A Model for Detecting and Merging Vertically Spanned Table Cells in Plain Text Documents

Vanessa Long, Robert Dale and Steve Cassidy
Centre for Language Technology
Division of Information and Communication Sciences
Macquarie University
Sydney, Australia
{vanessa, rdale, cassidy}@ics.mq.edu.au

Abstract

A spanned cell in a table is a single, complete unit that physically occupies multiple columns and/or multiple rows. Spanned cells are common in tables, and they are a significant cause of error in the extraction of tables from free text documents. In this paper, we present a model for the detection and merging of vertically spanned cells for tables presented in plain text documents. Our model and algorithm are based purely on the layout features of the tables, and they require no semantic understanding of the documents. When tested on the 98 tables appearing in 40 randomly selected documents from a corpus of company announcements from the Australian Stock Exchange (ASX), our algorithm achieves an accuracy of 86.79% in detecting and merging vertically spanned cells.

1. Introduction

Tables are a convenient device for conveying information, and they are widely used in documents. For any kind of automated higher-level processing of real text, it is therefore essential to be able to extract the information embedded in tables correctly. To achieve this, we need a robust algorithm for the identification of table structures. In this paper, we are interested in the identification of tables in plain text documents, of the kind that might be generated by extracting text from a richer format such as PDF, or as might be found in email messages and other native plain text formats.

Documents in this form contain no explicit markup tags, and necessarily maintain a consistent vertical line height throughout. In such documents, a table is a superstructure imposed on a character-level grid. We define a *simple table cell* as a contiguous horizontal sequence of characters that together make up a minimal meaningful constituent of a table; these cells are arranged in the rows and Not all cells are simple: a *spanned cell* is a single, complete unit that physically occupies multiple columns and/or rows. *Horizontally* spanned cells occupy multiple contiguous columns; *vertically* spanned cells occupy multiple contiguous lines. Vertically spanned cells are commonly used in tables. We are working with a large corpus of company announcements in plain text form provided by the Australian Stock Exchange (ASX); these are documents whose publication is generally required for regulatory reasons, and many include tables containing financial information. In our random sample of 40 documents containing a total of 98 tables, 85 of these tables contained vertically spanned cells. Spanned cells have been identified in the literature as one of the major factors that contribute to errors in table recognition [1, 4]. Unlike the problem of detecting horizontally spanned cells, which has been addressed by several researchers [2, **?**, 3, 5], the problem of detecting vertically spanned cells has not received much attention. In this paper, we present a spanned-cell detection model and a cell-merging algorithm for tables presented in plain text documents.

2. Terminology

We will use the following terminology to describe tables as they appear in plain text documents:

1. A table consists of a contiguous series of lines, which are either *line-art lines*, *row-lines*, or *blank lines*.
2. A *line-art line* is a line whose purpose is to serve as a vertical delimiter. Line-art lines typically consist only of punctuation characters, with the hyphen and underscore being very common, and the plus sign being used to indicate column boundaries. For example, the last line in Table 2 is a line-art line.[1]

[1] Unless otherwise indicated, all example tables shown here are from

Proceedings of the 2005 Eight International Conference on Document Analysis and Recognition (ICDAR'05)
1520-5263/05 $20.00 © 2005 IEEE

123456789012345678901234567890123456789012345678901234567890123456789012

Sales revenue	60,492	61,224
Net increment in net mkt value of SGARAs	2,083	
Other revenue	6,032	3,395
Revenues from ordinary activities	68,607	64,619
Cost of sales	(49,065)	(48,125)
Distribution expenses	(1,378)	(2,143)
Marketing expenses	(663)	(835)
Operational expenses	(10,277)	(11,497)
Admin Expenses	(3,002)	(3,150)
Research & development expenses	(403)	(63)
Investing expenses	(222)	
Expenses from ordinary activities	(65,010)	(65,813)

Table 1. Implicit delimiters used between rows.

Tower Limited 2004-05-27 ASX-SIGNAL-G

HOMEX - Sydney

Tower today released its half year report for the period ended 31 March 2004. The company said the revenue from ordinary activities was up 71% from previous corresponding period (pcp) to NZ$512,799,000. The net profit was up 113% from pcp to NZ$20,454,000. No interim dividend was declared. Tower Reports Half Year Net Profit Up 113% to NZ$20.5 Mln

	Current Period ended 31 March 2004 (NZ$000)	Previous Period ended 31 March 2003 (NZ$000)
Total operating revenue	512,799	300,461
EBITDA	-	-
Pre-Tax Profit	50,883	(157,042)
Non-Recurring Items	-	-
Net Profit	20,454	(154,370)
Operating Cash Flow	(3,901)	(72,100)
Dividend	-	-
EPS (basic) (cents)	5.04	(93.07)
NTA (NZ$)	1.94	3.2

Table 2. Line-art lines used as explicit row delimiters.

		CURRENT PERIOD AUD000	PREVIOUS CORRESPONDING PERIOD AUD000
1.1	Revenues from ordinary activities	68,607	64,619
1.2	Expenses from ordinary activities (see items 1.24 + 12.5 + 12.6)	65,010	65,613
1.3	Borrowing costs	(1,518)	(1,441)
1.4	Share of net profit (loss) of associates and joint venture entities (see item 16.7)	1,484	30
1.5	Profit (loss) from ordinary activities before tax	595	(2,605)
1.6	Income tax on ordinary activities (see note 4)	15	126
1.7	Profit (loss) from ordinary activities after tax	580	(2,731)
1.8	Profit (loss) from extraordinary items after tax (see item 2.5)	-	-
1.9	Net profit (loss)	580	(2,731)
	outside equity interests	(18)	(57)
	attributable to members	598	(2,674)

Table 3. Blank lines used as explicit row delimiters.

Use a landscape setting to print this table.

Table D. Median Income of Households by State

(In 1998 dollars. For meaning of symbols, see text)

States	Three-year average 1996-1998 Median income	Three-year average 1996-1998 Standard error	Two-year moving averages 1997-1998 Median income	Two-year moving averages 1997-1998 Standard error	Two-year moving averages 1996-1997 Median income	Two-year moving averages 1996-1997 Standard error	Differences in 2-year moving averages 1997-98 less 1996-97 Difference	Differences in 2-year moving averages 1997-98 less 1996-97 Percent change
United States	37,779	137	38,233	167	37,227	148	1007 *	2.7 *
Alabama	33,394	1,003	34,351	1,210	31,958	1,211	2393 *	7.5 *
Alaska	51,421	1,236	49,717	1,418	51,786	1,354	-2069	-4.0 *
Arizona	34,402	909	35,170	1,057	33,058	1,085	2112 *	6.4 *
Arkansas	27,471	784	27,117	958	27,373	900	-256	-0.9
California	40,522	548	40,623	604	40,317	710	307	0.8
Colorado	44,349	1,075	45,253	1,282	43,224	1,389	2028 *	4.7 *
Connecticut	44,978	1,832	45,589	1,961	44,213	2,174	1376	3.1
Delaware	42,000	1,260	42,581	1,583	42,270	1,470	310	0.7
District of Columbia	32,999	911	32,895	953	32,783	1,099	112	0.3
Florida	33,234	442	33,935	561	32,396	462	1538 *	4.7 *
Georgia	36,553	891	37,950	869	35,497	1,117	2453 *	6.9 *
Hawaii	41,932	1,325	41,199	1,580	42,484	1,400	-1285	-3.0

Table 4. Heterogeneous row delimiters used. Source: U.S. Census Bureau at http://www.census.gov on 1st September 2004.

3. A *row-line* is a line that contains text that contributes to the content of the cells in a table. For example, the first line in Table 3 is a row-line. Note that row-lines may also contain punctuation marks that are used to indicate column boundaries.
4. A *blank line* is either an empty line or a line that contains only white space characters.
5. We define a *row delimiter* as any sequence of lines that separates two row-lines. In Table 2, each pair of adjacent rows is separated by a line-art line; and in Table 3, all the blank lines are row delimiters.
6. A *column delimiter* is a sequence of characters that separate table columns. For example, the '|' characters and their preceding white spaces in Table 2 are column delimiters.
7. A *cell segment*, marked by one or two column delimiters, is a sequence of text tokens embedded in the same row-line. For example, the first row-line in Table 2 contains two cell segments: 'Current Period ended 31' and 'Previous Period ended 31'.
8. A *cell* is the most basic, semantically complete unit in a table. A simple table cell consists of just one cell segment, but in the case of a vertically spanned cell, the cell will contain two or more cell segments from adjacent row-lines. For example, the last line in Table 1 is a row-line containing three cell segments, and each of these cell segments makes up a complete cell. However, the first row of Table 2 has two cells each containing two cell segments on adjacent row-lines.
9. A *row* is one or more row-lines that contain cells that are horizontally aligned. A row can be made up of one

our ASX Corpus.

Proceedings of the 2005 Eight International Conference on Document Analysis and Recognition (ICDAR'05)
1520-5263/05 $20.00 © 2005 IEEE

or more row-lines: for example, the last line in Table 1 is a row, and the first four lines in Table 3 also form one row.

10. A *row-block* is the longest block of adjacent row-lines. For example, the first four lines in Table 3 form one row-block, because the fifth line is not a row-line. However, all the lines in Table 1 are in one row-block, and the table contains one big row block.

3. Task Definition

Identifying the structure of even a simple plain text table can be a challenging task, primarily because of the ambiguous use of the space character as both a means of separating the contents of one column from another, and as a means of separating tokens within a column. The presence of vertically spanned cells, such as those occurring in Table 3, brings the additional difficult requirement of determining which cells contribute to spanned cells and then merging the content. The techniques required for detecting vertically spanned cells are more complex than those for detecting horizontally spanned cells. To detect horizontally spanned cells, we look for column alignments, which can be obtained by calculating the overlapped area using the starting and ending positions of each cell segment. To detect vertically spanned cells, we also have to decide whether two aligned cell segments should be merged. Our goal is to develop an algorithm that can identify and merge the relevant cell segments that make up vertically spanned cells. As an example, we want to be able to merge the first four row-lines in Table 3 to form a single table row that consists of two cells, the first containing the text 'CURRENT PERIOD AUD000' and the second containing the text 'PREVIOUS CORRESPONDING PERIOD AUD000'.

4. Approach

Our algorithm for detecting vertically spanned cells is based on the observation that authors of tables tend to use row delimiters to separate the adjacent row-lines that form vertically spanned cells from other row-lines. This leads to our first heuristic for detecting spanned cells:

Heuristic #1: The row-lines that contain segments of the same spanned cell should belong to the same row-block.

Of course, the presence of a row-block does not necessarily mean that we have spanned cells. This leads to our second heuristic for merging cell-lines:

Heuristic #2: Two cell-lines within a row-block should only be merged if they contain at least one pair of vertically aligned cell segments, and if there is no reason not to merge the cell segments.

4.1. Detecting Vertically Spanned Cells

A table can contain implicit row delimiters, explicit row delimiters or both. An implicit row delimiter is the line-feed character that identifies a single row-line as a table row. As noted earlier, an explicit row delimiter is one or more consecutive lines, including blank lines, that visually separate two table rows. Explicit row delimiters can appear in various forms: Table 2 uses line-art lines as row delimiters; Table 3 uses blank lines as row delimiters. Broadly speaking, tables can be categorised into two classes according to the types of row delimiters they contain; each class displays different merging characteristics. For ease of reference, we call these here *simple* and *complex* tables respectively.

Simple tables contain only implicit row delimiters. Each table row occupies exactly one line, and table cells are not spread across multiple row-lines. No merging should be performed for tables in this class.

Complex tables contain explicit row delimiters. The row delimiters can be all of the same type, as in Tables 2 and 3, or they can be of different types, as in Table 4. Merging might be required for tables in this class.

Essentially, this distinction embodies the hypothesis that tables which contain vertically spanned cells will always rely on the use of explicit row delimiters to make the structure of the table clear; if there are no explicit row delimiters, we assume the table has no vertically spanned cells.

4.2. Merging Vertically Spanned Cells

If a table contains explicit row delimiters, then we assume that it may contain vertically spanned cells; and if it does, these vertically spanned cells will always occur within row-blocks.

Given two cell segments belonging to the same row-block, the merging heuristic merges them if both of the following conditions are satisfied.

Merging Condition #1: The cell segments are vertically aligned, as determined by the column zoning algorithm described in Section 4.2.1.

Merging Condition #2: The cell segments are mergible, as determined by the criteria presented in Section 4.2.2.

4.2.1 Column Zoning

The purpose of the column zoning step is to determine the alignments of all the cell segments in a table. After the zoning step, we should know how many columns a table has,

Proceedings of the 2005 Eight International Conference on Document Analysis and Recognition (ICDAR'05)
1520-5263/05 $20.00 © 2005 IEEE

and where each column starts and ends. We carry out column zoning by using two data structures: a set of *position vectors* and a *column boundary map*.

A position vector, $p = \langle Start, End \rangle$, is used to record the horizontal start and end indices of the content of a cell segment. For example, the three cell segments in the first cell line in Table 1 are represented by three position vectors: $\langle 1, 13 \rangle$ for the '*Sales revenue*' segment, $\langle 44, 49 \rangle$ for the '*60,492*' segment, and $\langle 57, 62 \rangle$ for the '*61,224*' segment.

A column boundary map, denoted as *CBmap*, is a list of ordered position vectors that indicate the horizontal extents of the columns in a table, in left to right order, so that the i^{th} element in the map records the starting and ending indices of the i^{th} column of the table. For example, once completed, the *CBmap* for Table 1 should contain a list of three position vectors: $\{\langle 1, 40 \rangle, \langle 42, 49 \rangle, \langle 55, 62 \rangle\}$.

When we identify a cell-segment, we push its position vector into a sorted queue, denoted here by *sortedQ*. The *sortedQ* sorts the position vectors in ascending order of the lengths of the segments they represent. Once all the position vectors have been entered in the the queue, we use this information to compute the column boundary map for the table.

Let p_i be the i^{th} position vector in *CBmap*; then the algorithm is as follows. First, we initialize:

CBmap = empty;
$p_1 =$ *sortedQ*.$dequeue()$.

Then, while *sortedQ* is not empty, we do the following:

1. A position vector $P =$ *sortedQ*.dequeue()
2. Calculate the left alignment index, *LAI*, for P. *LAI* is the biggest *CBmap* index, i, that satisfies the condition $p_i[End] < P[Start]$. If such an index does not exist, then *LAI* is set to 0.
3. Calculate the right alignment index, *RAI*, for P. *RAI* is the smallest *CBmap* index, i, that satisfies the condition $P[End] < p_i[Start]$. If such an index does not exist, then *RAI* is set to $\|CBmap\| + 1$.
4. Update *CBmap* based on the difference between *RAI* and *LAI*. Let $d = RAI - LAI$. Our algorithm guarantees $d \geq 1$, and the update rules are the following.
   (a) If $d = 1$ then P marks a new column in the *CBmap*. Update *CBmap* by inserting P as the $(LAI + 1)^{th}$ element in *CBmap*.
   (b) If $d = 2$ then P is aligned with an existing column in *CBmap*. Update the $(LAI+1)^{th}$ element in *CBmap* as follows.
   $P_{LAI+1}[Start] = min(P_{LAI+1}[Start], P[Start])$
   $P_{LAI+1}[End] = max(P_{LAI+1}[End], P[End])$
   (c) If $d \geq 3$ then P is spanned across multiple columns. Update *CBmap* as follows.
   $P_{LAI+1}[Start] = min(P_{LAI+1}[Start], P[Start])$
   $P_{RAI-1}[End] = max(P_{RAI-1}[End], P[End])$

The end result is that we have determined the horizontal extents of each of the columns in the table, and we know which cell segments belong to which columns; i.e., we have the vertical alignment information we need for the next step.

4.2.2 Determining Mergibility

If two row-lines belong to the same row-block, and they contain cell segments that are vertically aligned, we then need to check whether the rows can be merged. The basic idea here is simple: two row-lines should not be merged if they both contain non-mergible cell segments. In our experiment, a non-mergible cell segment is a cell segment that meets any of the following criteria: it contains a numeric value; it contains a currency value; or it contains 'NA', 'N/A', '$-$', or '_'. This set of criteria could, of course, be extended and made more sophisticated.

Then, two adjacent row-lines r_i and r_{i+1} should be merged if all of the following conditions apply.

1. r_i and r_{i+1} are in the same row-block.
2. r_i and r_{i+1} have at least one pair of vertically aligned cell segments.
3. r_i and r_{i+1} do not both contain non-mergible cell segments.

After merging, the newly merged line should belong to the same row-block as r_i and r_{i+1}, and it will contain non-mergible cell segments if and only if r_i or r_{i+1} contains non-mergible cell segments. The merging process is repeatedly applied to every pair of adjacent row-lines within a row-block. The process stops only when any two adjacent row-lines contain no aligned cell segments, or they both contain non-mergible cell segments.

5. Evaluation

Our test data set contains a set 98 tables extracted from 40 documents randomly selected from the Australian Stock Exchange (ASX) corpus; these tables contain a mixture of spanned cells and non-spanned cells.

5.1. Accuracy Definition

For each document in the test data set, we manually compared the actual extracted result against the expected answer; three performance measures, recall rate (R), precision rate (P) and F-measure rate (F), are calculated as follows

Proceedings of the 2005 Eight International Conference on Document Analysis and Recognition (ICDAR'05)
1520-5263/05 $20.00 © 2005 IEEE

	Documents with VSC		Documents without VSC	All Documents		
Num of documents	25		15	40		
	VSC	NVSC	NVSC	VSC	NVSC	Overall
Num of cells in the original documents	565	2815	113	565	2928	3493
A	454	2424	110	454	2534	2988
B	111	391	3	111	394	505
C	33	274	2	33	276	309
Recall ($\frac{A}{A+B}$)	80.35%	86.11%	97.35%	80.35%	86.54%	85.54%
Precision ($\frac{A}{A+C}$)	93.22%	89.84%	98.21%	93.22%	90.18%	90.63%
F-Measure ($\frac{Recall+Precision}{2}$)	86.79%	87.98%	97.78%	86.79%	88.36%	88.09%

Figure 1. Table Extraction Results

Recall: $R = \frac{A}{A+B}$,
Precision: $P = \frac{A}{A+C}$,
F-measure: $F = \frac{R+P}{2}$,

where A is the number of cells that are correctly identified, B is the number of table cells missed by our extraction algorithm, and C is the number of non-table cells that are incorrectly identified as table cells by our algorithm.

5.2. Test Results

Our algorithm correctly identifies 454 of 565 vertically spanned cells (80.35%) while maintaining an accuracy of 88.36% for extracting cells that are not vertically spanned. The overall F-measure is 88.09%. The test results are summarised in Figure 1.[2]

6. Conclusions and Future Work

Vertically spanned cells are one of the major sources of errors in table extraction. In this paper we have presented a model to extract vertically spanned cells with an overall accuracy of 86.79%. The main sources of error come from the assumptions that our heuristics are based on:

1. Our work assumes that there is neither a blank line nor a line-art line between the row-lines that belong to a single spanned cell. While this assumption is valid most times, there are exceptions: in our test cases, there are table cells that are physically spanned across multiple lines and there are blank lines between the row-lines.

2. When deciding whether two adjacent cell-lines within the same row-block should be merged or not, we require that they do not both contain non-mergible cell segments, such as numeric data. This requirement is too strict, causing our algorithm to fail to merge row-lines when they should be merged.

Some relatively simple extensions to the heuristics presented here will increase the performance of our algorithm; it remains to be seen what the upper bound of performance without recourse to semantic information will be.

[2] VSC = Vertically-Spanned Cells; NVSC = Non-Vertically-Spanned Cells.

References

[1] S. Douglas, M. Hurst, and D. Quinn. Using natural language processing for identifying and interpreting tables in plain text. In *Proceedings of the Fourth Symposium on Document Analysis and Information Retrieval*, pages 535–546, Las Vegas, Nevada. University of Nevada, 1995.

[2] J. Hu, R. Kashi, D. Lopresti, and G. Wilfong. A system for understanding and reformulating tables. In *Fourth ICPR Workshop on Document Analysis Systems*, pages 361–372, Rio De Janeiro, Brazil, 2000.

[3] H. T. Ng, C. Y. Lim, and J. L. T. Koo. Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics*, pages 443–450, College Park, Maryland, USA, 1999.

[4] D. Pinto, A. McCallum, X. Lee, and W. B. Croft. Table extraction using conditional random fields. In *Proceedings of the 26th ACM SIGIR*, pages 235–242, Toronto, Canada, 2003.

[5] J. Ramel, M. Crucianu, N. Vincent, and C. Faure. Detection, extraction and representation of tables. In *The 7th International Conference on Document Analysis and Recognition*, pages 374–378, Edinburgh, Scotland, 2003.

Proceedings of the 2005 Eight International Conference on Document Analysis and Recognition (ICDAR'05)
1520-5263/05 $20.00 © 2005 IEEE