 - 1 \right)^{-\frac{m}{2}} \int_{z}^{\infty} \cdots \int_{z}^{\infty} Q_{\nu}(z)(dz)^m \]

Figure 8.12: Example for table representation model
Figure 8.13: The graph representation of the table in figure 8.12

must be satisfied before the rule is applied for rewriting a graph that represents a table.

**A set of graph rewriting rules**

Based on the fact that any table must have at least two rows and two columns, these rules are built to occasionally interpret the table layout structure (in case of having the smallest table) and more often to direct the recognition processing to a possible table interpretation. Each rule here consists of several possible cell combinations \( g_r \) that can replace a number of cells in \( g_l \in G \). The application of these rules are controlled using the application conditions. This would prevent the possibility of a collision of two or more table structure interpretations. This section illustrates these rules in more detail. Figures under this section show the production rules described by the graph illustrated in section 8.3.1 where \((\rightarrow, | \) and \(\emptyset\) denote derivation, selection and null element, respectively).

To make the allocating of the node combinations on the graph \( g_l \) easier, the initial graph described in section 5.1 is recalled. As a consequence, it is found that the following
possible interpretations from rules one, two and three are enough to cover all different node combinations. Experiments, using these three rules, are discussed in next chapter.

**Production Rule One:**

A node combination of two nodes that have horizontal overlap with two nodes is located in the graph $G$. In this case, there are fifteen possible interpretations $g_r$ that can replace this combination of cells in $g_l \in G$. The following figure 8.14 illustrates this rule in detail.
Definition 30 (Rule One). Let $g_i \in G$ be a combination containing $n_1, n_2$ as two nodes vertically overlaps with each others and $n_3, n_4$ as two nodes vertically overlaps with each others such that $n_1$ horizontally overlaps with $n_3$ and $n_2$ horizontally overlaps with $n_4$. We generate the following possible interpretations in $g_i$ based on this combination in $g_i$:

1. The same combination $g_i$ remains but clustered in one column $col$.

2. The combination splits into two columns $col$ where the first $col$ contains a node $n_1$ horizontally overlaps with $n_3$ and the second $col$ encompasses $n_2$ horizontally overlaps with $n_4$.

3. Similar to (2), the combination splits into two columns $col$. However, the first $col$ encompasses a node $n_1$ and the second $col$ contains a nodes $n_2$ horizontally overlaps with $n_4$ which vertically overlaps with $n_3$.

4. Likewise (3), the combination splits into two columns $col$. However, the first $col$ encompasses a node $n_1$ horizontally overlaps with $n_3$ which vertically overlaps $n_4$ and the second $col$ contains a node $n_2$.

5. Again, the combination splits into two columns $col$. However, this time, the first $col$ encompasses a node $n_3$ and the second $col$ contains a node $n_1$ vertically overlaps with $n_2$ which horizontally overlaps with $n_4$.

6. In this possible interpretation, the combination splits into two columns $col$. The first $col$ encompasses a node $n_1$ horizontally overlaps with $n_3$ and vertically overlaps with $n_2$ and the second $col$ contains a node $n_4$.

7. This time, the combination splits into two columns $col$. The first $col$ encompasses a node $n_1$ vertically overlaps with $n_2$ and the second $col$ contains a node $n_3$ vertically overlaps with $n_4$.

8. Similar to (7), the combination splits into two columns $col$. The first $col$ encompasses a node $n_3$ vertically overlaps with $n_4$ and the second $col$ contains a node $n_1$ vertically overlaps with $n_2$. 

121
9. Three columns \textit{col} are constructed where the first, second and third columns contain \(n_1, n_3\) vertically overlaps with \(n_4\) and \(n_2\) respectively.

10. Likewise (9), three columns \textit{col} are constructed. However, the first, second and third columns contain \(n_3, n_1\) vertically overlaps with \(n_2\) and \(n_4\) respectively.

11. In this possible interpretation, four columns \textit{col} are constructed where the first, second, third and four columns contain \(n_3, n_1, n_2\) and \(n_4\) respectively.

12. Similar to (11), four columns \textit{col} are constructed. However, the first, second, third and four columns contain \(n_1, n_2, n_3\) and \(n_4\) respectively.

13. Likewise (12), four columns \textit{col} are constructed. However, the first, second, third and four columns contain \(n_3, n_4, n_1\) and \(n_2\) respectively.

14. Three columns \textit{col} are constructed where the first, second and third columns contain \(n_1\) horizontally overlaps with \(n_3, n_4\) and \(n_2\) respectively.

15. Likewise (14), three columns \textit{col} are constructed. However, the first, second and third columns contain \(n_1\) horizontally overlaps with \(n_3, n_2\) and \(n_4\) respectively.

**Associated embedded notation:** Each different cell combination in \(g_l\) can be replaced by more that one possible interpretation in \(g_r\). This involves some edge conversions. Definition 31 formally expresses the edge conversions that are needed for each replacing of the combination in \(g_l\) with each possible interpretation \(g_r\) in definition 30.

**Definition 31** (Notation One). Let \(g_l \in G\) be the combination mentioned in Def. 30. We call the following notations as the corresponding edge conversions needed to replace \(g_l\) with one of the possible interpretations \(g_r\) that also defined in Def. 30 respectively.

1. \((\text{node}_1, \uparrow, \text{node}_3, \text{node}_5, \downarrow, \text{node}_7)\)

2. \((\text{node}_1, \uparrow, \text{node}_3, \text{node}_5, \uparrow, \text{node}_7)\)
Production Rule Two:

A combination of two nodes which are node₁, node₂ where node₁ has horizontal overlapping with node₂ is located in G. In this case, there are three possible cell interpretations g_r that can replace this combination of cells in g_l ∈ G. The following figure 8.15 illustrates this rule in detail.

Definition 32 (Rule Two). Let g_l ∈ G be a combination containing n₁, n₂ as nodes such that n₁ horizontally overlaps with n₂. We generate the following possible interpretations in g_r based on this combination in g_l:
1. The same combination $g_l$ remains but clustered in one column $\text{col}$.

2. The combination splits into two columns $\text{col}$ where the first $\text{col}$ contains node $n_2$ and the second $\text{col}$ encompasses $n_1$.

3. In this possible interpretations, the combination splits into two columns $\text{col}$. The first and second columns $\text{col}$ encompass $n_1$ and $n_2$ respectively.

**Associated embedded notation:** Each different cell combination in $g_l$ can be replaced by more that one possible interpretation in $g_r$. This involves some edge conversions. Definition 33 formally expresses the edge conversions that are needed for each replacing of the combination in $g_l$ with each possible interpretation $g_r$ in definition 32.

**Definition 33** (Notation Two). Let $g_l \in G$ be the combination mentioned in Def. 32. We call the following notations as the corresponding edge conversions needed to replace $g_l$ with one of the possible interpretations $g_r$ that also defined in Def. 32 respectively.

1. $(\text{node}_1 \downarrow, \text{node}_2, \text{node}_3, \uparrow, \text{node}_4)$

2. $(\text{node}_1 \downarrow, \text{node}_2, \text{node}_3, \emptyset, \text{node}_4)$

3. $(\text{node}_1 \downarrow, \text{node}_2, \text{node}_3, \emptyset, \text{node}_4)$
Production Rule Three:

A combination of two nodes which are $node_1, node_2$ where $node_1$ has horizontal overlapping with $node_2$ is located in $G$. In this case, there are three possible cell interpretations $g_r$ that can replace this combination of cells in $g_l \in G$. The following figure 8.16 illustrates this rule in detail.

**Definition 34** (Rule Three). Let $g_l \in G$ be a combination containing $n_1, n_2$ as nodes such that $n_1$ vertically overlaps with $n_2$. We generate the following possible interpretations in $g_r$ based on this combination in $g_l$:

1. The same combination $g_l$ remains but clustered in one column $col$.

2. Similar to (1), the same combination $g_r$ is remained but this time, the nodes are split into two column $col$ where first $col$ contains $N' \in N$ and the second $col$ contains $N'' = N \setminus N'$.

### 8.4 Table structure analysis

Using the graph rewriting rules, an analysis of the table structure is performed. Since the information of the table structure is fully described in the graph that can be re-written by applying the rewriting rules, one can utilise a general graph parser for table structure analysis. As these rewriting rules have a form which is equivalent to a context sensitive
grammar, it is not easy to parse the tables.

To overcome this problem, a constraint is associated with each rule and performed prior applying it, to help with recognising the table structure. As observed and also mentioned in [Alka 13a], the tables in our dataset have more than one possible interpretation of their structures. Therefore, one has to use a suitable constraint on the rewriting rules each time one attempts to produce a particular desirable output. Next, an example of constraints that are imposed on the rewriting rules is shown to clarify how one can select from the proposed rewriting rules, one or more possible interpretation $g_r$ for node combination $g_t \in G$, to use them in obtaining a specific possible table interpretation. Parsing a table, which is selected from the table dataset that is composed from [Grad 07], using the rewriting rules presented in section 8.3.2 that pass specific constraints, is illustrated in the next example.

8.4.1 Apply these rules (example)

The goal of using these rules is to rewrite the initial graph that is introduced in section 5.1, so that the columns that have not been correctly split (see figure 8.17) using the technique in chapter 5 are detected and cleverly split.

The rewriting process starts from the top-left nodes and from top to bottom (See figure 8.18). The priority of applying these rules will be for rule one, rule two and rule three respectively.

Before starting the rewriting process on the table which presents in figure 8.17, some constraints are applied to restrict the usage of the rules. Here are the used rules and their constraints that are needed to rewriting this table:

1. Each $(node_1,node_2)$ and $(node_3,node_4)$ which are in same lines $l_1$ and $l_2$ respectively
An example of two columns that have not been correctly split. Green-bordered cells should be in one column and black-bordered cells should be in other column.

Figure 8.17: An example of columns that have not been correctly split and vertically overlapped as well as node\(_1\) is horizontally overlapped with node\(_3\) and node\(_2\) is horizontally overlapped with node\(_4\) must be split into three different columns such that first, second and third columns contain node\(_1\) is horizontally overlapped node\(_3\), node\(_2\) and node\(_4\) respectively. Rule one presents this combination \(g_l\) and its possible interpretation \(g_r\) labelled (15) is applied on this combination to rewriting a sub of graph \(G\), if (constraint):

\[
(b(node_2) - t(node_2)) > (b(node_1) - t(node_4)) * e
\]

where \(e\) is a fixed value. In my experiment, \(e = 1.5\) (which is determined empirically).

Otherwise:

possible interpretation \(g_r\) labelled (2) in rule one is applied.

2. Each node\(_1\) and node\(_2\) which are in different lines \(l_1\) and \(l_2\) respectively as well as horizontally overlapped must be split into two different columns. Rule two presents this combination \(g_l\) and its possible interpretation \(g_r\) labelled (3) is applied on this combination to rewrite a sub of graph \(G\). if (constraint):

\[
(b(node_1) - t(node_1)) > (b(node_2) - t(node_2)) * e
\]

where \(e\) is a fixed value. In my
Figure 8.18: How the rewriting Process starts

experiment, \( e = 1.5 \) (which is determined empirically).

Otherwise:

possible interpretation \( g_r \) labelled (1) in rule two is applied.

Now, the rewriting process starts by rewriting the two nodes in each first and second rows (See figure 8.19 which show the initial graph of the table in figure 8.17).

Since, \( (b(\text{node}_2) - t(\text{node}_2)) > (b(\text{node}_4) - t(\text{node}_4)) \times 1.5 \) then, as mentioned above, the possible interpretation \( g_r \) labelled (15) in Rule one is applied on these nodes. This rewriting step causes \( \text{node}_4 \) replaces with a virtual node and moves to a new column (see figure 8.20). It is worth to mention that other cells in this column are represented as virtual nodes.

This process continues by rewriting the nodes illustrate in figure 8.21. In this case, \( (b(\text{node}_2) - t(\text{node}_2)) \not> (b(\text{node}_4) - t(\text{node}_4)) \times 1.5 \). Therefore, the possible interpretation \( g_r \) labelled (2) in rule one is applied in these nodes.
It can be noticed that the result of this step, which illustrates in figure 8.22, has almost the same graph structure as in figure 8.20. This is because nodes in this rewriting step stayed in their original positions.

After visiting and rewriting all nodes in this graph, the graph presents in figure 8.23 is obtained. As it can be seen, the rewriting process succeed in correctly splitting all columns and eventually find the maximum cells that can be found in this table. Now, to find the set of possible interpretations of this table, the technique in section 6.2 is recall.
8.5 Summary

The framework represented in this chapter was built on the observation of tabular forms that are obtained from [Grad 07] which contain complex structures with lots of mis-aligned cells that make them too hard to be structurally interpreted. The abstract components of this framework can be used as a basis for a wide range of other applications to document recognition. To achieve this, we first give formal definitions to all possible relationships that can be found between table cells. Then, a graph model is described for representing
After the final rewriting took place, table layout structure. Also, formal definitions for several rewriting rules are illustrated. Finally, an example of how the selected possible interpretations of the rewriting rules can be applied on a graph of table and produce an interpretation of this table structure.

In the next chapter, a discussion and experimental evaluation of this framework is given. First, constraints that can be used to select, from the rewriting rules, the suitable possible interpretations \( g_r \) to rewrite node combination \( g_l \in G \) are described. Then, experiments on two different table datasets are discussed.
CHAPTER 9

EVALUATION OF THE SECOND TABULAR RECOGNITION METHOD

In addition to the example that is given in section 8.4.1, which shows a possible case of applying the framework in chapter 8 on tables, and for more experiments, we have run the implementation of this framework over 150 and 110 tables, that are taken from [Grad 07] and represent training and testing datasets respectively.

To make the allocating of the node combinations on the graph $g_l$ easier, the initial graph described in section 5.1 is recalled. As a consequence, it is found that the following possible interpretations from rules one and two are enough to cover all different node combinations. These interpretations are selectively applied on graphs, using some constraints as following:

1. Each $(node_1,node_2)$ and $(node_3,node_4)$ which are in same lines $l_1$ and $l_2$ respectively and vertically overlapped as well as $node_1$ is horizontally overlapped with $node_3$ and $node_2$ is horizontally overlapped with $node_4$ must be split into three different columns such that first, second and third columns contain $node_1$ is horizontally overlapped $node_3$, $node_2$ and $node_4$ respectively. Rule one presents this combination $g_l$ and its possible interpretation $g_r$ labelled (15) is applied on this combination to rewriting a sub of graph $G$, if(constraint):

$$(b(node_2) - t(node_2)) > (b(node_4) - t(node_4)) \times e$$

where $e$ is a fixed value. In my
experiment, $e = 1.5$ (which is determined empirically).

Otherwise:

possible interpretation $g_r$ labelled (2) in rule one is applied.

2. Each node\textsubscript{1} and node\textsubscript{2} which are in different lines $l_1$ and $l_2$ respectively as well as horizontally overlapped must be split into two different columns. Rule two presents this combination $g_l$ and its possible interpretation $g_r$ labelled (3) is applied on this combination to rewrite a sub of graph $G$. if(constraint):

$$(b(\text{node}_1) - t(\text{node}_1)) > (b(\text{node}_2) - t(\text{node}_2)) \ast e$$

where $e$ is a fixed value. In my experiment, $e = 1.5$ (which is determined empirically).

Otherwise:

possible interpretation $g_r$ labelled (1) in rule two is applied.

Note: the goal of this experiment is to find the misaligned columns and split them from other columns. This would itself form one possible interpretation of table structure.

These constraints that were used to select these possible interpretations are inferred and constructed based on observing common features of the table structure that we have in the 150 training table dataset.

Table 9.1 shows the results of running my technique over 150 tables in a concise manner. The table contains in its first column, three different numbers of tables that have same number of possible interpretations that present in second column. The rest of columns in this table contain numbers of tables that have all their columns correctly extracted, numbers of tables that their columns were partially extracted (greater than or equal 75\%) and numbers of tables that the technique was unable to correctly extract their columns (less than 75\%).
<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table Interpretations</td>
<td>Total</td>
</tr>
<tr>
<td>14</td>
<td>4</td>
<td>56</td>
</tr>
<tr>
<td>132</td>
<td>8</td>
<td>1056</td>
</tr>
<tr>
<td>4</td>
<td>16</td>
<td>64</td>
</tr>
</tbody>
</table>

Table 9.1: Results of applying the proposed table interpretation technique on 150 tables

Although, the table 9.1 shows very promising results. One should still do more experiments on another dataset (110 testing dataset) to ensure the technique performance consistency. Using the same rules and constraints that present in the beginning of this chapter, the framework is run over 110 tables that are taken, as 150 dataset, from [Grad 07]. Table 9.2 shows the result of this experiment. As can be inferred, the technique not only keeps its performance in the border line but it was improved.

<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table Interpretations</td>
<td>Total</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>101</td>
<td>8</td>
<td>808</td>
</tr>
<tr>
<td>8</td>
<td>16</td>
<td>128</td>
</tr>
</tbody>
</table>

Table 9.2: Results of applying the proposed table interpretation technique on 110 tables

### 9.1 More experiments

For testing the robustness of my technique, it is essential to test using tables from different domains and with various structures. Therefore, the implementation of our method was run over a dataset which was used for a competition at ICDAR 2013 conference. The dataset is available online which is freely downloadable at http://www.tamirhassan.com/competition.html, contains 67 excerpts as individual PDF files, with a total of 150 tables. It is worth to mention that this dataset was used in the competition to compare the ability of different techniques to correctly find tables location and table cell borders. Therefore, it
is not possible to compare the results of this competition with the result of this technique that is for recomposing table.

9.1.1 No need of constraints

The following points state the selected possible interpretations $g_r$ of combination of nodes $g_t$, appear in the rewriting rules, which are obtained by observing the tables structure of this dataset. Since, there is no misaligned cells within this dataset tables, no constraints are imposed on applying these possible interpretations.

1. **Each** ($node_1,node_2$) and ($node_3,node_4$) **which are in same lines** $l_1$ and $l_2$ **respectively and vertically overlapped as well as** $node_1$ **is horizontally overlapped with** $node_3$ **and** $node_2$ **is horizontally overlapped with** $node_4$ **must be split into two different columns such that first columns contains** $node_1$ **that is horizontally overlapped with** $node_3$ **and second column contains** $node_2$ **that is horizontally overlapped with** $node_4$. **Rule one presents this combination** $g_t$ **and its possible interpretation** $g_r$ **labelled (2) is applied on this combination to rewriting a sub of graph $G$,

2. **Each** $node_1$ **and** $node_2$ **which are in different lines** $l_1$ and $l_2$ **respectively as well as horizontally overlapped remain the same and form one columns.** **Rule two presents this combination** $g_t$ **and its possible interpretation** $g_r$ **labelled (1) is applied on this combination to rewrite a sub of graph $G$.

9.1.2 Experimental results

After running the described technique, using these possible interpretations, over the target dataset, the following results are concisely expressed in table 9.3.

After observing the tables that their columns either partially extracted or not correctly extracted, we found that the most common error in these tables occurs when the spanning cell does not horizontally overlap all cells that it should do, due to the fact that
Table 9.3: Results of applying the proposed table interpretation technique on 150 ICDAR tables

<table>
<thead>
<tr>
<th>No. of Tables</th>
<th>Ground Truth Table Dataset</th>
<th>Output Table Dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>All Possible Table</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Interpretations</td>
<td>Total</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>24</td>
<td>4</td>
<td>96</td>
</tr>
<tr>
<td>33</td>
<td>8</td>
<td>264</td>
</tr>
<tr>
<td>19</td>
<td>16</td>
<td>304</td>
</tr>
<tr>
<td>15</td>
<td>32</td>
<td>480</td>
</tr>
<tr>
<td>8</td>
<td>64</td>
<td>512</td>
</tr>
<tr>
<td>11</td>
<td>128</td>
<td>1408</td>
</tr>
<tr>
<td>6</td>
<td>256</td>
<td>1536</td>
</tr>
<tr>
<td>10</td>
<td>512</td>
<td>5210</td>
</tr>
<tr>
<td>6</td>
<td>1024</td>
<td>6144</td>
</tr>
<tr>
<td>2</td>
<td>4096</td>
<td>8192</td>
</tr>
<tr>
<td>1</td>
<td>8192</td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td></td>
<td>No. of TICE</td>
</tr>
<tr>
<td></td>
<td></td>
<td>22</td>
</tr>
<tr>
<td></td>
<td></td>
<td>92</td>
</tr>
<tr>
<td></td>
<td></td>
<td>232</td>
</tr>
<tr>
<td></td>
<td></td>
<td>304</td>
</tr>
<tr>
<td></td>
<td></td>
<td>448</td>
</tr>
<tr>
<td></td>
<td></td>
<td>448</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1280</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1024</td>
</tr>
<tr>
<td></td>
<td></td>
<td>512</td>
</tr>
<tr>
<td></td>
<td></td>
<td>6144</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td></td>
<td>8192</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

the cells were extracted based on their contents. Manual intervention, as it is described in chapter 5, would be one of the solutions to this problem. Another solution is to extend the cell segmentation technique in chapter 4 so that, it extracts the real borders of cells.

A comparison of the performance of my technique on this dataset with other techniques performance on the same dataset is not possible due to the absence of any available published results. In addition to this, current table recognition methods are informally presented [Zani 04]. Detail of how these techniques work is usually not fully described. This makes it difficult to compare techniques performance.

**Comparison to the first method**

By analysing the experimental results in this chapter and the one in chapter 7, one can conclude that the second method yield better results. This is because the method uses an intelligent approach to extract columns, unlike the first method, where a heuristic technique is used. The figures in 9.1 and 9.2 show the results of applying the two methods on a table. It can be seen that the table in figure 9.1 contains three columns where the
second column encompasses two columns that should have been split up from each other but, due to the strong mis-aligned cells, the heuristic approach in the first method fails to distinguish them. Figure 9.2 shows the final result of applying the second method, that has a clever technique to extract columns, on the same table. The output table contains four columns which correspond to the actual number of the table columns.


9.2 Summary

In this chapter, some of possible interpretations $g_r$ in rule one, two and three are selected, using specific constraints. Then, evaluating of this framework is accomplished by applying the implementation of it on 150, 110 and 150 tables from different datasets. The experiments show promising results.
CHAPTER 10

CONCLUSIONS AND FUTURE WORK

10.1 Contributions

In this section, a summary of contributions, that are described in this thesis, is given. Four pieces of research work are listed to handle several existing problems that face the recognition of table structure which contains mathematical expressions.

10.1.1 Mathematical line segmentation

In chapter 3, we presented a line detection technique that is geared towards documents that contain a large number of complex mathematical expressions. Our approach can not only deal with detecting compound lines that consist of combinations of several independent lines separated by vertical whitespace, but we also have most recently added a method to detect and split mathematical lines that share vertically overlapping characters. The procedure exploits only simple spatial features in a histogrammatic approach, avoiding the use of many parameters that need to be fine tuned or relying on statistical data from large sample sets. Our experiments show that we nevertheless get a high rate of accuracy in detecting correct lines. The algorithm currently serves as a basis for our work on layout analysis of tabular mathematical expressions.
10.1.2 Cell segmentation

In chapter 4, the proposed cell segmentation technique is based on the idea of detecting cells first, rather than doing a spatial analysis of rows and columns. In the absence of a language model for mathematical expressions we again use spatial features analogous to our line finding algorithm. Experiments on 200 pages that are taken from [Grad 07] yield promising results.

10.1.3 First table interpretation method

The proposed framework introduced in chapters 5, 6 and 7, is able to produce several interpretations of a table. Unlike other table representation techniques, the proposed approach has the capability to deal with misaligned columns that sometimes appear in tabular mathematical components. This method uses a heuristic approach to extract table columns. Adding virtual nodes to the initial graph prevents a complex relationship between nodes and would contribute to decide the actual table’s rows. Using the described production rules allows the production of more than one possible valid interpretations of table structure. The experiments on 150 tables show promising results.

10.1.4 Second table interpretation method

Similar to the framework in chapter 5, the framework represented in chapter 8 was built on the observation of a wide range of tabular forms which occur in many documents from different domains. The abstract components of this framework can be used as a basis for a wide range of other applications to document recognition. The framework is also able to produce several interpretations of a table. This method uses an intelligent approach to infer table columns. The experiments on 110 and 157 tables from two datasets show better results compared to the results of the first table interpretation method.
10.2 Future Work

This section is about presenting some issues that one can look into as future work. Two pieces of work are discussed of which the former suggests a way to improve the technique in chapter 4 concerning cell segmentation and the latter suggests an automatic approach for selecting rewriting rules.

10.2.1 Improving cells segmentation technique

Although, the cell detection technique described in chapter 4 gives already reasonable results, there are several cases where this method is still not able to cover. These cases usually occur when the spaces between cells are either too wide or too tight. As a solution, we are thinking to, use beside the current cell separation condition of my technique, additional constraints in hoping that these constraints will deal with such cases. The following Figure 10.1 shows an example of table that illustrates a case where the current technique fails to correctly segment cells. Then, we will state the suitable constraint that solves this problem in this example.

As can be seen, some of the cells in figure 10.1 are wrongly split into two parts. To
sort this case out, we state a constraint says that any two adjacent cells in a row which
the first one contains a bracket “[” as first character in its content and where the last
character can be any character but a bracket “]” is clustered with the second cell that has
the missing bracket “]” of first cell but has no bracket “[” as first character in its content.

10.2.2 Automatic selecting of rewriting rules for different interpretations of table structure

In order to be able to run the implementation of my technique over some tables with-
out any parameters (For instance, the constraints used in section 10.3) that have to be
manually extracted each time one wants to recognise tables from particular domain, one
should find an approach to construct these parameters automatically. One could consider
choosing these parameters by means such as genetic algorithms, but this is beyond the
scope of this thesis.
Part IV

Appendixes
Appendix A: A proposal of alternative rules for graph rewriting system

The following six rules are introduced as an alternative, that one could also use (with the above three rules described in Chapter 8) to infer table structure (It is worth to mention that these rules are considered but did not pursue). However, this time, these rules are applied on a graph $G$ that builds based on the original positions of cells. As a consequence, allocating the $g_l$ of these rules on the graph $G$ would be a hard mission to accomplish. Next, the rules are formally expressed and then an example of applying some of these rules on a table is given.

Production Rule Four:

In this case, a node combination of spanning node that has horizontal overlap with more than one node is located in the graph $G$. There are eight possible interpretations $g_r$ that can replace this combination of cells in $g_l \in G$. The following figure 2 illustrates this rule in detail.

**Definition 35** (Spanning Node). Let $N = \{n_1, ... n_m\}$ be a set of nodes vertically overlapping each other where $m >= 2$. We say $n_v$ is a vertical spanning node, if $n_v$ vertically overlaps with $N$. Similarly, we say $n_h$ is a horizontal spanning node, if $n_h$ horizontally overlaps with $N$.

**Definition 36** (Production Rule Four). Let $g_l \in G$ be a combination containing a horizontal spanning node $n_h$ which horizontally overlaps with the set of nodes $N$. We generate the following possible interpretations in $g_r$ based on this combination in $g_l$:

1. The same combination $g_l$ remains but clustered in one column $col$.

2. The combination splits into two columns $col$ where the first $col$ contains a set of nodes $N' \in N$ and the second $col$ encompasses $n_h$ horizontally overlaps with a set
of nodes $N'' = N \setminus N'$

3. Similar to (2), the combination splits into two columns $col$. However, the first $col$ encompasses $n_h$ horizontally overlaps with a set of nodes $N' \in N$ and the second $col$ contains a set of nodes $N'' = N \setminus N'$

4. In this possible interpretation, three columns $col$ are constructed where the first, second and third columns contain $n_h$, $N'$ and $N''$ respectively.

5. Similar to (4), three columns $col$ are constructed where the first, second and third columns contain $N'$, $n_h$ and $N''$ respectively.

6. Likewise (4) and (5), three columns $col$ are constructed where the first, second and third columns contain $N'$, $N''$ and $n_h$ respectively.

7. The combination splits into two columns $col$. The first $col$ encompasses $N$ and the
second col contains \( n_h \)

8. In this possible interpretation, two columns col are constructed where the first and second columns contain \( n_h \) and \( N \) respectively.

**Associated embedded notation:** As can be seen in definition 36, each different cell combination in \( g_l \) can be replaced by more that one possible interpretation in \( g_r \). This involves some edge conversions. Definition 37 formally expresses the edge conversions that are needed for each replacing of the combination in \( g_l \) with each possible interpretation \( g_r \) in definition 36.

**Definition 37** (Notation Four). Let \( g_l \in G \) be the combination mentioned in Def. 36. We call the following notations as the corresponding edge conversions needed to replace \( g_l \) with the possible interpretations \( g_r \) that also defined in Def. 36 respectively.

1. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \downarrow, \text{node}_5)\)
2. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \downarrow, \text{node}_6)\)
3. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \downarrow, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)
4. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)
5. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)
6. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)
7. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)
8. \((\text{node}_1, \uparrow, \text{node}_2, \text{node}_4, \emptyset, \text{node}_5), (\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)

**Production Rule Five:**

Similarly to rule one, a node combination of spanning node that has horizontal overlap with more than one node is located in the graph \( G \). However, in this case, they are upside
down. There are eight possible interpretations $g_r$ that can replace this combination of cells in $g_l \in G$. The following figure 3 illustrates this rule in detail.

**Definition 38** (Rule Five). Let $g_l \in G$ be a combination containing $N$ nodes horizontally overlaps with a horizontal spanning node $n_h$. We generate the following possible interpretations in $g_r$ based on this combination in $g_l$:

1. The same combination $g_l$ remains but clustered in one column $col$.

2. The combination splits into two columns $col$ where the first $col$ contains a set of nodes $N' \in N$ and the second $col$ encompasses a set of nodes $N'' = N \setminus N'$ horizontally overlaps with $n_h$.

3. Similar to (2), the combination splits into two columns $col$. However, the first $col$ encompasses a set of nodes $N' \in N$ horizontally overlaps with $n_h$ and the second
col contains a set of nodes $N'' = N \setminus N'$

4. In this possible interpretation, three columns col are constructed where the first, second and third columns contain $n_h$, $N'$ and $N''$ respectively.

5. Similar to (4), three columns col are constructed where the first, second and third columns contain $N'$, $n_h$ and $N''$ respectively.

6. Likewise (4) and (5), three columns col are constructed where the first, second and third columns contain $N'$, $N''$ and $n_h$ respectively.

7. The combination splits into two columns col. The first col encompasses $N$ and the second col contains $n_h$

8. In this possible interpretation, two columns col are constructed where the first and second columns contain $n_h$ and $N$ respectively.

**Associated embedded notation:** Each different cell combination in $g_l$ can be replaced by more that one possible interpretation in $g_r$. This involves some edge conversions. Definition 39 formally expresses the edge conversions that are needed for each replacing of the combination in $g_l$ with each possible interpretation $g_r$ in definition 38.

**Definition 39** (Notation Five). Let $g_l \in G$ be the combination mentioned in Def. 38. We call the following notations as the corresponding edge conversions needed to replace $g_l$ with one of the possible interpretations $g_r$ that also defined in Def. 38 respectively.

1. $(\text{node}_1, \uparrow, \text{node}_3, \text{node}_4, \uparrow, \text{node}_6)$
2. $(\text{node}_1, \uparrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)$, $(\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \uparrow, \text{node}_6)$
3. $(\text{node}_1, \uparrow, \text{node}_3, \text{node}_4, \downarrow, \text{node}_6)$, $(\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \emptyset, \text{node}_6)$
4. $(\text{node}_1, \uparrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)$, $(\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \emptyset, \text{node}_6)$
5. (node₁, ↑, node₃, node₄, ∅, node₆), (node₂, ↑, node₃, node₅, ∅, node₆)
6. (node₁, ↑, node₃, node₄, ∅, node₆), (node₂, ↑, node₃, node₅, ∅, node₆)
7. (node₁, ↑, node₃, node₄, ∅, node₆), (node₂, ↑, node₃, node₅, ∅, node₆)
8. (node₁, ↑, node₃, node₄, ∅, node₆), (node₂, ↑, node₃, node₅, ∅, node₆)

Production Rule Six:

A combination of three nodes which are node₁, node₂, node₃ where node₁ has vertical overlapping with node₂ and in the same time node₂ has horizontal overlapping with node₃ is located in G. In this case, there are four possible cell interpretations gr that can replace this combination of cells in gl ∈ G. The following figure 4 illustrates this rule in detail.

Definition 40 (Rule Six). Let gl ∈ G be a combination containing n₁, n₂ and n₃ as nodes such that n₁, n₂ vertically overlap with each others and n₂ horizontally overlaps with n₃. We generate the following possible interpretations in gr based on this combination in gl:

1. The same combination gl remains but clustered in one column col.
2. The combination splits into two columns col where the first col contains node \( n_1 \) and the second col encompasses \( n_2 \) horizontally overlaps with \( n_3 \).

3. In this possible interpretations, the combination splits into three columns col. The first, second and third columns encompass \( n_1, n_2 \) and \( n_3 \) respectively.

4. Likewise (3), the combination splits into three columns col. However, the first, second and third columns col encompass \( n_1, n_3 \) and \( n_2 \) respectively.

**Associated embedded notation:** Each different cell combination in \( g_l \) can be replaced by more that one possible interpretation in \( g_r \). This involves some edge conversions. Definition 41 formally expresses the edge conversions that are needed for each replacing of the combination in \( g_l \) with each possible interpretation \( g_r \) in definition 40.

**Definition 41** (Notation Six). Let \( g_l \in G \) be the combination mentioned in Def. 40. We call the following notations as the corresponding edge conversions needed to replace \( g_l \) with one of the possible interpretations \( g_r \) that also defined in Def. 40 respectively.

\[
\begin{align*}
1. & \quad (\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \uparrow, \text{node}_6) \\
2. & \quad (\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \uparrow, \text{node}_6) \\
3. & \quad (\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \emptyset, \text{node}_6) \\
4. & \quad (\text{node}_2, \uparrow, \text{node}_3, \text{node}_5, \emptyset, \text{node}_6)
\end{align*}
\]

**Production Rule Seven:**

A combination of three nodes which are \( \text{node}_1, \text{node}_2, \text{node}_3 \) where \( \text{node}_1 \) has vertical overlapping with \( \text{node}_2 \) and also \( \text{node}_1 \) has horizontal overlapping with \( \text{node}_3 \) is located in \( G \).

In this case, there are four possible cell interpretations \( g_r \) that can replace this combination of cells in \( g_l \in G \). The following figure 5 illustrates this rule in detail.
Definition 42 (Rule Seven). Let $g_l \in G$ be a combination containing $n_1$, $n_2$ and $n_3$ as nodes such that $n_1$, $n_2$ vertically overlap with each other and $n_1$ horizontally overlaps with $n_3$. We generate the following possible interpretations in $g_r$ based on this combination in $g_l$:

1. The same combination $g_l$ remains but clustered in one column $col$.

2. The combination splits into two columns $col$ where the first $col$ contains node $n_1$ horizontally overlaps with $n_3$ and the second $col$ encompasses $n_2$.

3. In this possible interpretations, the combination splits into three columns $col$. The first, second and third columns $col$ encompass $n_3$, $n_1$ and $n_2$ respectively.

4. Likewise (3), the combination splits into three columns $col$. However, the first, second and third columns $col$ encompass $n_1$, $n_3$ and $n_2$ respectively.

Associated embedded notation: Each different cell combination in $g_l$ can be replaced by more than one possible interpretation in $g_r$. This involves some edge conversions. Definition 43 formally expresses the edge conversions that are needed for each replacing of the combination in $g_l$ with each possible interpretation $g_r$ in definition 42.

Definition 43 (Notation Seven). Let $g_l \in G$ be the combination mentioned in Def. 42. We call the following notations as the corresponding edge conversions needed to replace
Figure 6: Rule Eight

$g_l$ with one of the possible interpretations $g_r$ that also defined in Def. 42 respectively.

1. \((\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \uparrow, \text{node}_6)\)

2. \((\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \uparrow, \text{node}_6)\)

3. \((\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)

4. \((\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)\)

Production Rule Eight:

A combination of three nodes which are $\text{node}_1, \text{node}_2, \text{node}_3$ where $\text{node}_1$ has horizontal overlapping with $\text{node}_2$ and also $\text{node}_2$ has vertical overlapping with $\text{node}_3$ is located in $G$. In this case, there are four possible cell interpretations $g_r$ that can replace this combination of cells in $g_l \in G$. The following figure 6 illustrates this rule in detail.

**Definition 44** (Rule Eight). Let $g_l \in G$ be a combination containing $n_1$, $n_2$ and $n_3$ as nodes such that $n_1$ horizontally overlaps with $n_2$ and $n_2, n_3$ vertically overlap with each others. We generate the following possible interpretations in $g_r$ based on this combination in $g_l$: 
1. The same combination $g_l$ remains but clustered in one column $col$.

2. The combination splits into two columns $col$ where the first $col$ contains node $n_1$ horizontally overlaps with $n_2$ and the second $col$ encompasses $n_3$.

3. In this possible interpretations, the combination splits into three columns $col$. The first, second and third columns encompass $n_1$, $n_2$ and $n_3$ respectively.

4. Likewise (3), the combination splits into three columns $col$. However, the first, second and third columns encompass $n_2$, $n_1$ and $n_3$ respectively.

**Associated embedded notation:** Each different cell combination in $g_l$ can be replaced by more that one possible interpretation in $g_r$. This involves some edge conversions. Definition 45 formally expresses the edge conversions that are needed for each replacing of the combination in $g_l$ with each possible interpretation $g_r$ in definition 44.

**Definition 45** (Notation Eight). Let $g_l \in G$ be the combination mentioned in Def. 44. We call the following notations as the corresponding edge conversions needed to replace $g_l$ with one of the possible interpretations $g_r$ that also defined in Def. 44 respectively.

1. $(\node_1, \uparrow, \node_2, \node_4, \uparrow, \node_5)$

2. $(\node_1, \uparrow, \node_2, \node_4, \downarrow, \node_5)$

3. $(\node_1, \uparrow, \node_2, \node_4, \emptyset, \node_5)$

4. $(\node_1, \uparrow, \node_2, \node_4, \emptyset, \node_5)$

**Production Rule Nine:**

A combination of three nodes which are $\node_1, \node_2, \node_3$ where $\node_1$ has horizontal overlapping with $\node_3$ and also $\node_3$ has vertical overlapping with $\node_2$ is located in
In this case, there are four possible cell interpretations \( g_r \) that can replace this combination of cells in \( g_l \in G \). The following figure 7 illustrates this rule in detail.

**Definition 46** (Rule Nine). Let \( g_l \in G \) be a combination containing \( n_1, n_2 \) and \( n_3 \) as nodes such that \( n_1 \) horizontally overlaps with \( n_3 \) and \( n_2, n_3 \) vertically overlap with each others. We generate the following possible interpretations in \( g_r \) based on this combination in \( g_l \):

1. The same combination \( g_l \) remains but clustered in one column \( \text{col} \).
2. The combination splits into two columns \( \text{col} \) where the first \( \text{col} \) contains node \( n_2 \) and the second \( \text{col} \) encompasses \( n_1 \) horizontally overlap with \( n_3 \).
3. In this possible interpretations, the combination splits into three columns \( \text{col} \). The first, second and third columns \( \text{col} \) encompass \( n_2, n_3 \) and \( n_1 \) respectively.
4. Likewise (3), the combination splits into three columns \( \text{col} \). However, the first, second and third columns \( \text{col} \) encompass \( n_2, n_1 \) and \( n_3 \) respectively.

**Associated embedded notation**: Each different cell combination in \( g_l \) can be replaced by more that one possible interpretation in \( g_r \). This involves some edge conversions.
Definition 47 formally expresses the edge conversions that are needed for each replacing of the combination in $g_l$ with each possible interpretation $g_r$ in definition 46.

**Definition 47** (Notation Nine). Let $g_l \in G$ be the combination mentioned in Def. 46. We call the following notations as the corresponding edge conversions needed to replace $g_l$ with one of the possible interpretations $g_r$ that also defined in Def. 46 respectively.

1. $(\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \uparrow, \text{node}_6)$
2. $(\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \uparrow, \text{node}_6)$
3. $(\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)$
4. $(\text{node}_1, \downarrow, \text{node}_3, \text{node}_4, \emptyset, \text{node}_6)$

**0.3 An example of constraints associate with each rule**

To have the desirable possible interpretation of table structure that is shown in figure 8, we use two constraints, for selecting the right rewriting rule.

- **The first constraint states:**

  If $\text{node}_1$ and $\text{node}_2$ vertically overlap within a line, $\text{node}_2$ and $\text{node}_3$ horizontally overlap and also a specific relationship between $\text{node}_2$ and $\text{node}_3$ does exist

  then the three nodes must be placed in separate columns.

  Otherwise $\text{node}_1$ is split in one column and $\text{node}_2,\text{node}_3$ together in other column

Rule six presents this combination of nodes $g_l$ and its possible interpretation $g_r$ labelled (3) is applied on this combination to rewriting a sub of graph $G$

If $(b(\text{node}_2) - t(\text{node}_2)) > (b(\text{node}_3) - t(\text{node}_3)) \times e$ where $e$ is a fixed value. In my experiment, $e = 1.5$ (which is determined empirically).
Otherwise possible interpretation $g_r$ labelled (2) in rule six is applied.

- The second constraint states:

  If $node_1$ and $node_2$ are in different lines and are horizontally overlapping and also a specific relationship between $node_1$ and $node_2$ does exist

  then they must be placed in the same column.

  Otherwise split the nodes in different columns

Rule two presents this combination of nodes $g_l$ and its possible interpretation $g_r$ labelled (1) is applied on this combination to rewrite a sub of graph $G$

\[
(b(node_1) - t(node_1)) < (b(node_2) - t(node_2)) \times e
\]

where $e$ is a fixed value. In my experiment, $e = 1.5$ (which is determined empirically).

Otherwise possible interpretation $g_r$ labelled (3) in rule two is applied.

In the next steps, a description of how to apply the possible interpretations, that are selected above, on the graph in figure 9, that represents the table in figure 8, is given, to obtain a possible interpretation of this table structure. Figure 10 shows the output of this process. To visually show this output, we border every column’s cells in this table with a unique colour.

**Steps toward interpretation of table structure: example**

For our example, we order the application of these possible interpretations as follows:

1. possible interpretation labelled (3) in Rule six (thereafter called $PI_6$).
1. \[ \int_0^\infty \sin^2 k \cosh z \sinh u \, \sinh u \, \text{Se}_{2n+1}(u, q) \, du = \frac{\pi B_r^{(2n+1)}}{4 \omega_{2n+1} \left( \frac{1}{2}, q \right)} \text{Se}_{2n+1}(z, q) \quad |q| > 0 \]

2. \[ \int_0^\infty \cos^2 k \cosh z \sinh u \, \sinh u \, \text{Se}_{2n+1}(u, q) \, du = \frac{\pi B_r^{(2n+1)}}{4 \omega_{2n+1} \left( \frac{1}{2}, q \right)} \text{Cy}_{2n+1}(z, q) \quad |q| > 0 \]

3. \[ \int_0^\infty \sin^2 k \cosh z \sinh u \, \sinh u \, \text{Se}_{2n+2}(u, q) \, du = \frac{k \pi B_r^{(2n+2)}}{4 \omega_{2n+2} \left( \frac{1}{2}, q \right)} \text{Se}_{2n+2}(z, q) \quad |q| > 0 \]

4. \[ \int_0^\infty \cos^2 k \cosh z \sinh u \, \sinh u \, \text{Se}_{2n+2}(u, q) \, du = \frac{k \pi B_r^{(2n+2)}}{4 \omega_{2n+2} \left( \frac{1}{2}, q \right)} \text{Se}_{2n+2}(z, q) \quad |q| > 0 \]

The rewriting procedure begins by searching in the graph in figure 9, starting from the top-left node, for a combination of nodes that can be replaced by \( PI_6 \) or \( PI_2 \). Once a candidate combination of nodes is found, the replacement process is accomplished. In our case, the first two nodes in the first row on the graph as well as the first node in the second row are marked as a candidate combination that can be replaced by \( PI_6 \). Figure 11 shows the first rewriting of the graph in figure 9.

As it can be seen in the figure 11, the combination of three nodes were split to three different columns by applying \( PI_6 \). As a consequence, some edges are removed. The same process is repeatedly performed on the rest of graph nodes whenever the same combination of nodes as the one on \( g_l \) in figure 4 is found.

Similar to the way of applying \( PI_6 \), the possible interpretation \( PI_2 \) is implemented. Figure 11 shows the result of performing this \( PI_2 \) for the first time on two nodes horizontally overlapped. The two nodes remain horizontally overlapped and clustered in one column. The same process is repeatedly performed on the rest of graph nodes whenever the same combination of nodes as the one on \( g_l \) in figure 8.15 is found.
Figure 9: A graph represents table which is taken from [Grad 07]

Figure 10: Possible interpretation of a table which is taken from [Grad 07]

Figure 12 illustrates the final result of rewriting the graph in figure 9. It is clear that applying the selected possible interpretations $PI_6$ and $PI_2$ has successfully contributed to obtain one of the possible interpretations of the table structure that is shown in figure 8. This interpretation presents the maximum columns that can be found in the table. It is worth to say that the technique in chapter 5 failed to produce such interpretation.

Appendix B: Manual statistical results for the output of line segmentation approach
In this appendix, we include a table that provides concise information about the output of applying my line segmentation approach on 1000 pages selected from more than 60 scientific papers. The table shows the page number and how many principal lines in it have been incorrectly interpreted as non-principal. It also shows the non-principal lines which are wrongly interpreted as principal and the lines that are correctly classified as non-principal.

In addition, the table represents in the end of it the total lines of incorrect principal lines and both correct and incorrect non-principal lines. The total of all lines and the total of lines that are correctly classified are also given. Finally, the calculated accuracy rate of the output is showed.

The table is available online which is freely downloadable at http://www.cs.bham.ac.uk/~maa897/REPORT_of_1000pages.pdf
The table shows in this appendix provides us with statistical numbers that represent in detail the output of running my table recognition tool over 150 tables which are taken from [Grad 07]. The second column in this table gives a concise information about how many possible interpretations any table structure might have. This is obtained by using the ground truth of the 150 tables. The rest of columns in this table show a classification of the 150 tables that are created by my tool. This classification is based on how far an outputted possible table interpretation matches one of the possible interpretations of the same table from the ground truth tables.
In third column represents the number of possible table interpretations that match 100% where similarly the fourth column shows the number of possible table interpretations that their matching rate are between 75% and 95%. In the last column, the number of missed possible table interpretations are shown. In this case, the matching rate is 0%.

The table is available online which is freely downloadable at http://www.cs.bham.ac.uk/~maa897/Report_on_Table_Interpretation.pdf
Appendix D: Several ocaml codes for implementing our mathematical line segmentation technique

Listing 1: Ocaml code for composing line

```

let rec sortVert = function
| [] -> []
| pivot :: rest ->
  let is_less x = vertLess x pivot in
  let left, right = List.partition is_less rest in
  sortVert left @ [pivot] @ sortVert right

let rec getLines by glyphs lineGap curLine lines =
  match glyphs with
  | [] -> (curLine::lines)
  | h::t ->
    if ((h.Jsonfio.JsonfIO.y - lineGap) > by) then
      getLines (h.Jsonfio.JsonfIO.y + h.Jsonfio.JsonfIO.h) glyphs
      lineGap [] (curLine::lines)
    else
      getLines (max by (h.Jsonfio.JsonfIO.h + h.Jsonfio.JsonfIO.y)) t
      lineGap (h::curLine) lines
```

Listing 2: Ocaml code for detecting and splitting overlapping lines

```
let rec specificHorOverlapping ?(isTwoLines = false) line =
  match line with
  | h::t1::t when (CheckOverlapExistence h t1) ->
    specificHorOverlapping ~isTwoLines:true []
  | h::t1::t ->
    specificHorOverlapping ~isTwoLines (t1::t)
  | h::[] ->
    specificHorOverlapping ~isTwoLines []
```
let spliteLines ?(firList=[]) ?(secList=[]) line =
  firList = (firstLine((seperator line) line))
  secList = (secondLine((seperator line) line))
  (firList, secList)

let rec overlappingChecker lines newLines =
  match lines with
  | h::t when (specificHorOverlapping (sortGlyph h)) ->
    let fir, sec = (spliteLines (sortGlyph h)) in
    overlappingChecker t (fir::sec::newLines))
  | h::t -> (overlappingChecker t (h::newLines))
  | [] -> newLines
-> post_non_principal_detection ~ check:1 gap gap1
pre line_data (height_ratio (pre.glyphs)
"N": post_data)

| h::t1::t when (check=0)
-> post_non_principal_detection ~ check:1 gap gap1
pre line_data (height_ratio (pre.glyphs)
"P": post_data)

| h::t1::t when (compare h) & (compare t1) & ((previous1 pre h) > (next1 t1 h)) &
not(t1.right < h.left) &
not(t1.left > h.right) &
not(height_relationship t1 h)
-> post_non_principal_detection ~ check:1 gap gap1 h
(t1::t) (height_ratio (h.glyphs) "P": post_data)

| h::t1::t when (compare h) & (compare pre) & ((previous1 pre h) < (next1 t1 h)) &
not(pre.right < h.left) &
not(pre.left > h.right) &
not(height_relationship pre h)
-> post_non_principal_detection ~ check:1 gap gap1 h
(t1::t) (height_ratio (h.glyphs) "N": post_data)

| h::t1::t when (compare h) & (height_relationship pre h) &
((previous1 pre h) < (next1 t1 h)) &
not(pre.left > h.right)
-> post_non_principal_detection ~ check:1 gap gap1 h
(t1::t) (height_ratio (h.glyphs) "N": post_data)

| h::t1::t when (compare h) & (height_relationship t1 h) &
((previous1 pre h) > (next1 t1 h)) &
not(t1.left > h.right)
-> post_non_principal_detection ~ check:1 gap gap1 h
(t1::t) (height_ratio (h.glyphs) "N":::post_data)
| h::t1::t when (compare h) && (height_relationship pre h) &&
not(merge h.glyphs t1.glyphs ((List.hd (sortGlyph h.glyphs))).x) gap)
-> post_non_principal_detection ~check:1 gap gap1 h
(t1::t)(height_ratio (h.glyphs) "N":::post_data)
| h::t1::t when (compare h) && (height_relationship t1 h) &&
not(merge h.glyphs pre.glyphs ((List.hd (sortGlyph h.glyphs))).x) gap)
-> post_non_principal_detection ~check:1 gap gap1 h
(t1::t)(height_ratio (h.glyphs) "N":::post_data)
| h::t1::t -> post_non_principal_detection ~check:1 gap gap1 h
(t1::t) (height_ratio (h.glyphs) "P":::post_data)
| [] -> post_data

Listing 4: Ocaml code for merging non-principal with principal lines

1
2 let rec non_principal_merging ?(check=0) gap pre post_data
3 full_line_data=
4
5 match post_data with
6 | h::[] when (compare h) && (check=1)
7 -> non_principal_merging ~check gap h []
8 (height_ratio ((List.hd full_line_data).glyphs @h.glyphs) "P"::(List.tl full_line_data))
9 | h::[] -> non_principal_merging ~check gap h []
10 (h::full_line_data)
11 | h::t1::t when (List.length h.glyphs=1) && ((height_Glyph h.glyphs h.glyphs)<(height_Glyph h.glyphs t1.glyphs)/3) && ((previous1 pre h)>(next1 t1 h))
non_principal_merging ~ check_gap t1 t
(height_ratio (t1.glyphs@h.glyphs) t1.typ:::
full_line_data)

| h::t1::t when ((full_line_data)<>[]) && (compare h) &&
  (compare pre) && ((previous pre h) < (next t1 h))
  && not(height_relationship pre h)
-> non_principal_merging ~ check_gap h (t1::t)
(height_ratio ((List.hd full_line_data).glyphs
  @h.glyphs) "N"::(List.t1 full_line_data))

| h::t1::t when ((full_line_data)<>[]) && (compare h) &&
  (compare t1) && ((previous pre h) > (next t1 h))
  && not(height_relationship t1 h)
-> non_principal_merging ~ check_gap h (t1::t)
(height_ratio (t1.glyphs@h.glyphs) "N"::
full_line_data)

| h::t1::t when (compare pre) && (check=0)
-> non_principal_merging ~ check:1 gap h (t1::t)
(height_ratio (pre.glyphs@h.glyphs) "P"::
full_line_data)

| h::t1::t when (check=0)
-> non_principal_merging ~ check:1 gap pre
post_data (pre::full_line_data)

| h::t1::t when not(compare pre) && (compare h) &&
  (height_relationship pre h) && ((previous pre h)
  < (next t1 h)) && (merge h.glyphs pre.glyphs
  ((List.hd (sortGlyph h.glyphs)).x) gap)
-> non_principal_merging ~ check_gap h (t1::t)
(height_ratio ((List.hd full_line_data).glyphs
  @h.glyphs) "P"::(List.t1 full_line_data))

| h::t1::t when not(compare t1) && (compare h) &&
(height_relationship t1 h) && ((previous pre h) > (next t1 h)) && (merge h.glyphs t1.glyphs ((List.hd (sortGlyph h.glyphs)).x) gap) -> non_principal_merging \~\text{check gap} t1 t (height_ratio (t1.glyphs@h.glyphs) "P":: full_line_data) |

| h::t1::t -> non_principal_merging \~\text{check gap} h (t1::t) (h::full_line_data) |

| [] -> List.rev full_line_data |
Appendix E: Several ocaml codes for implementing our cell segmentation technique

Listing 5: Ocaml code for determining T value

1 let rec threshold ?(trash = 0) gap =
2
3 match gap with
4 |h::t -> if (t <> [])
5  then
6   if (((List.hd t) - h)<= 10)
7   then
8     threshold ~trash:h t
9   else
10    threshold ~trash:(List.hd t) []
11  else
12    threshold ~trash []
13 |[] -> trash

Listing 6: Ocaml code for detecting cells

1 let cell_info cell =
2 {glyphs=cell; left= (min_x cell); right= (max_x cell);
3 top= (min_y cell); bottom= (max_y cell); typ="P"}
4
5 let rec cells1 trash ~(cluster=[])(cells=[])(maxi cell =
6
7 match cell with
8 |h::t ->if (t <> [])
9  then
10   if (((List.hd t).x - maxi) < trash)
11  then
12    cells1 trash ~cluster:(h::cluster) ~cells (max max
((List.hd t).x + (List.hd t).w)) t

else

cells1 trash ~cluster:[] ~cells:((cell_info (h::cluster)) :: cells) (max maxi ((List.hd t).x + (List.hd t).w)) t

else

cells1 trash ~cluster ~cells:((cell_info (h::cluster)) :: cells) (max maxi (h.x + h.w)) t

|[] -> List.rev cells

let rec cells_in_lines ?(recursive = []) gap cells =

match cells with

|h::t -> cells_in_lines ~recursive:((cells1 (trashold1 gap)

(((List.hd(sortGlyph h.glyphs)).x)+((List.hd(sortGlyph

h.glyphs)).w)) (sortGlyph h.glyphs))::recursive) gap t

|[] -> recursive

169
Appendix F: Several ocaml codes for implementing our table representation technique using virtual nodes

Listing 7: Ocaml code for defining cell’s borders

```ocaml
let cell_info cell =
{ glyphs = cell; left = (min_x cell); right = (max_x cell);
  top = (min_y cell); bottom = (max_y cell); typ = "P"}
```

Listing 8: Ocaml code for extracting initial columns

```ocaml
let rec check ?(boo = true) chCell col =

match col with
| h::t when (chCell.left < h.right) -> check ~boo chCell t
| h::t -> check ~boo:false chCell []
| [] -> boo

let rec columnDetector cells out listOut =
match cells with
|h::t when (check h out) -> columnDetector t (h::out) listOut
|h::t -> columnDetector (h::t) [] (out::listOut)
| [] -> (out::listOut)
```

Listing 9: Ocaml code for adding virtual cells

```ocaml
let checkExistence line cell =
if(cell.top >= line.top)&&(cell.bottom <= line.bottom)
then true else false

let rec visualDetector ?(newColumn = []) mini column cellsInLines =
```
match cellsInLines with
  |h::t when (checkExistence h (List.hd column)) &&
  ((List.tl column)<>[]) -> visualDetector ~ newColumn:((List.hd column)
  :: newColumn) mini (List.tl column) t
  |h::t when (checkExistence h (List.hd column)) -> visualDetector ~ newColumn:((List.hd column)
  :: newColumn) mini column t
  |h::t -> visualDetector ~ newColumn:({glyphs=[];
  left=mini.left; right=mini.right; top=h.top; bottom=h.bottom; typ="P"})::newColumn)
  mini column t
  |[] -> List.rev newColumn

Listing 10: Ocaml code for creating a graph

let rec sortAndSplite ?(temp=[])?(newList=[]) 
mergedList = 

match mergedList with
  |h::t1::t when(int_of_string(String.sub (h.typ) 0 3)
  = int_of_string(String.sub (t1.typ) 0 3)) -> sortAndSplite ~temp:(h::temp) ~newList 
  (t1::t) 
  |h::t1::t -> sortAndSplite ~temp:[ ] ~newList:
  ((sortVert(h::temp))::newList) (t1::t) 
  |h::[] -> List.rev ((sortVert(h::temp))::newList) 
  |[] -> List.rev newList 

let array1 = Array.make_matrix (List.length(List.hd
(sortAndSplit mergedList) = (List.length (sortAndSplit mergedList)) {glyphs=[], left=0; right=0; top=0; bottom=0; typ=""}
Appendix G: Several ocaml codes for implementing multiple spatial relationships between table’s cells

Listing 11: Ocaml code for detecting cells that are internally and vertically overlapped

```ocaml
let para_Interior_Vertical_Overlap array2D fixCol row count
  = if (((array2D.(row).(count).top) > (array2D.(fixCol).
      (count).top) && (array2D.(row).(count).bottom) <
        (array2D.(fixCol).(count).bottom)) || ((array2D.(fixCol).
            (count).top) > (array2D.(row).(count).top) &&
            (array2D.(fixCol).(count).bottom) < (array2D.(row).
              (count).bottom)) && (array2D.(row).(count).glyphs <> [])
      && (array2D.(fixCol).(count).glyphs <> []))
    then true else false
```

Listing 12: Ocaml code for detecting cells that are totally and vertically overlapped

```ocaml
let para_Full_Match_Vertical_Overlap array2D fixCol row count
  = if (((array2D.(row).(count).top) = (array2D.(fixCol).
        (count).top) && (array2D.(row).(count).bottom) =
        (array2D.(fixCol).(count).bottom)) &&
        (array2D.(row).(count).glyphs <> []) &&
        (array2D.(fixCol).(count).glyphs <> []))
    then true else false
```

Listing 13: Ocaml code for detecting cells that are centrally and vertically overlapped

```ocaml
let para_Center_Vertical_Overlap array2D fixCol row count
  = if ((abs(((array2D.(row).(count).top) - (array2D.
            (fixCol).(count).top))) - (abs(((array2D.(fixCol).
              (count).bottom) - (array2D.(row).(count).bottom)))) <= 4)
    then true else false
```
Listing 15: Ocaml code for detecting cells that are sided and vertically overlapped

```ocaml
let para_Sided_Vertical_Overlap array2D fixCol row count =
  then true else false
```

Listing 16: Ocaml code for detecting cells that are internally and horizontally overlapped

```ocaml
let para_Interior_Horizontal_Overlap array2D fixRow row count =
  then true else false
```

Listing 17: Ocaml code for detecting cells that are fully and horizontally overlapped

```ocaml
let para_full_Match_Horizontal_Overlap array2D fixRow row count =
  then true else false
```
Listing 18: Ocaml code for detecting cells that are centrally and horizontally overlapped

```ocaml
let para_Center_Horizontal_Overlap array2D fixRow row count =
  then true else false
```

Listing 19: Ocaml code for detecting cells that are partially and horizontally overlapped

```ocaml
let para_Partial_Horizontal_Overlap array2D fixRow row count =
  then true else
    then true else false
```

176
Listing 20: Ocaml code for detecting cells that are sided and horizontally overlapped

```ocaml
let para_Sided_Horizontal_Overlap array2D fixRow row count =
&& (array2D.(count).(row).glyphs <> []) && (array2D.(count).(fixRow).glyphs <> [])
then true else false
```

---

Listing 20: Ocaml code for detecting cells that are sided and horizontally overlapped

```ocaml
let para_Sided_Horizontal_Overlap array2D fixRow row count =
&& (array2D.(count).(row).glyphs <> []) && (array2D.(count).(fixRow).glyphs <> [])
then true else false
```
LIST OF REFERENCES


