MACQUARIE UNIVERSITY

SYDNEY ~ AUSTRALIA

An Agent-Based Approach to Table Recognition and Interpretation

by

Vanessa Long

Department of Computing
Macquarie University
Sydney, Australia

A dissertation submitted in fulfilment of the requirements for the degree of
Doctor of Philosophy in Computer Science

May 23, 2010

Abstract

The goal of this research is to improve methods for extracting and interpreting tabular data embedded in printed documents. In pursuing this goal, this dissertation makes two main contributions. The first contribution is a blackboard based framework, which has the ability to incorporate a wide range of knowledge sources, for improving table analysis. Compared to traditional programming architectures, the blackboard framework makes it easy to add or to remove knowledge sources in a table processing task. This, in turn, results in several other benefits: it allows participation of partial knowledge in table analysis tasks; it encourages development of diverse and independent knowledge sources; it helps integrate programs from different programming paradigms; and it provides flexibility of experimenting and evaluating the effectiveness of individual knowledge sources.

The blackboard architecture was conceived in the 1980s for solving the speech recognition problem. In this research project, it was successfully applied to table analysis tasks because of a key innovation: the ability to find a suitable knowledge representation and communication protocol that support knowledge sharing. This innovation provides the foundation for introducing multiple sources of knowledge in table analysis.

The second contribution is a multi-level table extraction evaluation method that provides error analysis information. Traditionally, recall and precision have been the mainstream reporting measures for evaluating table structural analysis systems. Although these measures provide an overall measure for the strengths and weaknesses of the systems, they cannot give feedback on partially correct answers. The evaluation method proposed in this dissertation overcomes this deficiency by reporting error types at cell-level, row-level and table-level. With this fine grained information, the effectiveness of various structural analysis systems can be directly compared.

Through experiments, this dissertation shows not only the feasibility but also the benefits arising from the blackboard architecture and the multi-level evaluation method.

Acknowledgements

This dissertation could not have been produced without the immense support of many individuals and organisations. Firstly, I would like to thank my supervisors Steve Cassidy and Robert Dale for their encouragement and support in this project. Words are not enough to express my gratitude to Steve, who guided me through every step along the way - from project planning to proofreading the many pieces of writing that led to the final draft of this dissertation. It was Steve's patience, tolerance, and constructive criticism that kept me on track. His example of maintaining a high level of integrity and openness in research, and persistently pursuing critical thinking will continue to have positive influence on my professional career in the years to come.

This research project would not have existed without the vision of Robert, who saw the need for describing a table in words as we moved towards a speech-based web, and who initiated the Talking Tables project that subsequently developed into the current project. I wish to thank Robert for not only the financial support and the corpus data that he provided, but also for his enthusiasm and openness. He has been truly inspirational.

I would also like to express my appreciation to Macquarie University for providing financial support and office facilities during my PhD candidature, to CMCRC for providing top-up funding and test data, to the Centre of Language Technology research group, and to the IT support group at Macquarie University for their strong support.

I could never thank my unfailing dude, Arthur Watton, enough for everything he has done for me. The thank-you list for Arthur could easily occupy an entire chapter, but among many things, I would like to thank him for being such an excellent teacher and a faithful friend; for showing me how wonderful the research world is; for walking by my side throughout the entire journey, showing me new ways of seeing things; and for cheering me up with the best jokes in the world when I was over worked.

Finally, I want to thank my supportive husband, Jasper, for looking after our newborn babies and sharing the never ending house duties while I was completing this dissertation. Jasper, thank you for finding my research interesting, for sharing your ideas with me, for marking up expected answers for me, and for taking over some of the more difficult typesetting tasks - I could not have completed these alone.

Preface

The research presented in this dissertation is original work by the author except where otherwise indicated. Some parts of the dissertation include revised versions of published papers. The dissertation has not been submitted in whole or part for a degree at any other University or Institution.

The length of this dissertation excluding footnotes and appendices is approximately 50,000 words.

Vanessa Long

Contents

List of Tables

List of Figures

1

An Introduction to Table Analysis

1.1 Introduction

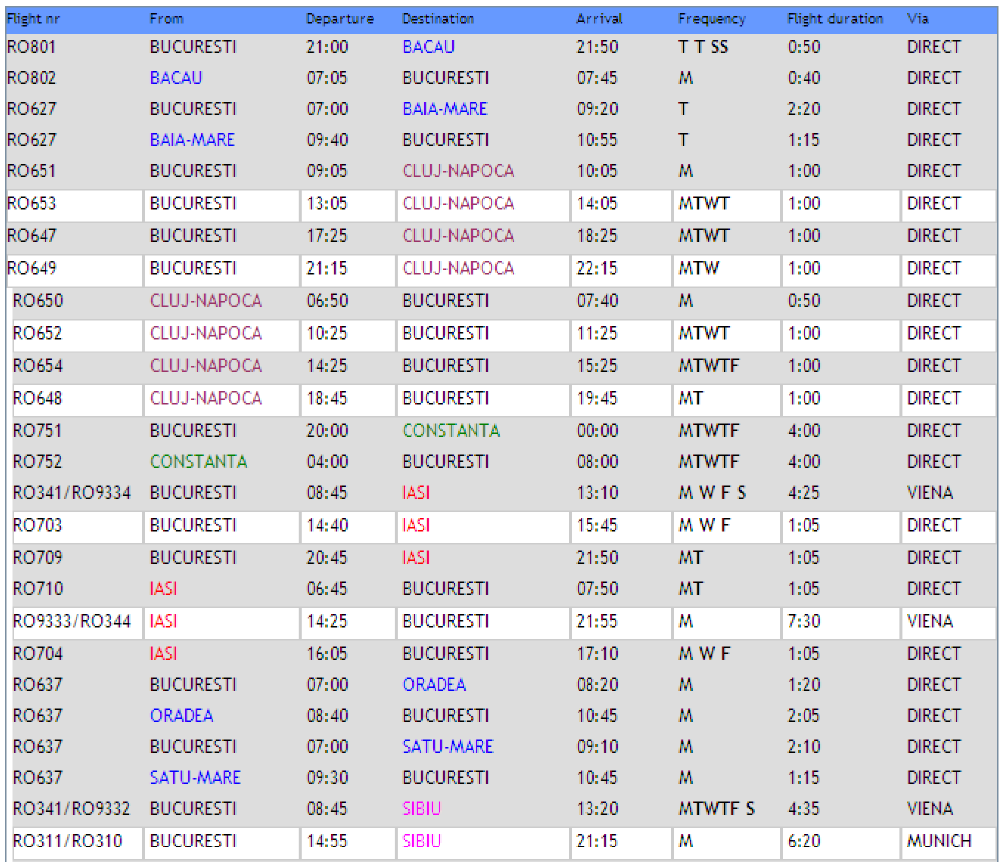
Tables have a long history in human civilization: the column-row structure that modern tables are based upon was found to be used for game-boards in the Near East during the Bronze Age (3500 BC - 1600 BC) (Whittaker, 2005). Developed originally in the days of handwritten documents, the use of tables dates back more than 4600 years when tables were used to record the number of workmen at a work site (Hooker (1990) cited in Hurst (2000)). Because of their labour intensive nature, handwritten tables are not in heavy use nowadays. Since the invention of computers, the way that printed documents are produced and stored has undergone a dramatic change: slow, inefficient typesetters have been replaced by intelligent document composers, and paper documents are often stored in electronic formats. This, together with the fact that there are more documents available than individuals can assimilate, creates new challenges in finding and absorbing information, including tabular information, in documents. Over the last 25 years, the field of table analysis has emerged as a research area to find solutions to meet the increasing demand for recognising and understanding tabular data stored in electronic documents.

Table layout analysis is one of the main research areas in this field, and it focuses on improving techniques for identifying table locations and recovering table structures from documents without modern markup. Over the years, various techniques and heuristics were reported to be effective for performing certain table analysis tasks on specific collections of documents.

However, narrow applicability is a common problem in these techniques and heuristics. For example, the benchmark table extraction program discussed in Chapter 5 was designed to extract tables from the Wall Street Journal, and it was reported to have about 70% accuracy in the literature. The same program delivered only 7% of accuracy when it was tested using the Australian Stock Exchange (ASX) corpus. This was because the algorithm was highly tailored for a small domain: a common problem shared by many existing programs. The work in this dissertation tries to widen the applicabilities of existing programs by connecting multiple programs together and letting them produce a joint result.

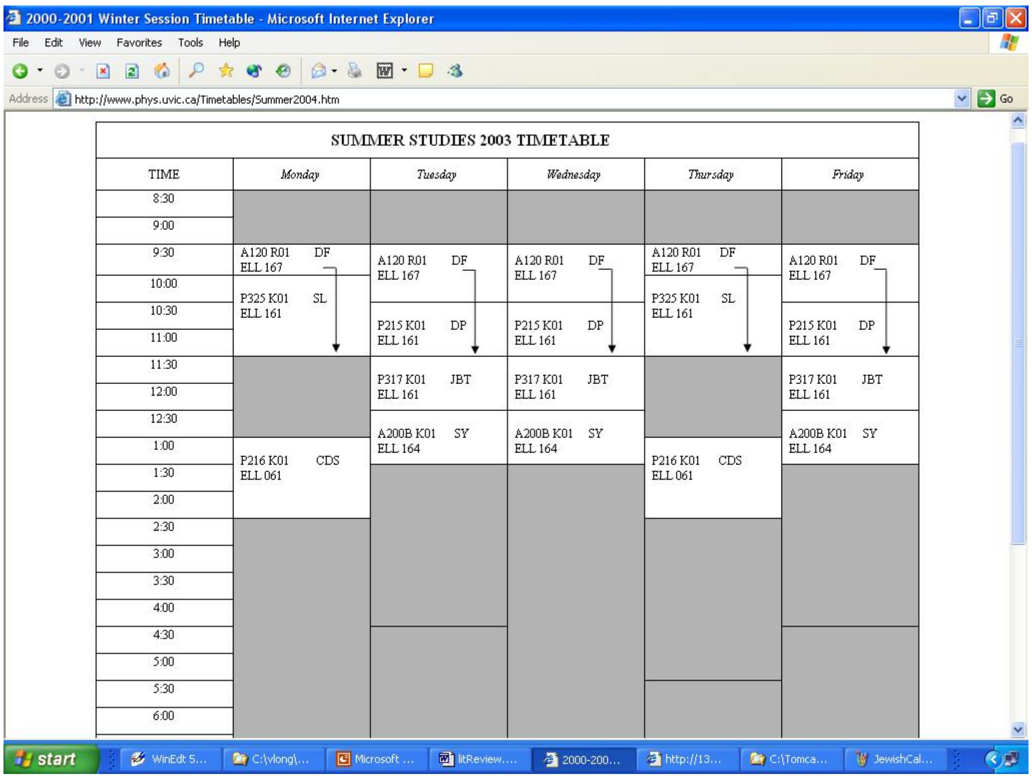
1.2 The Diversity of Tables

Most people would have some knowledge of tables as they are frequently used to provide information in our daily life. For example, the timetables in Figures 1.1 and 1.2 have been designed to inform users about scheduled activities in compact form; and finance tables are published every day in newspapers. In science, tables are often used to contrast information, to index information for fast searching, and to encode special relationships among data. For example, the truth table (Figure 1.3), which is designed to show computational relationships; the log table (Figure 1.4), which is to store pre-calculated results for fast look up purposes; and the currency table in Figure 1.5, which shows the relationships between currency pairs, take advantage of the compact nature of tables and express the relationships among data effectively. Tables are diverse structural objects that are ubiquitously used.



Flight nr	From	Departure	Destination	Arrival	Frequency	Flight duration	Via
RO801	BUCURESTI	21:00	BACAU	21:50	T T SS	0:50	DIRECT
RO802	BACAU	07:05	BUCURESTI	07:45	M	0:40	DIRECT
RO627	BUCURESTI	07:00	BAIA-MARE	09:20	T	2:20	DIRECT
RO627	BAIA-MARE	09:40	BUCURESTI	10:55	T	1:15	DIRECT
RO651	BUCURESTI	09:05	CLUJ-NAPOCA	10:05	M	1:00	DIRECT
RO653	BUCURESTI	13:05	CLUJ-NAPOCA	14:05	MTWT	1:00	DIRECT
RO647	BUCURESTI	17:25	CLUJ-NAPOCA	18:25	MTWT	1:00	DIRECT
RO649	BUCURESTI	21:15	CLUJ-NAPOCA	22:15	MTW	1:00	DIRECT
RO650	CLUJ-NAPOCA	06:50	BUCURESTI	07:40	M	0:50	DIRECT
RO652	CLUJ-NAPOCA	10:25	BUCURESTI	11:25	MTWT	1:00	DIRECT
RO654	CLUJ-NAPOCA	14:25	BUCURESTI	15:25	MTWTF	1:00	DIRECT
RO648	CLUJ-NAPOCA	18:45	BUCURESTI	19:45	MT	1:00	DIRECT
RO751	BUCURESTI	20:00	CONSTANTA	00:00	MTWTF	4:00	DIRECT
RO752	CONSTANTA	04:00	BUCURESTI	08:00	MTWTF	4:00	DIRECT
RO341/RO9334	BUCURESTI	08:45	IASI	13:10	M W F S	4:25	VIENA
RO703	BUCURESTI	14:40	IASI	15:45	M W F	1:05	DIRECT
RO709	BUCURESTI	20:45	IASI	21:50	MT	1:05	DIRECT
RO710	IASI	06:45	BUCURESTI	07:50	MT	1:05	DIRECT
RO9333/RO344	IASI	14:25	BUCURESTI	21:55	M	7:30	VIENA
RO704	IASI	16:05	BUCURESTI	17:10	M W F	1:05	DIRECT
RO637	BUCURESTI	07:00	ORADEA	08:20	M	1:20	DIRECT
RO637	ORADEA	08:40	BUCURESTI	10:45	M	2:05	DIRECT
RO637	BUCURESTI	07:00	SATU-MARE	09:10	M	2:10	DIRECT
RO637	SATU-MARE	09:30	BUCURESTI	10:45	M	1:15	DIRECT
RO341/RO9332	BUCURESTI	08:45	SIBIU	13:20	MTWTF S	4:35	VIENA
RO311/RO310	BUCURESTI	14:55	SIBIU	21:15	M	6:20	MUNICH

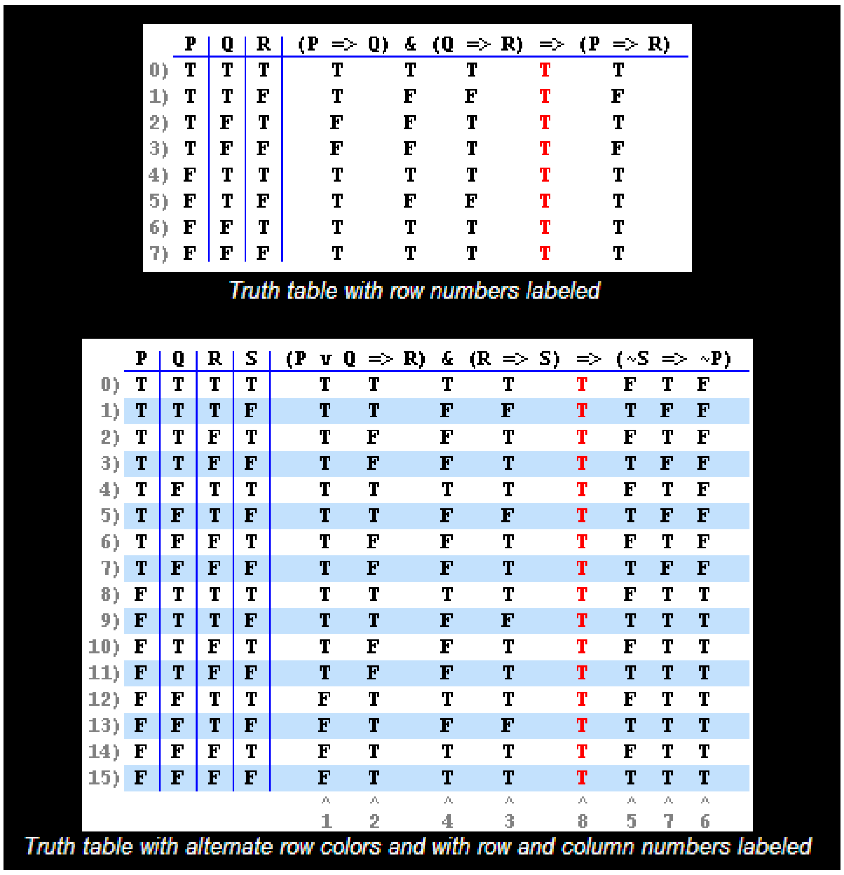
Figure 1.1: A bus timetable built for providing information to the public. (Source: `http://www.kaventravel.com/more/tarom.asp`. Last accessed on 28 November 2009.)


2000-2001 Winter Session Timetable - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://www.phys.uvic.ca/Timetables/Summer2004.htm
Go
start
WinEdt 5...
C:\vlong\...
Microsoft ...
litReview....
2000-200...
http://13...
C:\Tomca...
JewishCal...

SUMMER STUDIES 2003 TIMETABLE

TIME	*Monday*	*Tuesday*	*Wednesday*	*Thursday*	*Friday*
8:30					
9:00					
9:30	A120 R01 DF ELL 167	A120 R01 DF ELL 167	A120 R01 DF ELL 167	A120 R01 DF ELL 167	A120 R01 DF ELL 167
10:00	P325 K01 SL ELL 161			P325 K01 SL ELL 161	
10:30		P215 K01 DP ELL 161	P215 K01 DP ELL 161		P215 K01 DP ELL 161
11:00					
11:30		P317 K01 JBT ELL 161	P317 K01 JBT ELL 161		P317 K01 JBT ELL 161
12:00					
12:30		A200B K01 SY ELL 164	A200B K01 SY ELL 164		A200B K01 SY ELL 164
1:00	P216 K01 CDS ELL 061			P216 K01 CDS ELL 061	
1:30					
2:00					
2:30					
3:00					
3:30					
4:00					
4:30					
5:00					
5:30					
6:00					

Figure 1.2: Example of a timetable (Source: `http://www.phys.uvic.ca/Timetables/Summer2004.htm`. Last accessed on 13 May 2004.)



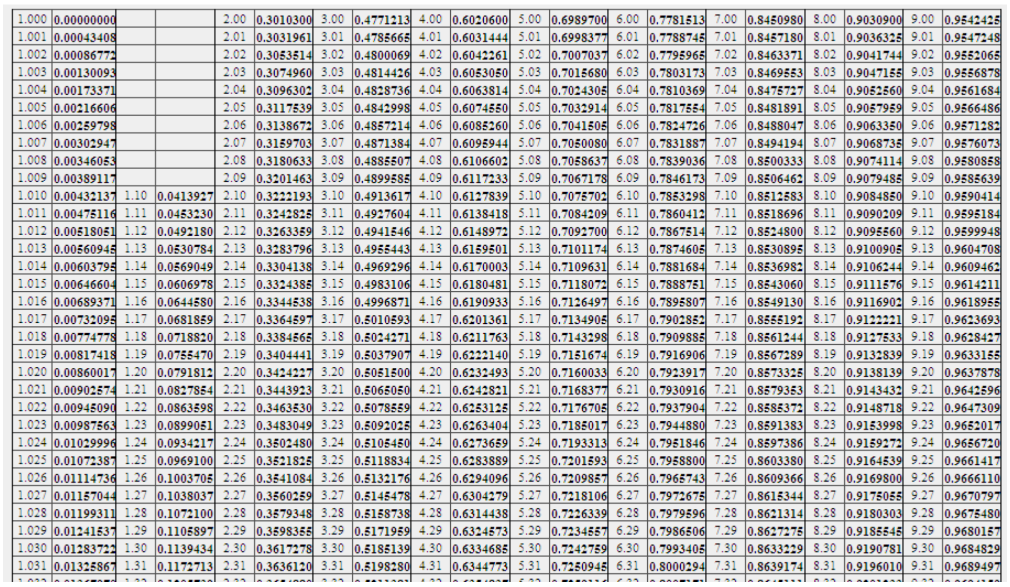
	P	Q	R	(P => Q)	&	(Q => R)	=>	(P => R)
0)	T	T	T	T	T	T	T	T
1)	T	T	F	T	F	F	T	F
2)	T	F	T	F	F	T	T	T
3)	T	F	F	F	F	T	T	F
4)	F	T	T	T	T	T	T	T
5)	F	T	F	T	F	F	T	T
6)	F	F	T	T	T	T	T	T
7)	F	F	F	T	T	T	T	T

Truth table with row numbers labeled

	P	Q	R	S	(P v Q	=> R)	&	(R => S)	=>	(~S	=>	~P)
0)	T	T	T	T	T	T	T	T	T	F	T	F
1)	T	T	T	F	T	T	F	F	T	T	F	F
2)	T	T	F	T	T	F	F	T	T	F	T	F
3)	T	T	F	F	T	F	F	T	T	T	F	F
4)	T	F	T	T	T	T	T	T	T	F	T	F
5)	T	F	T	F	T	T	F	F	T	T	F	F
6)	T	F	F	T	T	F	F	T	T	F	T	F
7)	T	F	F	F	T	F	F	T	T	T	F	F
8)	F	T	T	T	T	T	T	T	T	F	T	T
9)	F	T	T	F	T	T	F	F	T	T	T	T
10)	F	T	F	T	T	F	F	T	T	F	T	T
11)	F	T	F	F	T	F	F	T	T	T	T	T
12)	F	F	T	T	F	T	T	T	T	F	T	T
13)	F	F	T	F	F	T	F	F	T	T	T	T
14)	F	F	F	T	F	T	T	T	T	F	T	T
15)	F	F	F	F	F	T	T	T	T	T	T	T
					^ 1	^ 2	^ 4	^ 3	^ 8	^ 5	^ 7	^ 6

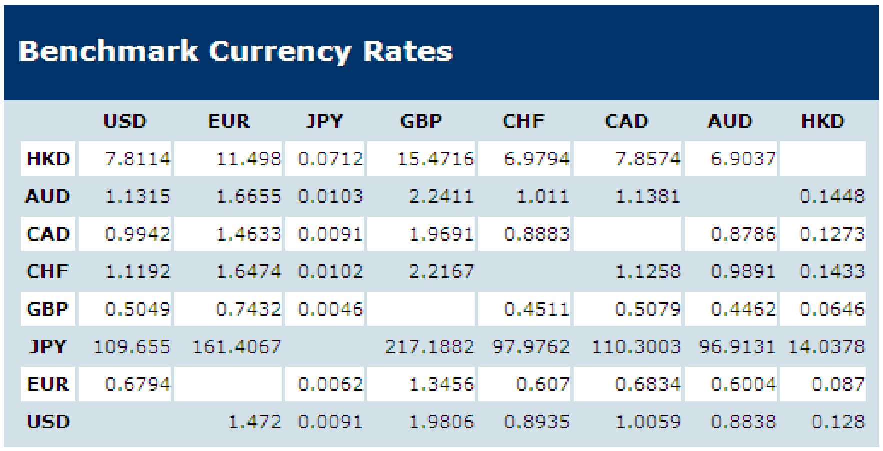
Truth table with alternate row colors and with row and column numbers labeled

Figure 1.3: An example of a table showing computational results for fast look up purposes. (Source: `http://www.merton.gov.uk/enrolment_form.pdf`. Last accessed on 28 November 2009.)



1.000	0.00000000			2.00	0.3010300	3.00	0.4771213	4.00	0.6020600	5.00	0.6989700	6.00	0.7781513	7.00	0.8450980	8.00	0.9030900	9.00	0.9542425
1.001	0.00043408			2.01	0.3031961	3.01	0.4785665	4.01	0.6031444	5.01	0.6998377	6.01	0.7788745	7.01	0.8457180	8.01	0.9036325	9.01	0.9547248
1.002	0.00086772			2.02	0.3053514	3.02	0.4800069	4.02	0.6042261	5.02	0.7007037	6.02	0.7795965	7.02	0.8463371	8.02	0.9041744	9.02	0.9552065
1.003	0.00130093			2.03	0.3074960	3.03	0.4814426	4.03	0.6053050	5.03	0.7015680	6.03	0.7803173	7.03	0.8469553	8.03	0.9047155	9.03	0.9556878
1.004	0.00173371			2.04	0.3096302	3.04	0.4828736	4.04	0.6063814	5.04	0.7024305	6.04	0.7810369	7.04	0.8475727	8.04	0.9052560	9.04	0.9561684
1.005	0.00216606			2.05	0.3117539	3.05	0.4842998	4.05	0.6074550	5.05	0.7032914	6.05	0.7817554	7.05	0.8481891	8.05	0.9057959	9.05	0.9566486
1.006	0.00259798			2.06	0.3138672	3.06	0.4857214	4.06	0.6085260	5.06	0.7041505	6.06	0.7824726	7.06	0.8488047	8.06	0.9063350	9.06	0.9571282
1.007	0.00302947			2.07	0.3159703	3.07	0.4871384	4.07	0.6095944	5.07	0.7050080	6.07	0.7831887	7.07	0.8494194	8.07	0.9068735	9.07	0.9576073
1.008	0.00346053			2.08	0.3180633	3.08	0.4885507	4.08	0.6106602	5.08	0.7058637	6.08	0.7839036	7.08	0.8500333	8.08	0.9074114	9.08	0.9580858
1.009	0.00389117			2.09	0.3201463	3.09	0.4899585	4.09	0.6117233	5.09	0.7067178	6.09	0.7846173	7.09	0.8506462	8.09	0.9079485	9.09	0.9585639
1.010	0.00432137	1.10	0.0413927	2.10	0.3222193	3.10	0.4913617	4.10	0.6127839	5.10	0.7075702	6.10	0.7853298	7.10	0.8512583	8.10	0.9084850	9.10	0.9590414
1.011	0.00475116	1.11	0.0453230	2.11	0.3242825	3.11	0.4927604	4.11	0.6138418	5.11	0.7084209	6.11	0.7860412	7.11	0.8518696	8.11	0.9090209	9.11	0.9595184
1.012	0.00518051	1.12	0.0492180	2.12	0.3263359	3.12	0.4941546	4.12	0.6148972	5.12	0.7092700	6.12	0.7867514	7.12	0.8524800	8.12	0.9095560	9.12	0.9599948
1.013	0.00560945	1.13	0.0530784	2.13	0.3283796	3.13	0.4955443	4.13	0.6159501	5.13	0.7101174	6.13	0.7874605	7.13	0.8530895	8.13	0.9100905	9.13	0.9604708
1.014	0.00603795	1.14	0.0569049	2.14	0.3304138	3.14	0.4969296	4.14	0.6170003	5.14	0.7109631	6.14	0.7881684	7.14	0.8536982	8.14	0.9106244	9.14	0.9609462
1.015	0.00646604	1.15	0.0606978	2.15	0.3324385	3.15	0.4983106	4.15	0.6180481	5.15	0.7118072	6.15	0.7888751	7.15	0.8543060	8.15	0.9111576	9.15	0.9614211
1.016	0.00689371	1.16	0.0644580	2.16	0.3344538	3.16	0.4996871	4.16	0.6190933	5.16	0.7126497	6.16	0.7895807	7.16	0.8549130	8.16	0.9116902	9.16	0.9618955
1.017	0.00732095	1.17	0.0681859	2.17	0.3364597	3.17	0.5010593	4.17	0.6201361	5.17	0.7134905	6.17	0.7902852	7.17	0.8555192	8.17	0.9122221	9.17	0.9623693
1.018	0.00774778	1.18	0.0718820	2.18	0.3384565	3.18	0.5024271	4.18	0.6211763	5.18	0.7143298	6.18	0.7909885	7.18	0.8561244	8.18	0.9127533	9.18	0.9628427
1.019	0.00817418	1.19	0.0755470	2.19	0.3404441	3.19	0.5037907	4.19	0.6222140	5.19	0.7151674	6.19	0.7916906	7.19	0.8567289	8.19	0.9132839	9.19	0.9633155
1.020	0.00860017	1.20	0.0791812	2.20	0.3424227	3.20	0.5051500	4.20	0.6232493	5.20	0.7160033	6.20	0.7923917	7.20	0.8573325	8.20	0.9138139	9.20	0.9637878
1.021	0.00902574	1.21	0.0827854	2.21	0.3443923	3.21	0.5065050	4.21	0.6242821	5.21	0.7168377	6.21	0.7930916	7.21	0.8579353	8.21	0.9143432	9.21	0.9642596
1.022	0.00945090	1.22	0.0863598	2.22	0.3463530	3.22	0.5078559	4.22	0.6253125	5.22	0.7176705	6.22	0.7937904	7.22	0.8585372	8.22	0.9148718	9.22	0.9647309
1.023	0.00987563	1.23	0.0899051	2.23	0.3483049	3.23	0.5092025	4.23	0.6263404	5.23	0.7185017	6.23	0.7944880	7.23	0.8591383	8.23	0.9153998	9.23	0.9652017
1.024	0.01029996	1.24	0.0934217	2.24	0.3502480	3.24	0.5105450	4.24	0.6273659	5.24	0.7193313	6.24	0.7951846	7.24	0.8597386	8.24	0.9159272	9.24	0.9656720
1.025	0.01072387	1.25	0.0969100	2.25	0.3521825	3.25	0.5118834	4.25	0.6283889	5.25	0.7201593	6.25	0.7958800	7.25	0.8603380	8.25	0.9164539	9.25	0.9661417
1.026	0.01114736	1.26	0.1003705	2.26	0.3541084	3.26	0.5132176	4.26	0.6294096	5.26	0.7209857	6.26	0.7965743	7.26	0.8609366	8.26	0.9169800	9.26	0.9666110
1.027	0.01157044	1.27	0.1038037	2.27	0.3560259	3.27	0.5145478	4.27	0.6304279	5.27	0.7218106	6.27	0.7972675	7.27	0.8615344	8.27	0.9175055	9.27	0.9670797
1.028	0.01199311	1.28	0.1072100	2.28	0.3579348	3.28	0.5158738	4.28	0.6314438	5.28	0.7226339	6.28	0.7979596	7.28	0.8621314	8.28	0.9180303	9.28	0.9675480
1.029	0.01241537	1.29	0.1105897	2.29	0.3598355	3.29	0.5171959	4.29	0.6324573	5.29	0.7234557	6.29	0.7986506	7.29	0.8627275	8.29	0.9185545	9.29	0.9680157
1.030	0.01283722	1.30	0.1139434	2.30	0.3617278	3.30	0.5185139	4.30	0.6334685	5.30	0.7242759	6.30	0.7993405	7.30	0.8633229	8.30	0.9190781	9.30	0.9684829
1.031	0.01325867	1.31	0.1172713	2.31	0.3636120	3.31	0.5198280	4.31	0.6344773	5.31	0.7250945	6.31	0.8000294	7.31	0.8639174	8.31	0.9196010	9.31	0.9689497

Figure 1.4: A table showing the logarithms (with base 10) for numbers between 1 and 10. The logarithm is denoted in bold font. (Source: `http://www.sosmath.com/tables/logtable/logtable.html`. Last accessed on 28 November 2009.)



Benchmark Currency Rates

	USD	EUR	JPY	GBP	CHF	CAD	AUD	HKD
HKD	7.8114	11.498	0.0712	15.4716	6.9794	7.8574	6.9037	
AUD	1.1315	1.6655	0.0103	2.2411	1.011	1.1381		0.1448
CAD	0.9942	1.4633	0.0091	1.9691	0.8883		0.8786	0.1273
CHF	1.1192	1.6474	0.0102	2.2167		1.1258	0.9891	0.1433
GBP	0.5049	0.7432	0.0046		0.4511	0.5079	0.4462	0.0646
JPY	109.655	161.4067		217.1882	97.9762	110.3003	96.9131	14.0378
EUR	0.6794		0.0062	1.3456	0.607	0.6834	0.6004	0.087
USD		1.472	0.0091	1.9806	0.8935	1.0059	0.8838	0.128

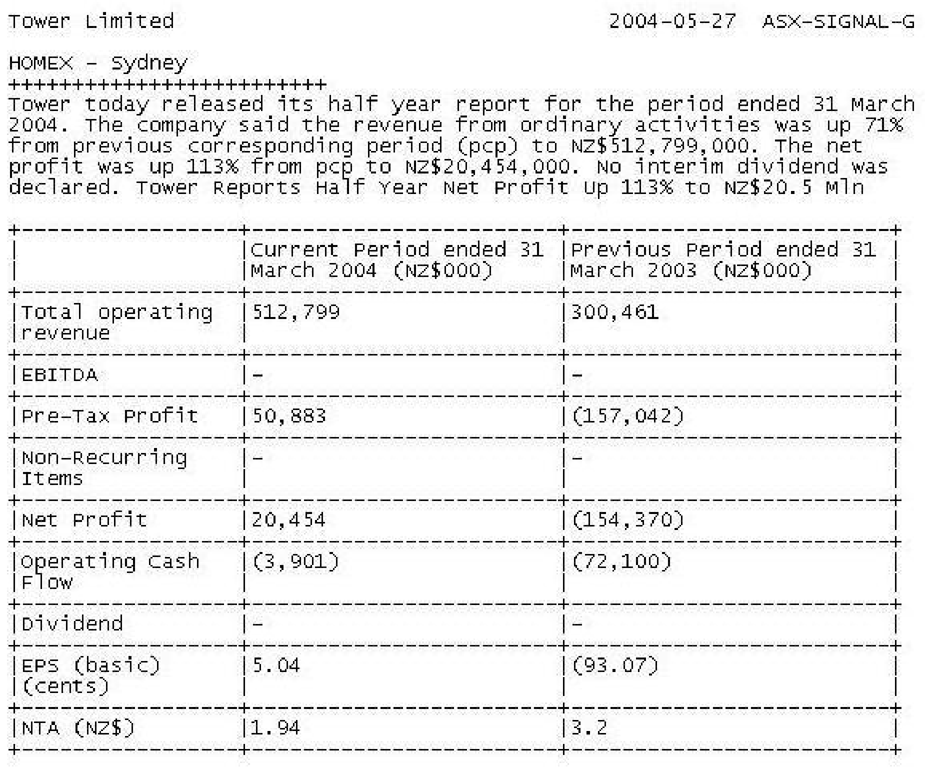
Figure 1.5: A currency conversion table showing the conversion rates between two pairs of currencies. The special property of this type of table is that the number of rows is equal to the number of columns, the cells on the diagonal line are empty, and the part of the table above the diagonal line can be computed from the one below the diagonal line. (Source: `http://www.bloomberg.com/markets/currencies/fxc.html`. Last accessed on 28 November 2009.)

Tables can be hosted in document images, plain text documents or markup documents. Each of these media uses different ways to represent tables. In document images, tables are made up of pixels; in plain text documents, tables are comprised of characters (including space characters and punctuation characters); in markup documents, tables are encoded in pairs of markup tags. The hosting media require different data representations, and they have different processing complexities. Since table processing algorithms that work for one medium almost never work for others, the different hosting media increase the dimensions of the problem.

Tables are characterised by arrays of cells fitted in column-row structures that are typically rectangular. A table's column-row structure is created by inserting *column and row delimiters* among content data. These delimiters can appear in various forms. Tables in plain text documents, for example, frequently use sequences of punctuation characters (see Figure 1.6 for example), blank lines (see Figure 1.7 for example), the new line character (see Figure 1.8 for example), or a combination of the above as row delimiters. Likewise, column delimiters can have various forms such as consecutive white space, and vertical bar characters. Row delimiters and column delimiters define the boundaries of rows, columns, and hence cells. They are thus highly relevant to the table structure recognition processes.

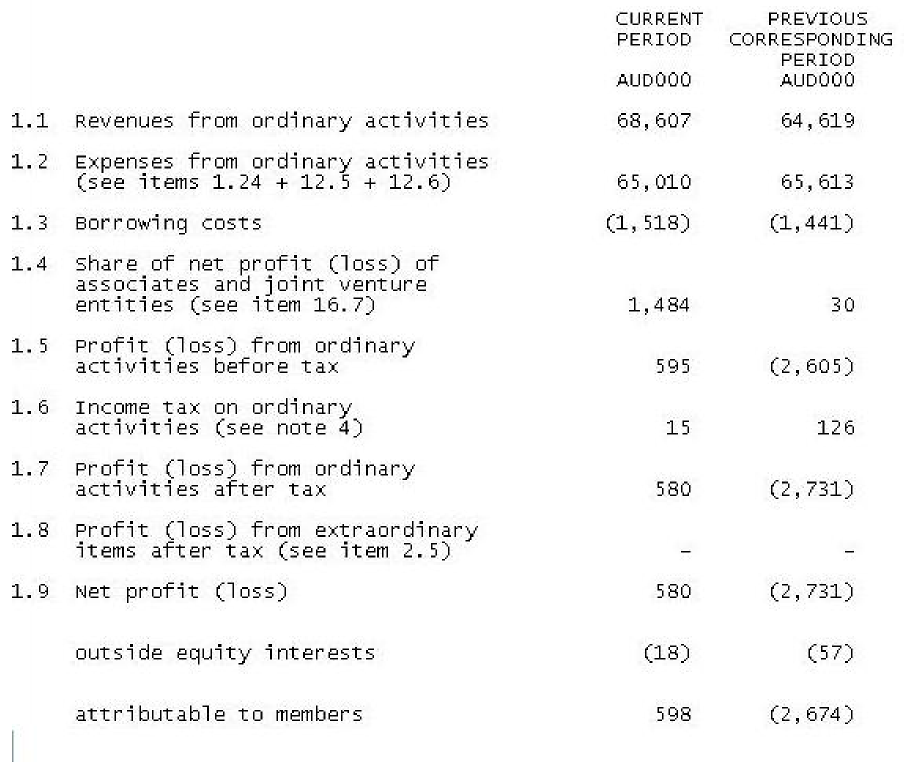
Table cells are place holders where table contents are stored. There is a wide range of information, such as symbols, text characters and graphs, that a table cell can store. A table cell can even store another table, in which case, the table is called a *nested table*. Most tables contain headers, which are cells containing descriptive terms for the data underneath. Headers are usually stored in the first row and/or column. Figure 1.9 shows an example of a table whose header contains a nested table.

Figure 1.10 gives an example of a special document class, called *table form documents* (or *form documents* for short), which takes advantage of the neat structures arising from the rectangular shape of the table cells. Due to the clear association between preprinted data fields and their corresponding user filled-in data, form documents are widely used for information collection by government agencies and businesses. Because of their heavy



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

Figure 1.6: An example of a table using ruling lines as explicit row delimiters. Extracted from ASX Corpus.

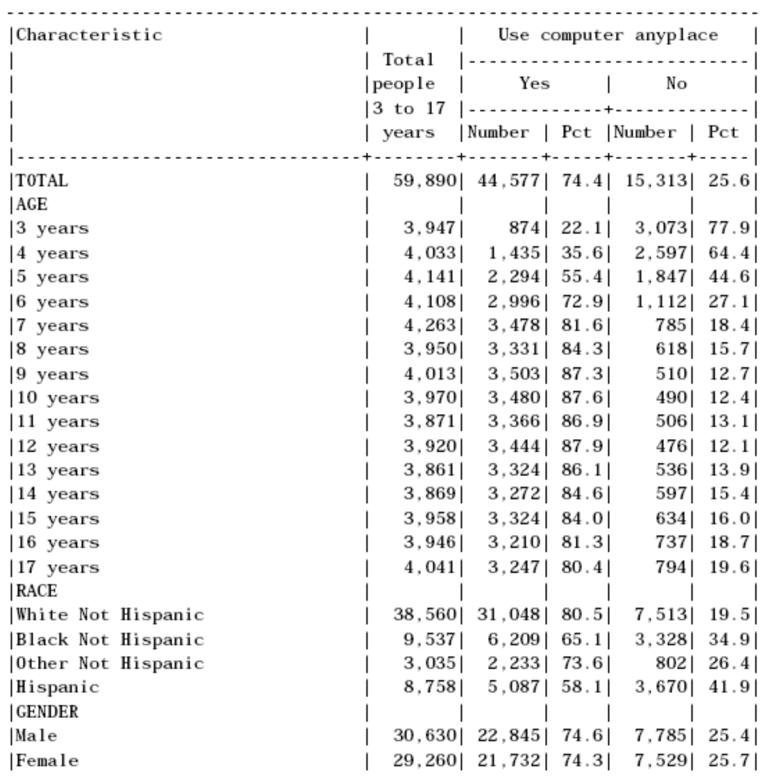


		CURRENT PERIOD AUD000	PREVIOUS CORRESPONDING PERIOD AUD000
1.1	Revenues from ordinary activities	68,607	64,619
1.2	Expenses from ordinary activities (see items 1.24 + 12.5 + 12.6)	65,010	65,613
1.3	Borrowing costs	(1,518)	(1,441)
1.4	Share of net profit (loss) of associates and joint venture entities (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)

Figure 1.7: An example of a table using blank lines as explicit row delimiters. Extracted from ASX Corpus.

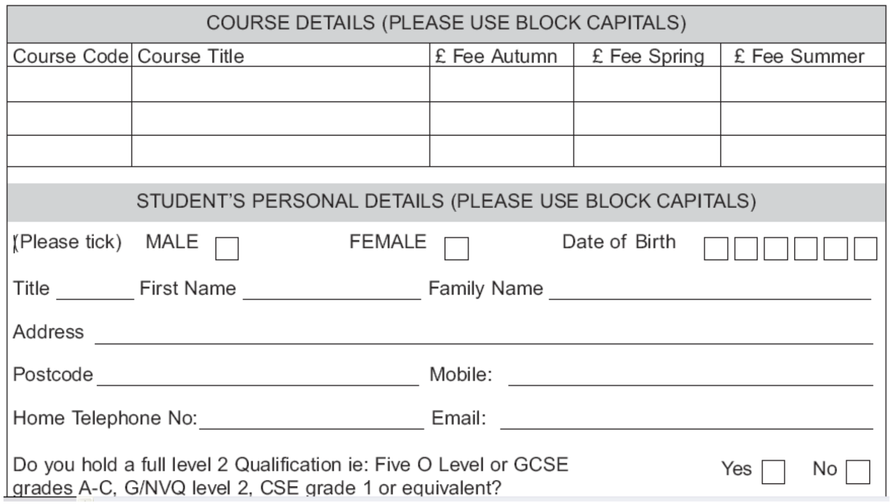
```
Sales revenue                              97,408      96,117
Cost of Sales                            (60,042)    (61,077)
Gross Profit                               37,366      35,040
Other Revenue                               1,762       1,570
Selling and marketing                    (21,042)    (17,437)
Distribution                              (3,499)     (3,181)
Administration                            (5,852)     (5,343)
Borrowing costs (net)                       (746)       (968)
```

Figure 1.8: An example of a table using implicit row delimiters. There is no explicit row delimiter used in this table. Extracted from ASX Corpus.



Characteristic	Total people 3 to 17 years	Use computer anyplace			
		Yes		No	
		Number	Pct	Number	Pct
TOTAL	59,890	44,577	74.4	15,313	25.6
AGE					
3 years	3,947	874	22.1	3,073	77.9
4 years	4,033	1,435	35.6	2,597	64.4
5 years	4,141	2,294	55.4	1,847	44.6
6 years	4,108	2,996	72.9	1,112	27.1
7 years	4,263	3,478	81.6	785	18.4
8 years	3,950	3,331	84.3	618	15.7
9 years	4,013	3,503	87.3	510	12.7
10 years	3,970	3,480	87.6	490	12.4
11 years	3,871	3,366	86.9	506	13.1
12 years	3,920	3,444	87.9	476	12.1
13 years	3,861	3,324	86.1	536	13.9
14 years	3,869	3,272	84.6	597	15.4
15 years	3,958	3,324	84.0	634	16.0
16 years	3,946	3,210	81.3	737	18.7
17 years	4,041	3,247	80.4	794	19.6
RACE					
White Not Hispanic	38,560	31,048	80.5	7,513	19.5
Black Not Hispanic	9,537	6,209	65.1	3,328	34.9
Other Not Hispanic	3,035	2,233	73.6	802	26.4
Hispanic	8,758	5,087	58.1	3,670	41.9
GENDER					
Male	30,630	22,845	74.6	7,785	25.4
Female	29,260	21,732	74.3	7,529	25.7

Figure 1.9: A table showing the use of computers at home, school, and work by people from ages 3 to 17 in October 1997. (numbers in thousands) (Source: `http://www.census.gov/population/socdemo/computer/report97/tab04.pdf`. Last accessed on 28 November 2009.) This table contains complex table headers as well as multiple views of the same data (views of data based on age, race and gender).



COURSE DETAILS (PLEASE USE BLOCK CAPITALS)				
Course Code	Course Title	£ Fee Autumn	£ Fee Spring	£ Fee Summer

STUDENT'S PERSONAL DETAILS (PLEASE USE BLOCK CAPITALS)

(Please tick) MALE ☐ FEMALE ☐ Date of Birth ☐☐☐☐☐☐

Title ______ First Name ______________ Family Name ____________________

Address ____

Postcode ____________________ Mobile: ____________________

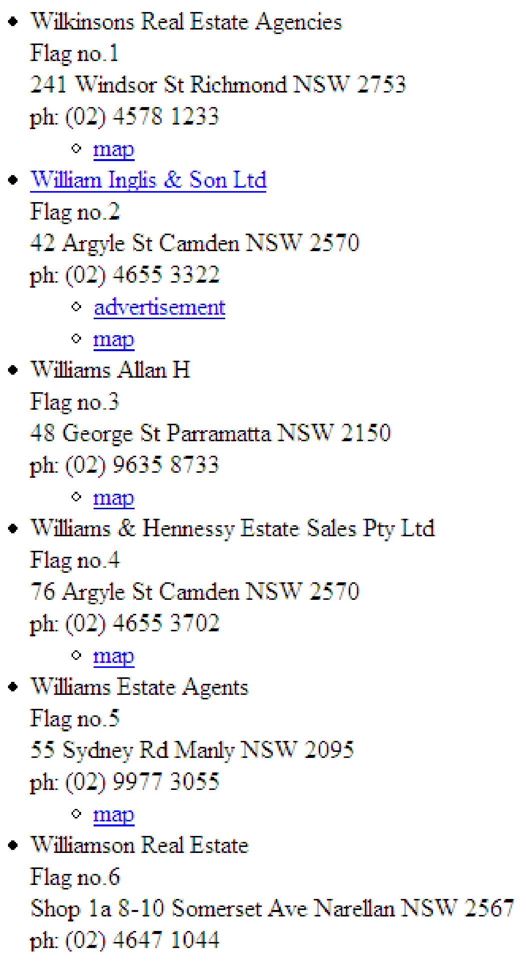
Home Telephone No: ______________ Email: ____________________

Do you hold a full level 2 Qualification ie: Five O Level or GCSE grades A-C, G/NVQ level 2, CSE grade 1 or equivalent? Yes ☐ No ☐

Figure 1.10: An example of a table-form document (Source: `http://www.merton.gov.uk/enrolment_form.pdf`. Last accessed on 28 November 2009.)

usage, and the importance of the information contained within, table form documents have attracted a fair amount of research attention (see, for example, Amano & Asada (2002), Amano, Asada & Mukunoki (2004), Chen & Tseng (1996), Y. Belaid & A. Belaid (1999)). It is worth noting that although the fields in table forms are typically rectangular, they are not necessarily arranged as aligned rows and columns (see, for example, the second half of the form-document in Figure 1.10), because the lengths and widths of form document fields can vary.

Tabular data often have recurring patterns, and tables display these patterns by organising the data into rows or columns. However, there are exceptions. Instead of organising data into columns and rows, data are sometimes written in non-tabular formats (as illustrated in Figure 1.11). Overall, tables are diverse objects in terms of their physical structures, their hosting media, the purposes of their creation, and the domains of their content. These diversities, together with other factors such as missing data and occasional irregularities of layout structures, make table processing challenging.



- Wilkinsons Real Estate Agencies
  Flag no.1
  241 Windsor St Richmond NSW 2753
  ph: (02) 4578 1233
  - map
- William Inglis & Son Ltd
  Flag no.2
  42 Argyle St Camden NSW 2570
  ph: (02) 4655 3322
  - advertisement
  - map
- Williams Allan H
  Flag no.3
  48 George St Parramatta NSW 2150
  ph: (02) 9635 8733
  - map
- Williams & Hennessy Estate Sales Pty Ltd
  Flag no.4
  76 Argyle St Camden NSW 2570
  ph: (02) 4655 3702
  - map
- Williams Estate Agents
  Flag no.5
  55 Sydney Rd Manly NSW 2095
  ph: (02) 9977 3055
  - map
- Williamson Real Estate
  Flag no.6
  Shop 1a 8-10 Somerset Ave Narellan NSW 2567
  ph: (02) 4647 1044

Figure 1.11: Example of data with repeating patterns. The data could be re-written in tabular form.

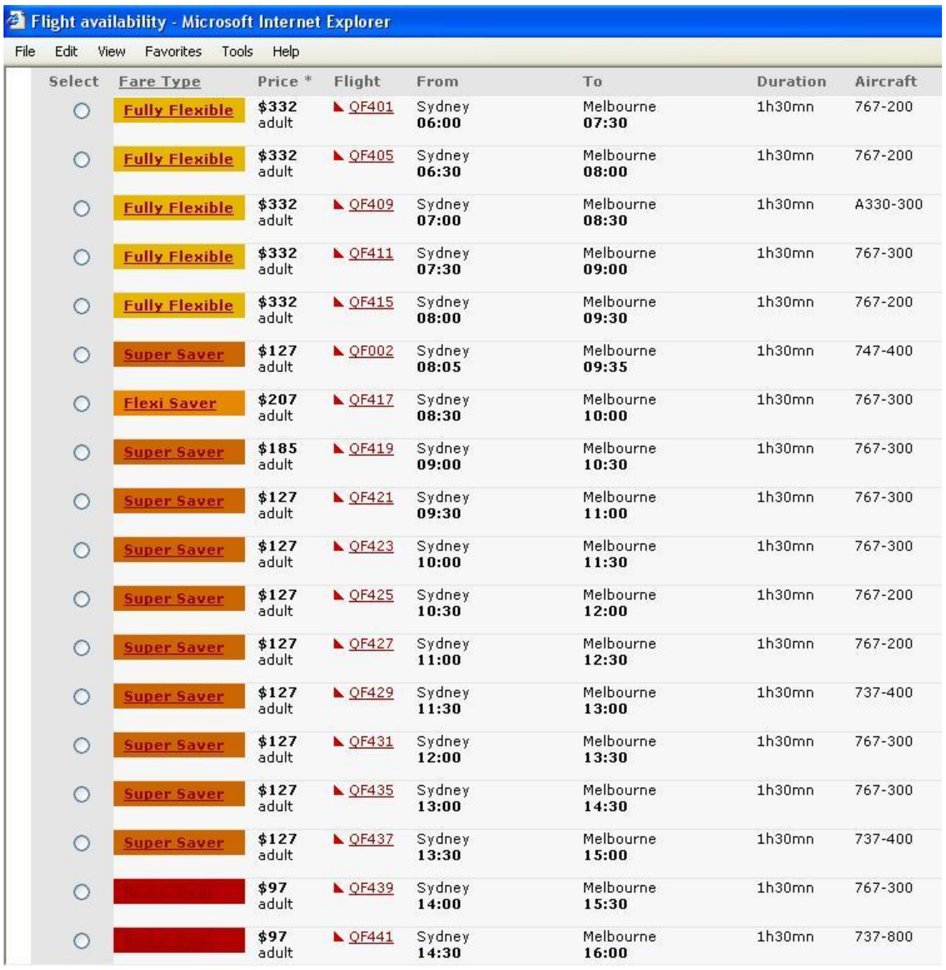
1.3 Motivations for Research on Tables

Research on tables is interesting for a number of reasons. The foremost one is the sheer ubiquity of tables; due to their expressive power and compact nature, tables are often the preferred means of storing, representing and communicating structured data. Because tables contain important data that concerns many people, there is a high demand for detecting and interpreting tabular data in documents. For example, the ASX corpus used in this dissertation contains financial reports and statutory announcements made by Australian public companies. Much data that are relevant to the public are stored in tables. A survey of more than 100,000 Wall Street Journal (WSJ) documents shows that at least 1 in 30 documents contains tables (Ng, Lim, & Koo, 1999). The heavy use of tables makes table analysis research worthwhile.

The richness and the important nature of information contained within tables gives the second reason for conducting research in this area. As mentioned in the previous section, financial data concerning millions of people across the world are published in tabular formats every day. In science and education, tables are often used to communicate grade reports, research results and other structural data. There have been desires for computer programs that can locate and understand the tables, so that key information such as profit (or loss) figures can be automatically extracted and compared. Sufficient research on table processing could lead to the understanding of tables, such as the one in Figure 1.12, by machines. With such understanding, many tasks that require human intervention could be performed by machines. For example, the following statements can be automatically generated for question-answering systems if the table in Figure 1.12 is properly understood:

- The most expensive ticket from Sydney to Melbourne is $332, and the cheapest ticket is $97.
- There are 13 morning flights from Sydney to Melbourne, and there are 5 flights in the afternoon.
- The earliest flight starts at 6am in Sydney, and the last flight leaves Sydney at 2:30pm.

- The travelling time between Sydney and Melbourne is 1.5 hours.
- Between 6am and 2:30pm, there are 2 flights from Sydney to Melbourne every hour: one departs on the hour, the other departs 30 minutes after the hour. However, there is an additional flight at 8:05am, and there is no flight at 12:30pm.


Flight availability - Microsoft Internet Explorer
File Edit View Favorites Tools Help

Select	Fare Type	Price *	Flight	From	To	Duration	Aircraft
○	Fully Flexible	$332 adult	QF401	Sydney 06:00	Melbourne 07:30	1h30mn	767-200
○	Fully Flexible	$332 adult	QF405	Sydney 06:30	Melbourne 08:00	1h30mn	767-200
○	Fully Flexible	$332 adult	QF409	Sydney 07:00	Melbourne 08:30	1h30mn	A330-300
○	Fully Flexible	$332 adult	QF411	Sydney 07:30	Melbourne 09:00	1h30mn	767-300
○	Fully Flexible	$332 adult	QF415	Sydney 08:00	Melbourne 09:30	1h30mn	767-200
○	Super Saver	$127 adult	QF002	Sydney 08:05	Melbourne 09:35	1h30mn	747-400
○	Flexi Saver	$207 adult	QF417	Sydney 08:30	Melbourne 10:00	1h30mn	767-300
○	Super Saver	$185 adult	QF419	Sydney 09:00	Melbourne 10:30	1h30mn	767-300
○	Super Saver	$127 adult	QF421	Sydney 09:30	Melbourne 11:00	1h30mn	767-300
○	Super Saver	$127 adult	QF423	Sydney 10:00	Melbourne 11:30	1h30mn	767-300
○	Super Saver	$127 adult	QF425	Sydney 10:30	Melbourne 12:00	1h30mn	767-200
○	Super Saver	$127 adult	QF427	Sydney 11:00	Melbourne 12:30	1h30mn	767-200
○	Super Saver	$127 adult	QF429	Sydney 11:30	Melbourne 13:00	1h30mn	737-400
○	Super Saver	$127 adult	QF431	Sydney 12:00	Melbourne 13:30	1h30mn	767-300
○	Super Saver	$127 adult	QF435	Sydney 13:00	Melbourne 14:30	1h30mn	767-300
○	Super Saver	$127 adult	QF437	Sydney 13:30	Melbourne 15:00	1h30mn	737-400
○	[illegible]	$97 adult	QF439	Sydney 14:00	Melbourne 15:30	1h30mn	767-300
○	[illegible]	$97 adult	QF441	Sydney 14:30	Melbourne 16:00	1h30mn	737-800

Figure 1.12: A table containing flight information

The third reason for studying tables is related to the advancement of existing NLP systems. Research shows that having tables in documents makes document processing difficult. For example, Pinto, McCallum, Lee and Croft (2003) found that the performance of their question-answering system dropped dramatically when the answers were hidden in tables. Being able to automatically identify, extract and understand tables not only clears the obstacles existing NLP systems face, but also opens up many new opportunities for higher-level document processing. These opportunities include: populating databases with correct data, consolidating tables from multiple sources, providing audio access to tabular information, generating a domain specific ontology that might be difficult to produce from non-table contents (that is sentential text) (Embley, Tao, & Liddle, 2002; Tijerino, Embley, Lonsdale, & Nagy, 2003), restructuring the table on demand to generate different presentational formats, and producing summarised descriptions of the table (X. Wang, 1996; Wang & Wood, 1993a, 1993b).

Overall, tables are ubiquitous, tables often contain important data that is of interest to many people, and yet, the versatility, the applicabilities and accuracies of existing table processing systems have not reached a satisfactory level. For these reasons, further research on tables should be supported.

1.4 Scope of this Research

Table processing is a large research area that contains many overlapping sub-problems: to locate tables within documents, to segment tables into columns, rows, or cells, to identify column/row functions, to determine table headers, and to discover property-attribute pairs in tables are just some examples. Based on the natural progression, these tasks can be roughly categorised into three sub-areas: table boundary identification, which is about locating tables in documents; table structure recognition, which is about identifying columns and rows, hence cells, within table blocks; and table interpretation, which is about understanding tables (Hu, Kashi, Lopresti, & Wilfong, 2002; Lopresti & Nagy, 1999a; Y. Wang, 2000).

For a given table processing task, there are multiple input document types that the task could be performed upon: plain text documents, markup documents and document images. For example, table identification can be performed on HTML documents as well as plain text documents. For HTML documents, tables are found within ⟨TABLE⟩ tags; but for plain text documents, tables are identified by using blank lines, ruling lines and sequences of whitespace characters. The number of tasks, combined with the number of input document types, makes table processing a large research area.

The core of this dissertation is a framework that illustrates the benefits and the feasibility of applying multiple knowledge sources to table processing. Given the size of the research area, it is not feasible, and also not necessary, to conduct all table processing experiment types on all input media. The benefits and the feasibility are illustrated by solving the table boundary identification problem using the ASX corpus, a collection of plain text documents. These documents are chosen for a number of reasons. Firstly, these documents are readily available and their data patterns are unpredictable - a characteristic of documents in the real world. CMCRC, a research organisation funded by the Australian government, has granted permission to use its ASX corpus, which contains Australian public company reports. The corpus contains 19 document types of various sizes and structures: ranging from simple, short documents such as substantial holding notifications to complex documents such as annual reports. Some of these documents contain tables, which have no fixed structures, and whose lengthes and locations cannot be predicted. These qualities ensure that the table identification experiments conducted on these documents are not trivial.

The second reason is the scope of challenges represented by these documents. In order to identify and interpret tables contained within these documents, certain problems must be solved. In particular, this dissertation proposes an agent-based approach to table analysis, and in supporting this approach, a data representation scheme, an agent communication method, a confliction resolution strategy and an evaluation method were decided. All of these can be applied to other document types with little modification.

Finally, plain text documents are commonly used in the real world; they are one of the official document types accepted by government agencies and regulating organisations. Using real world documents adds potential industrial impacts to the research. Overall, the complexity, diversity and relevance of these documents make them suitable as test data in this dissertation.

1.5 Contributions

The goal of this research is to improve methods for processing tabular data embedded in printed documents. The pursuit of these solutions results in two main contributions. The first one is a *blackboard* based framework, which has the ability to incorporate a wide range of knowledge sources, for table analysis. Compared to traditional programming architectures, the blackboard framework makes it easy to add or to remove knowledge sources in a table processing task. This, in turn, results in several other benefits: it allows participation of partial knowledge in table analysis tasks; it encourages development of diverse and independent knowledge sources; it helps integrate programs from different programming paradigms; and it provides flexibility of experimenting and evaluating the effectiveness of individual knowledge sources.

The blackboard architecture, which was conceived in the 1980s for solving the speech recognition problem, was successfully applied to table analysis tasks in this research project because of a key innovation: the ability of finding a suitable knowledge representation and communication protocol that support knowledge sharing. This innovation provides the foundation for introducing multiple sources of knowledge in table analysis.

The second main contribution is a multi-level table structural evaluation matrix that has finer granularity than the popular evaluation measures of recall and precision. Recall and precision have been the mainstream reporting measures for table structural analysis systems. Although these measures provide an overall measure for the strengths and weaknesses of the systems, they cannot give feedback on partially correct results. The evaluation matrix in this dissertation outputs error types, such as insertion, deletion, splitting and merging errors, at cell-level, row-level and table-level. Through experiments, this dissertation shows not only the feasibility but also the benefits arising from these innovations.

1.6 The Organisation of this Dissertation

Chapter 2 discusses published information on the topic of table processing. It surveys knowledge and ideas that have been established, to show the areas that could be improved upon, and to provide the links between the work in the literature and the work in this dissertation.

Chapters 3 and 4 contain the main innovations of this dissertation. Chapter 3 discusses the motivation and the details of the blackboard architecture for table analysis. In particular, it provides solutions to the key problems that a general blackboard architecture must solve. Chapter 4 discusses table definition, annotation and evaluation methods, which facilitate the main contribution covered in Chapter 3. It defines how experimental results are represented, and how experiments are evaluated.

Chapters 5 and 6 illustrate the feasibility of the blackboard architecture and the usefulness of the knowledge representation using experiments - one for table recognition and one for table interpretation. Chapter 7 summarises the dissertation and provides suggestions for future work.

2 Literature Review

2.1 Introduction

Since the invention of computers, research has been devoted to various aspects of table processing, resulting in many applications. Broadly speaking, these applications fall into three categories: table composition systems that free humans from rendering physical tables manually, table extraction systems that determine table locations and structures from documents, and higher level table processing systems that attempt to understand table contents. This chapter surveys the table processing literature by looking at how these three categories of applications relate to each other.

Previous surveys have provided details covering existing table processing work with different focuses. Lopresti and Nagy (1999a, 1999b) conducted two surveys: one showing the challenges in detecting tables due to the variety of extant table forms, and the other covering the composition and use of tables. Focusing on the table recognition process, Zanibbi, Blostein, and Cordy (2004) discussed what tables (table models) are, what features are used when detecting tables, how to re-structure tables, and how to generate and test table hypotheses. Maoa, Rosenfelda, and Kanungob (2003) provided information on table analysis algorithms (both for analysing physical structures and logical structures) and various evaluation metrics. Embley, Hurst, Lopresti, and Nagy (2006) outlined the reasons for processing tables, the different models of tables, how to detect tables, the commercial application of table processing, and the research direction in current table processing work. The survey in this chapter differs from previous surveys in that it collects information about the ideas that have been established on the topic of table detection and interpretation; it identifies gaps in the area of table processing; and it shows the relationships between the work in this dissertation and what has already been achieved in the literature.

2.2 Levels of Descriptions of Tables

Before a table is drawn, it exists in the author's mind. It encapsulates and encodes the communicative intent of the author. It could be the relationships between data, or the changing pattern of the data that the author wants to convey. Tables at this level are called *abstract tables* or *abstract forms of tables* (X. Wang & Wood, 1993a).

Ultimately, abstract tables are communicated via *physical tables*, which consist of pixels, lines and text located in documents or other display devices. Physical tables are also referred to as the layout structures of the tables (Haralick, 1994; Hurst, 1999b; Nagy, 2000). Physical tables are seldom constructed manually; they are often created by tools based on some high level of description of the tables. For example, tables in HTML documents are rendered by browsers based on markup tags. Similarly, tables in PDF documents can be created from table description strings that meet the LaTex syntax. The tables, which are encoded in high levels of language, bridge abstract tables and physical tables by defining the table components and how they are arranged. Tables at this level are called *logical tables* (Haralick, 1994; Nagy, 2000; X. Wang, 1996; X. Wang & Wood, 1993a).

Logical tables describe the arrangement of table components. When creating physical tables, table rendering tools accept formatting specifications in addition to the logical descriptions of the tables. For example, HTML browsers allow style sheet information to be added to table descriptions. A logical table and its formatting attributes for rendering make up of the presentational form of the table (X. Wang, 1996; X. Wang & Wood, 1993a, 1993b).

An abstract table can be presented in different physical, hence logical, forms (Douglas, Hurst, & Quinn, 1995; X. Wang, 1996; X. Wang & Wood, 1993a). For example, Figures 2.1 and 2.2 convey the same inter-cell relationships, although the table cells are arranged differently. Douglas et al. (1995); X. Wang and Wood (1993a) note that the main difference between these tables are the search keys contained within them. X. Wang (1996) showed that how easily tables can be read depends on the presentational forms selected by the designer.

Year	Term	Mark					
		Assignments			Examinations		Grade
		Ass1	Ass2	Ass3	Midterm	Final	
1991	Winter	85	80	75	60	75	75
	Spring	80	65	75	60	70	70
	Fall	80	85	75	55	80	75
1992	Winter	85	80	70	70	75	75
	Spring	80	80	70	70	75	75
	Fall	75	70	65	60	80	70

Figure 2.1: A table of student marks. This table conveys the same information as Figure 2.2.

Term	Assignments	1991	1992
Winter	Ass1	85	85
	Ass2	80	80
	Ass3	75	70
	Midterm	60	70
	Final	75	75
	Grade	75	75
Spring	Ass1	80	80
	Ass2	65	80
	Ass3	75	70
	Midterm	60	70
	Final	70	75
	Grade	70	75
Fall	Ass1	80	75
	Ass2	85	70
	Ass3	75	65
	Midterm	55	60
	Final	80	80
	Grade	75	70

Figure 2.2: A table of student marks. This table conveys the same information as Figure 2.1.

2.2.1 Tables at the Abstract Level

At the abstract level, tables can be viewed as sets of functions associating labels with values. X. Wang and Wood (1993a, 1993b) define an abstract table as *a 4-tuple* $(n, \{D_1, D_2, ..., D_n\}, E, \sigma)$ *where n is a nonnegative integer denoting the dimension (the number of domains) of a table,* $D_1, D_2, ..., D_n$ *are different labeled domains, E is a set of entries, and* σ *is a partial function that maps from* $D_1, D_2, ..., D_n$ *onto E.* Under this definition, Figure 2.1 and the functions for assignment 1 in Figure 2.2 can be abstracted as follows.

$$\begin{aligned}
&(3, \{Year,\ Term,\ Mark\}, E, \sigma) where \\
&Year = \{1991,\ 1992\} \\
&Term = \{Winter,\ Spring,\ Fall\} \\
&Mark = \{Assignments,\ Examinations,\ Grade\} \\
&Assignments = \{Ass1,\ Ass2,\ Ass3\} \\
&Examinations = \{Midterm,\ Final\} \\
&E = \{55,\ 60,\ 65,\ 70,\ 75,\ 80,\ 85\}
\end{aligned}$$

$$\begin{aligned}
&\sigma(\{Year.1991,\ Term.Winter,\ Assignments.Ass1\}) = 85 \\
&\sigma(\{Year.1991,\ Term.Spring,\ Assignments.Ass1\}) = 80 \\
&\sigma(\{Year.1991,\ Term.Fall,\ Assignments.Ass1\}) = 80 \\
&\sigma(\{Year.1992,\ Term.Winter,\ Assignments.Ass1\}) = 85 \\
&\sigma(\{Year.1992,\ Term.Spring,\ Assignments.Ass1\}) = 80 \\
&\sigma(\{Year.1992,\ Term.Fall,\ Assignments.Ass1\}) = 75
\end{aligned}$$

Similar to X. Wang and Wood (1993a, 1993b)'s definition, there is another definition, called the *canonical table* or the *canonical form of tables*, discussed by Embley et al. (2006); Douglas et al. (1995); Hurst and Douglas (1997b). A canonical table is defined as *a set of functions* $T = \{t_1,\ ...,\ t_m\}$ *from L to D with the restriction that for each function* $t \in T$, $t(L_i) \in D_i, 1 \leq i \leq n$, *where* $L = l_1, ..., l_n$ *is a set of labels or phrases,*

D_i is a set of domains corresponding to L_i, and $D = D_1 \cup ... \cup D_n$ is the union of some domains. Under this definition, the canonical form for Figure 2.1 can be expressed as follows.

$$
\begin{array}{l}
\{ \\
\{ \\
(Year, 1991), \\
(Term, Winter), \\
(Mark.Assignments.Ass1, 85), \\
(Mark.Assignments.Ass2, 80), \\
(Mark.Assignments.Ass3, 75), \\
(Mark.Examinations.Midterm, 60), \\
(Mark.Examinations.Final, 75), \\
(Mark.Grade, 75) \\
\}, \\
\{ \\
(Year, 1991), \\
(Term, Spring), \\
(Mark.Assignments.Ass1, 80), \\
(Mark.Assignments.Ass2, 65), \\
(Mark.Assignments.Ass3, 75), \\
(Mark.Examinations.Midterm, 60), \\
(Mark.Examinations.Final, 70), \\
(Mark.Grade, 70) \\
\},
\end{array}
$$

$\{$

$(Year, 1991)$,

$(Term, Fall)$,

$(Mark.Assignments.Ass1, 80)$,

$(Mark.Assignments.Ass2, 85)$,

$(Mark.Assignments.Ass3, 75)$,

$(Mark.Examinations.Midterm, 55)$,

$(Mark.Examinations.Final, 80)$,

$(Mark.Grade, 75)$

$\}$,

$\{$

$(Year, 1992)$,

$(Term, Winter)$,

$(Mark.Assignments.Ass1, 85)$,

$(Mark.Assignments.Ass2, 80)$,

$(Mark.Assignments.Ass3, 70)$,

$(Mark.Examinations.Midterm, 70)$,

$(Mark.Examinations.Final, 75)$,

$(Mark.Grade, 75)$

$\}$,

$$
\begin{aligned}
&\{\\
&(Year, 1992),\\
&(Term, Spring),\\
&(Mark.Assignments.Ass1, 80),\\
&(Mark.Assignments.Ass2, 80),\\
&(Mark.Assignments.Ass3, 70),\\
&(Mark.Examinations.Midterm, 70),\\
&(Mark.Examinations.Final, 75),\\
&(Mark.Grade, 75)\\
&\},\\
&\{\\
&(Year, 1992),\\
&(Term, Fall),\\
&(Mark.Assignments.Ass1, 75),\\
&(Mark.Assignments.Ass2, 70),\\
&(Mark.Assignments.Ass3, 65),\\
&(Mark.Examinations.Midterm, 60),\\
&(Mark.Examinations.Final, 80),\\
&(Mark.Grade, 70)\\
&\}\\
&\}
\end{aligned}
$$

The terminology used in these two definitions are slightly different. The meaning of 'domain' (denoted as D) in the first definition is equivalent to the meaning of 'labels' (denoted as L) in the second definition, and the 'entries' (denoted as E) in the first definition is equivalent to the 'domain' (denoted as D) in the second definition. Using Figure 2.14 as an example, the first column (the names of the provinces) is a set of domains in the first definition, but it is a set of labels in the second definition. The second column (the number of students in each province) is a set of entries in the first definition, but it is a set of domains in the second definition. Despite the differences in terminology, these definitions regard a table as a set of functions that map labels to some data. Definitions at this level emphasise the relationships between data rather than the column-row (physical) characteristics of tables. These definitions are adopted in a number of table processing works that search attribute-value pairs (X. Wang, 1996; Yoshida, Torisawa, & Tsujii, 2001; X. Wang & Wood, 1993a, 1993b). Although these definitions allow data to be searched, they fail to capture the column-row characteristics in tables.

The relational model, which is what the relational database is based upon, organises data into tables called *relations*. In the relational model, information is stored in tables such as the one shown in Figure 2.3. Columns of a table are given names called the *attributes*, each row in a table is called a *tuple*, and a collection of relations is called *a database*. In relational tables, operations such as joining tables together, finding the common data shared by tables, and finding the differences between tables can be performed. Relational algebra is a mathematical language developed to allow manipulation of tables. One of the operations is the *selection* function, which selects data that satisfies certain conditions from a table. The selection function takes the form of $\sigma_C(R)$ where C is the condition that selected data must satisfied, and R is the table which the operation is performed upon. If the table being operated on is understood, the function can be abbreviated to σ_C. Using this notation, the result of expression $\sigma_{year=1993}$ for Figure 2.3 is 757918, and the results of expression $\sigma_{year>1995}$ are 714079 and 703293.

Year	NumberOfStudents
1993	757918
1994	741630
1995	729116
1996	714079
1997	703293

Figure 2.3: Table as a relational model

The general connection between an abstract table and its relational data model is the following. First, given a relational table, one can enumerate each piece of data using the selection function; the reverse of this operation is to construct a table from the abstract data models discussed above.

Second, the order in which the tuples are listed in a relational table has no significance. That is, the tuples can be re-arranged in any way without changing the meaning of the table. This is also true for the relationships between data in tables at the abstract level, but not necessarily true for all tables at the physical level. For example, the rows in a bus timetable are sorted chronologically, and the purpose of the timetable might be defeated if the row order is changed. Third, a relational data model does not permit composite cells, spanned cells and nested tables. Such restrictions do not exist for tables at the physical level. Finally, the relational model is about manipulating data in tables via relational algebra, and it has little relevance to recognising tables.

2.2.2 Tables at the Logical Level

2.2.2.1 Tree Models

It is well understood that table structures can be represented by trees. The *x-y tree* structure was first used by Nagy and Seth (1984) to represent tables. Since then, it has gained wide acceptance, and examples of its applications include building hierarchical representations of form documents (see, for example, Duygulu, Atalay, and Dincel (1998); Duygulu and Atalay (2000)), recognising table structures (see, for example, Ha, Haralick, and Phillips (1995); Sylwester and Seth (2001); Zou, Le, and Thoma (2007)), and extracting content data from tables (see, for example, Krupl, Herzog, and Gatterbauer (2005); Abu-Tarif (1998)).

To obtain a table's structure, the x-y cut algorithm recursively splits the table into rectangular blocks by alternating horizontal and vertical cuts along row delimiters and column delimiters. For example, a horizontal cut can segment the table in Figure 2.4 into two halves: the top half that contains cell A, and the bottom half that contains cells B and C, which can be further segmented by applying a vertical cut to the bottom half. The cut regions are represented by an x-y tree: the root of the tree is associated with the whole table, a leaf node represents a single cell, and a non-leaf node represents a region that can be further split, for example, into rows or columns. The characteristic of an x-y tree is that regions at even levels in the tree are cut along the x-axis, while regions at odd levels in the tree are cut along the y-axis (with the root being level 1). Figure 2.5 shows the x-y tree structure for the table in Figure 2.4. The structure of a x-y tree corresponds to the structure of the table it represents: the more columns or rows a table contains, the more internal nodes its x-y tree has; the hierarchy of a tree corresponds to the hierarchy of the blocks in the table; and the layout structure of a table can be recovered by traversing the tree.

A	
B	C

Figure 2.4: Table cells resulted from applying the x-y cut algorithm.

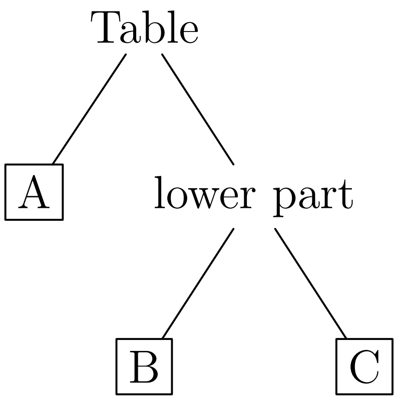


Figure 2.5: The x-y tree representation for the table cells in Figure 2.4

2.2.2.2 Graph Models

Amano and Asada (2002, 2003) proposed a graph representation scheme that directly encodes the geometric relationship between table cells: each cell is represented by a vertex, and there is an edge between two vertices if the cells represented by the vertices are adjacent. There are ten adjacency patterns between any two cells: the cells are horizontally aligned, and they have the same heights (see, for example, the geometric relationships between cells 2 and 3 in Figure 2.7); the left hand side cell boundary contains the border between the two cells (see, for example, the geometric relationships between cells 1 and 2 in Figure 2.7); the left hand side cell boundary is totally contained within the border between the two cells; the cells are vertically aligned, and they have the same widths (see, for example, the geometric relationships between cells 4 and 7 in Figure 2.7); the upper cell boundary contains the border between the two cells (see, for example, the geometric relationships between cells 3 and 6 in Figure 2.7); the upper cell boundary is totally contained within the border between the two cells (see, for example, the geometric relationships between cells 6 and 9 in Figure 2.7); the vertically aligned cells partially overlap (there are two sub-patterns: cells 3 and 5 in Figure 2.7, and cells 5 and 9 in Figure 2.7); and the horizontally aligned cells partially overlap.

In this representation method, the geometric relations between table cells are reflected by the edge types in the graph. For example, Figure 2.7 is the graph representation for Figure 2.6. The graphs can be encoded as XML documents meeting the *Document Type Definition* (DTD) specification mentioned by Amano and Asada (2002, 2003). By identifying all the possible layout relations any two cells can have, this representation paves the way for describing table layouts using context-free grammars. However, compared to a general graph representation, such as the RDF representation used in this dissertation, this representation scheme restricts the graph edge to one of the ten edge types.

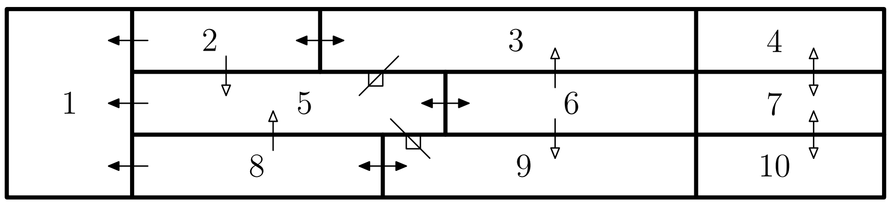


Figure 2.6: Various table cell adjacency types. The figure can be represented by the graph in Figure 2.7

$$\begin{tikzcd}
& \text{box2} \arrow[r, leftrightarrow] \arrow[dl] \arrow[d, -{Triangle[open]}] & \text{box3} \arrow[r, leftrightarrow] & \text{box4} \arrow[d, {Triangle[open]}-{Triangle[open]}] \\
\text{box1} & \text{box5} \arrow[l] \arrow[r, leftrightarrow] \arrow[ur, no head, "\square" marking] \arrow[dr, no head, "\square" marking] & \text{box6} \arrow[r, leftrightarrow] \arrow[u, -{Triangle[open]}] \arrow[d, -{Triangle[open]}] & \text{box7} \arrow[d, {Triangle[open]}-{Triangle[open]}] \\
& \text{box8} \arrow[ul] \arrow[u, -{Triangle[open]}] \arrow[r, leftrightarrow] & \text{box9} \arrow[r, leftrightarrow] & \text{box10}
\end{tikzcd}$$

Figure 2.7: The graph representation for the table in Figure 2.6

2.2.2.3 Grid Models

To address the problem of irregular physical layouts in tables, Ramel, Crucianu, Vincent, and Faure (2003) proposed a representation scheme that makes every table a perfect rectangle. Suppose that a table contains some spanned cells or misaligned cells (the cells in the original tables are referred to as *real cells* by the authors). These irregular cells can be eliminated by extending the borders of the smallest rectangles between the real cells up to the table's boundary, and the new cells created by the extended borders are called *virtual cells*. For example, the table cell in Figure 2.8 contains three real cells: cells A, B and C with cell A spanning over cells B and C. By extending the border between cells B and C, two virtual cells can be created (as illustrated by Figure 2.9). According to the authors, a real cell is made up of one or more virtual cells, and every table consists of a perfectly regular matrix of virtual cells. Thus, the table cell in Figure 2.8 sits on a matrix of four cells: physical cell A, which is made up of the top two virtual cells in Figure 2.9, and physical cells B and C, which are make up of virtual cells B and C in Figure 2.9 respectively. Tables under this representation are encoded as XML documents, such as the ones discussed by OASIS (1999, 1996); Crucianu, Ayadi, and Vincent (2001). The CALS model encodes a table row by row, and within each row, cells are listed. As a result, columns cannot be seen as easily as rows. Additionally, inserting or deleting columns is not as easy as inserting or deleting rows.

A	
B	C

Figure 2.8: A table contains three real cells under the grid model.

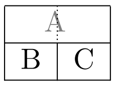


Figure 2.9: Virtual table cells in the grid model. This table is equivalent to Table 2.8, but with virtual cells marked by the dotted lines.

2.2.2.4 Dual Hierarchy Models

In the dual hierarchy representation, a table is represented as a row hierarchy and a column hierarchy simultaneously (Biggerstaff, Endres, & Forman, 1984; Beach, 1985; Cameron, 1989; Furuta, 1986). In this representation, a table contains two linked lists: a list of rows and a list of columns. Each table cell is an element in the row list and an element in the column simultaneously. If a table cell is deleted, then both the row list and the column list have to be updated.

Biggerstaff et al. (1984) implemented a table composition system called TABLE. Before the system was implemented, the effectiveness of the tree representation and the dual hierarchy representation were compared. The authors found that each of these representations had its advantages and disadvantages, and operations that were efficient in one representation were often inefficient in another. For example, operations performing on spanning rows and columns were easier under the tree representation, and operations involving keeping track of cursor movements within tables were easier to be implemented under the dual hierarchy representation. Overall, the dual hierarchy representation was used because it had greater symmetry between row operations and column operations.

2.2.2.5 The CALS Model

The CALS table model, put forward by the Organization for the Advancement of Structured Information Standards (OASIS), is a standard for representing tables in SGML/XML. Designed to handle a variety of complex tables in military technical documents, the CALS table model allows tables' geometric information and certain basic formatting attributes, such as cell alignment, borders and table cell orientations, to be encoded. Tables in the CALS model are defined as XML strings that meet the CALS DTD (see OASIS (1996, 1995, 1999)). There are many details contained within the CALS DTD. However, structurally, the CALS table model specifies that a table is made up of an optional title plus one or more TGROUPs. A TGROUP has its own body (TBODY) and its own independent set of optional column headings (THEAD) and footings (TFOOT).

One of the characteristics of the CALS model is that it allows authors to attach name strings to columns. Instead of referring to columns by their indices, such as column 1 and column 2, columns can be referenced by some strings, such as 'the province column' and 'the enrolment information column' in Figure 2.14. Although strings of characters can be attached to columns, the CALS model does not provide semantic interpretation of the naming strings. Under the CALS model, the logical structure of Figure 2.10 is illustrated by Figure 2.11.

Year	Number of Students Enrolled
1993/94	757918
1994/95	741630
1995/96	729116
1996/97	714079
1997/98	703293
Total	3646036

Figure 2.10: Canada University Undergraduate Enrolment Information Grouped by Year. Source: Statistics Canada Catalogue No. 81-003-XPB, Data release Table1 (1993/94 - 1997/98).

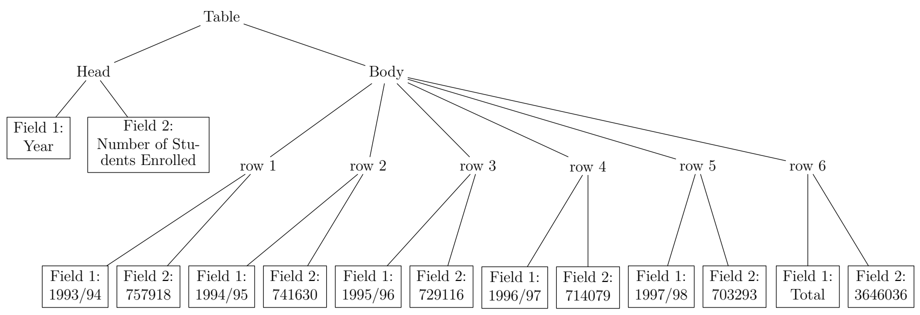


Figure 2.11: The logical structure of Figure 2.10 under the CALS model

2.2.3 Tables at the Physical Level

At the physical level, tables are seldom formally defined. However, the physical realisation of tables can occur in a number of formats. These include signals in electronic files, pixels on screens, printed characters on paper, or voice signals. In practice, the rectangular column-row structures that physical tables are built upon are widely accepted as tables. Before a table extraction system is evaluated, expected answers, which specify the locations of tables in a document, should be decided upon based on a formal definition. However, in the absence of a formal definition at the physical level, expected answers are often defined by annotators using their intuitive judgements (S. H. Lin & Ho, 2002; Ng et al., 1999). This works well under normal circumstance, as Hu, Kashi, Lopresti, and Wilfong (2000a) find that annotators could agree on the expected answers in most cases. However, several documents in their test set, which consists of 25 documents, caused disagreement among the annotators. In order to reduce ambiguities and disagreements with expected answers when identifying tables, Chapter 4 provides guidelines for table identification for the experiments covered in this dissertation.

2.2.4 Table Serialisation

This section surveys the table formats in external storage. A table existing in computer memory will have to be stored in external storage such as files. As Figure 2.17 shows, the serialisation step encodes logical tables and stores the encoded information in external storage. The most popular way to encode tables is to use the XML markup scheme. This is because the XML markup scheme allows users to encode a table's geometric structure as well as the functions of the entities in a table (Zanibbi et al., 2004). For example, the information that a table cell is a header cell can be recorded as an attribute in a XML open tag. An additional benefit of using the XML encoding scheme is that an XML description can be transformed into an appropriate HTML version or to other formats that are suitable for higher level processing (Hurst, 2003).

Hurst (2003) work attempts to derive the structure of a table from a flat, textual representation (that is, tables are treated as streams of characters in plain text documents). He used the following scheme to encode table cells: each cell is surrounded by a pair of ⟨CELL⟩ tags, each cell uses two pairs of coordinates, (X_0, Y_0) and (X_1, Y_1), to record its location, and the cell content is recorded at the end of the open tag and before the close tag. Using this method, the first row in Figure 2.1 would be encoded as follows.

```
<CELL X0="0" Y0="0" X1="0" Y1="3"> Year </CELL>
<CELL X0="1" Y0="0" X1="1" Y1="3"> Term </CELL>
<CELL X0="2" Y0="0" X1="7" Y1="0"> Mark </CELL>
<CELL X0="2" Y0="1" X1="4" Y1="1"> Assignments </CELL>
<CELL X0="5" Y0="1" X1="6" Y1="1"> Examinations </CELL>
<CELL X0="7" Y0="1" X1="7" Y1="2"> Grade </CELL>
<CELL X0="3" Y0="3" X1="3" Y1="3"> Ass1 </CELL>
<CELL X0="4" Y0="3" X1="4" Y1="3"> Ass2 </CELL>
<CELL X0="5" Y0="3" X1="5" Y1="3"> Ass3 </CELL>
<CELL X0="6" Y0="3" X1="4" Y1="6"> Midterm </CELL>
<CELL X0="7" Y0="3" X1="4" Y1="7"> Final </CELL>
```

In TINTIN, a question answering system based on table information, each table line is tagged by a matched pair of ⟨TABLE⟩ tags, and both column headings and table captions are tagged using the ⟨CAPTION⟩ tags (Pyreddy & Croft, 1997). Using this method, Figure 2.10 would be encoded in the same way as Figure 2.12. Obviously, it would be hard to differentiate table headings and table captions under this tagging method.

```
<CAPTION>   Year   Number of Students Enrolled   </CAPTION>
<TABLE>   1993/94   757918   </TABLE>
<TABLE>   1994/95   741630   </TABLE>
<TABLE>   1995/96   729116   </TABLE>
<TABLE>   1996/97   714079   </TABLE>
<TABLE>   1997/98   703293   </TABLE>
<TABLE>   Total   3646036   </TABLE>
<CAPTION>
    Canada University Undergraduate Enrolment Information
    Grouped by Year. Source: Statistics Canada Catalogue
    No. 81-003-XPB, Data release Table1 (1993/94 - 1997/98).
</CAPTION>
```

Figure 2.12: Table representation for Figure 2.10 using the method discussed in TINTIN (Pyreddy & Croft, 1997).

The HTML table markup scheme, which is specified by the World Wide Web Consortium (W3C), provides an alternative for table encoding. This table encoding scheme is commonly used in HTML documents. The formal specification of the HTML table model can be found at Consortium (1999a, 1999b). Figure 2.13 shows the HTML encoding for Figure 2.10.

In Latex , tables are strings of characters placed within the tabular environment. The strings that represent tables must use the double back slash characters as the row boundary, and within the same row the & character is used to separate cells. Details and sample usage of this tabular environment are provided by Lamport (1994). Figure 2.10 is coded in Latex as follows.

```
\begin{tabular}{|p{3cm}|p{3cm}|}
    \hline
    Year & Number of Students Enrolled \\ \hline
    1993/94 & 757918 \\ \hline
    1994/95 & 741630 \\ \hline
    1995/96 & 729116 \\ \hline
    1996/97 & 714079 \\ \hline
    1997/98 & 703293 \\ \hline
    Total & 3646036 \\ \hline
\end{tabular}
```

There are other legacy table encoding schemes, but they are less frequently used. For example, in Unix systems, TBL tables can be compiled to Troff files using the Tbl program. The Troff files are the intermediate files, and they can be used to create tables in various formats (such as postscript and ASCII tables). The TBL definition for tables in BNF (Backus Naur Form) notation is discussed by Freiburghouse (1974), and examples featuring the use of TBL are provided by Cherry and Lesk (January 1979).

The various table encoding schemes discussed in this section show that there are many ways to serialise a table, and a table's two-dimensional layout structure can be encoded in streams of characters. The common characteristic of these encoding schemes is that they use pre-defined characters as delimiters to indicate the boundaries of rows and columns, and hence cells. It is worth noting that every table model at the logical level can be serialised. This dissertation uses the directed graph model to represent tables, and tables are serialised using the XML markup scheme that is discussed in

```
<Table>
    <TH>
      <TD>Year</TD>
      <TD>Number of Students Enrolled</TD>
    </TH>
    <TR>
      <TD>1993/94</TD>
      <TD>757918</TD>
    </TR>
    <TR>
      <TD>1994/95</TD>
      <TD>741630</TD>
    </TR>
    <TR>
      <TD>1995/96</TD>
      <TD>729116</TD>
    </TR>
    <TR>
      <TD>1996/97</TD>
      <TD>714079</TD>
    </TR>
    <TR>
      <TD>1997/98</TD>
      <TD>703293</TD>
    </TR>
    <TR>
      <TD>1997/98</TD>
      <TD>703293</TD>
    </TR>
    <TR>
      <TD>Total</TD>
      <TD>3646036</TD>
    </TR>
</Table>
```

Figure 2.13: HTML encoding for Figure 2.10

Section 4.4.3.

2.3 The Dimensions of Tables

Very often, tabular data can be viewed from different perspectives by grouping the data under different categories. For example, Figures 2.10, 2.14 and 2.15 are three different views of the Canadian undergraduate enrolment data: Figure 2.10 presents the data in terms of years, Figure 2.14 groups the data by province, and Figure 2.15 groups the data based on student status. Depending on the intent of the author, the data could also be grouped under other categories such as faculty, or the gender, age and place of origin of the students. The number of categories that tabular data can be grouped by is called the dimension of the table (see X. Wang (1996)). From the logical point of view, tables are multi-dimensional objects, however they have to be presented as two-dimensional objects in documents. In order to present multi-dimensional tables in two-dimensions, some labels of the categories have to be repeated. Figure 2.16 groups the student enrolment information in three categories (year, province and student status). Hence, it is a three-dimensional table at the logical level. This three-dimensional table has repeating category labels (the names of the provinces) when it is presented in documents.

At the abstract level, because a table is a collection of entries that are semantically connected to multiple labels of different categories, abstract tables are multi-dimensional (X. Wang & Wood, 1993b). When abstract tables are realised as physical tables, the tables become two-dimensional. However, encoded tables [1] that are resulted from the serialisation step are one-dimensional, because they are streams of characters stored in strings. Because of their one-dimensional nature, encoded tables cannot treat operations on rows and columns symmetrically. For example, for the encoding scheme in Section 2.2.4, deleting a row involves deleting a line from the representations, but deleting a column involves removing one entry from every line.

[1]The concept of 'encoded tables' in this chapter is equivalent to the 'logical table' concept in X. Wang and Wood (1993b)

Province	Number of Students Enrolled
British Columbia	299801
Alberta	303264
Saskatchewan	141654
Manitoba	152030
Ontario	1393891
Quebec	985466
New Brunswick	112491
Nova Scotia	167547
Prince Edward Island	15252
Newfoundland	74640
Total	3646036

Figure 2.14: Canada University Undergraduate Enrolment Information Grouped by Province. Source: Statistics Canada Catalogue No. 81-003-XPB, Data release Table1 (1993/94 - 1997/98).

Student Status	Number of Students Enrolled
Full-time	2500837
Part-time	1145199
Total	3646036

Figure 2.15: Canada University Undergraduate Enrolment Information Grouped by Student Status (full-time or part-time). Source: Statistics Canada Catalogue No. 81-003-XPB, Data release Table1 (1993/94 - 1997/98).

		1993/94	1994/95	1995/96	1996/97	1997/98
Full-time	British Columbia	36195	37802	39545	40276	41724
	Alberta	44872	44702	46342	48875	48098
	Saskatchewan	21415	21319	21780	21973	22184
	Manitoba	17587	20308	18882	21857	21013
	Ontario	203624	202697	200560	199957	200482
	Quebec	115130	112953	110635	108098	107024
	New Brunswick	18434	18454	18399	17918	17344
	Nova Scotia	27483	27534	27442	27700	28850
	Prince Edward Island	2669	2519	2400	2450	2430
	Newfoundland	12146	12211	12203	12242	12100
Part-time	British Columbia	19446	19850	20129	22528	22306
	Alberta	15247	14158	13146	13125	14699
	Saskatchewan	7678	6799	6740	6076	5690
	Manitoba	15513	11603	10826	7454	6987
	Ontario	87531	82788	80218	70553	65481
	Quebec	97110	91429	86894	80819	75374
	New Brunswick	4850	4583	4745	4229	3535
	Nova Scotia	6343	5801	5539	5294	5561
	Prince Edward Island	771	581	470	480	482
	Newfoundland	3874	3539	2221	2175	1929

Figure 2.16: A 3D View of the Canada University Undergraduate Enrolment Information. Source: Statistics Canada Catalogue No. 81-003-XPB, Data release Table1 (1993/94 - 1997/98).

2.4 Table Processing

The process of transferring a table at its abstract level to its logical level is called *table composition*, and the process of transferring a table at its logical level to its physical level is called *table rendering*. In contrast to the table composition and table rendering processes, which start from the abstract level and work their way down to the physical level, there are processes that take documents as the input and work their way up to discover tables at their abstract level. There are two tasks in the natural progression of table recovery: table extraction and table interpretation.

The table extraction task discovers the physical attributes of tables, and the approach is medium dependent. Although different authors used different terminology, it has been generally agreed that there are two sub-tasks in table detection: table boundary identification, and table structure recognition. Table boundary identification is about discovering the locations of tables: it answers the questions of how many tables are in a document, and where the boundaries are for each of the tables. The table structure recognition step recovers the column-row structure of a table: it answers the questions of how many columns and rows are in a table, how many cells are in each column/row, and what the boundary is for each of the cells that form the column-row structure for a table. Examples of these tasks are discovering table titles, separating header cells from data cells, and finding merged cells within tables (Hu et al., 2000a; Ng et al., 1999; Y. Wang, Phillips, & Haralick, 2001).

Once a table's location and structure are identified, table interpretation can then take place. The ultimate goal of table interpretation is to recover the intent of the authors (hence the abstract tables) from tables at logical levels. Very often, table interpretation involves building semantic representations for the tables and working out the relationships between table components. This eventually leads to the understanding of the data in tables. Figure 2.17 illustrates the table processing tasks involved in translating tables from one level to the other. This section reviews each of these areas.

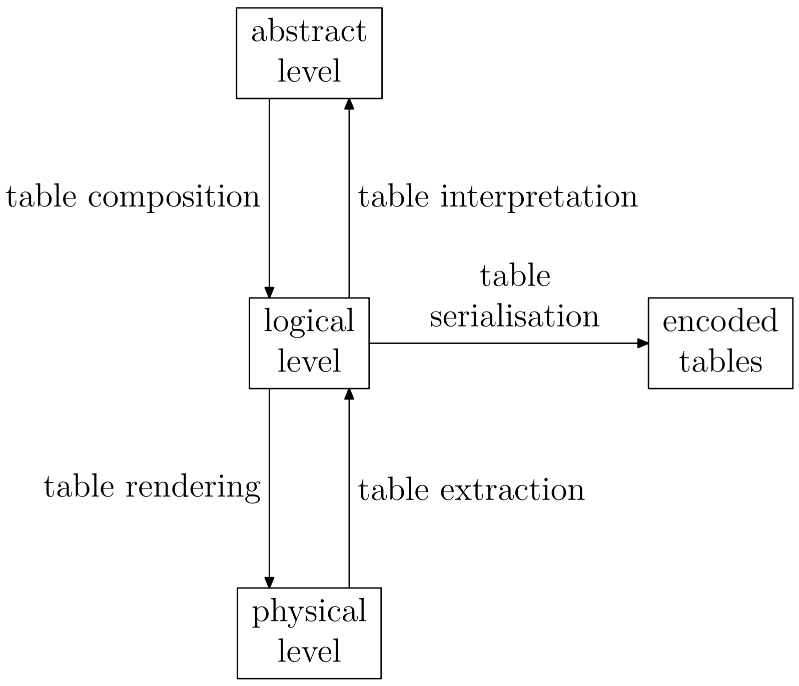


Figure 2.17: Table processing tasks

2.4.1 Table Composition and Table Rendering

Tables in plain text documents do not contain formatting attributes, and they can be created either manually or using software tools such as Groff (*The GNU Troff project*, n.d.). Modern document composition languages, such as Latex and HTML (and older ones such as TROFF and TBL) allow users to typeset tables using pre-defined tags. Users only need to describe the logical structures of the tables, and the systems can automatically render the tables based on the users' instructions. The separation of the logical and layout structures is widely used in table composition systems because it empowers authors to focus on *what* tables to create rather than *how* to create tables. In terms of the level of inputs, table composition systems can be classified into three groups: systems that take abstract tables as inputs, systems that accept logical tables as inputs, and systems that provide visual environment allowing users to create and edit physical tables directly. Examples of the third group of systems are Biggerstaff et al. (1984), Microsoft Word, and spreadsheets such as Lotus 1-2-3, Microsoft Excel and Open Office. The following two paragraphs provide more information about the first two groups of table composition systems.

The most representative work in the first group is the table composition system called X-Table, which was created by X. Wang (1996). X-Table renders tables based on the functional descriptions of tables (that is, tables at the abstract level). Because tables at the abstract level can be multi-dimensional and do not provide a geometric arrangement of table cells, systems in this group have to work out the order of the columns and rows as well as the size of each column and row (Embley et al., 2006; X. Wang, 1996). One way to create a physical table from an abstract table is to factor the labels as column headers, and allow each row to contain the corresponding value (Embley et al., 2006). When rendering a physical table, there are two main sub-problems: how to determine the size of each column subject to the constraint of the space available in the document, and how to determine the order of the columns so that the related materials are close together. The first sub-problem was proven to be a NP-complete problem, which could be solved by exponential-time algorithms, or by polynomial-time heuristics (X. Wang, 1996). The second sub-problem was attempted by Losee (2003), who used Gray code methods to minimise the semantic distances between columns.

For systems that accept logical tables as inputs, inputs are two-dimensional logical tables that contain the geometric arrangement of table cells capturing the intent of authors. Logical tables are expressed in high level languages with the option of specifying the formatting attributes of the table components. These systems are found in a range of sources (see, for example, Cherry and Lesk (January 1979); Lamport (1994); Quint and Vatton (1986, 1996); Reid (1980); Consortium (1999a, 1999b)). Tables are generated based on table descriptions commands and style specifications. It is worth noting that for the same set of description commands and the same formatting specification, a table can be rendered differently in different media. For example, the table generation subsystem in Latex would take the sizes and orientations of target documents into account when tables are produced.

2.4.2 Table Extraction

Table extraction is a general term for recovering tables from input media. There are a variety of tasks that fall under this table recovery category: table line identification, table boundary identification, cell segmentation (which leads to the identification of column-row structures, the titles, headers, spanned cells, and multi-line cells), and the association of headers and their corresponding data cells.

Plain text documents, markup documents, and document images could all become a source of inputs for table extraction systems. Both plain text documents and markup documents consist of a sequence of ASCII characters, and therefore they can be processed directly. Plain text documents do not support any type of text formatting (such as bold, italics, underlining, or font sizes), and as a result, the characters are displayed without further processing. Markup documents, such as HTML documents, on the other hand contain tags (sequences of characters that are reserved for special meanings), and they must be interpreted by some rendering tools before they are displayed. Table images printed on paper or on electronic devices must be scanned before being analysed.

2.4.2.1 Generalised Table Extraction

Table recovery is a challenging task for two main reasons. Firstly, because tables are so diverse, it is very difficult to identify the common characteristics for table recovery (Hu, Kashi, Lopresti, & Wilfong, 2001a; Hu, Kashi, Lopresti, Nagy, & Wilfong, 2001; Lopresti & Nagy, 1999a, 1999b; Ramel et al., 2003). Tables could contain auxiliary elements such as titles and footnotes, or they could contain spanning cells and/or multi-line cells, or they could provide explicit ruling lines to indicate the column-row structures. For those tables that use explicit ruling lines, the ruling lines could be incomplete and/or uneven, and the irregular spacing between columns causes table cells to have different sizes and not to be perfectly aligned (Douglas & Hurst, 1996; Hurst, 1999b, 2000, 2001a, 2001b; Hurst & Nasukawa, 2000; Hurst & Douglas, 1997b; Ramel et al., 2003). Most table extraction challenges are due to the layout complexity of tables. Take table titles as an example; for those tables that contain titles, some have titles above the table body, while others have them below the table body. Often, table titles are just one line long, although they can also occupy multiple lines. The same problem exists for table headers. Some tables have only column headers, some have only row headers, and some have both. Table headers can be short and non-nesting, or they can be long, span across multiple cells, or they can contain sub-tables. Secondly, there could be human errors in producing tables. Tables may be incomplete, poorly composed, or contain erroneous headers and entries. It is very difficult to recover tables if the inputs contain errors (Lopresti & Nagy, 1999a).

The characteristics of tables are often expressed as measurable information called *features* or *attributes*. For example, 'the percentage of the data cells in a column that share at least one common word' could be used as a feature for differentiating table regions from non-table regions. Very often, more than one feature is used to represent the characteristics of a group. A group of feature values can be written down in a fixed order, and it then becomes a *feature vector*. In table processing, two types of features have been identified: layout features (which are medium dependent, and used to detect physical tables) and language features (which are medium independent, and used to detect logical tables) (Hurst, 2001b). Very often, language features are specific to an environment. For example, when detecting table headings, if the word *change* appears in a cell, and if two or more cells below it contain numeric data, then the word *change* is likely to be a table heading (Tengli, Yang, & Ma, 2004).

Searching for tables often involves making binary decisions based on some feature measurements. For example, given some string segments in documents, are they length consistent? A more complex example would be: given some features of a block of lines, is the block a table? There are two types of methods used to answer these types of questions: non-adaptive methods (also referred to as *deterministic approach*) draw conclusions based on fixed thresholds; and the adaptive methods (also referred to as *statistical approach* or *machine learning approach*) decide the thresholds based on some (training) data drawn from the population, and make decisions using the dynamically calculated thresholds.

Due to their simple nature, non-adaptive methods for deciding pre-defined, fixed thresholds are widely used to measure string cohesion levels (Hurst & Douglas, 1997a), to judge the existence of header columns and rows, to find identical table cells (Tengli et al., 2004), and to judge the existence of tables (H.-H. Chen, Tsai, & Tsai, 2000; Hurst & Douglas, 1997b; Y. Wang et al., 2001). For example, if at least 50% of the data cells in a column/row are identified as headers, then the column is considered as a header column/row (Tengli et al., 2004).

Other non-adaptive methods include the matches of templates and grammar. For example, Yoshida et al. (2001) observe that the majority of table structures fall into nine categories, and table headers are mainly placed in the first row (in which case, tables are read in the top-down order) or first column (in which case, tables are read left to right). Layout information, together with other information such as cell types and cell relationships, can be encoded in templates for recognizing new tables: tables are detected if templates are matched in the input documents (E. Green & Krishnamoorthy, 1995a, 1995b, 1995c; E. A. Green, May 1996; Hurst & Douglas, 1997a; Watanabe, Luo., & Sugie, 1993; Watanabe, Luo, & Sugie, 1995; Yoshida et al., 2001). Similar to templates which store static information (information that does not change during run time), (context-free) grammars capture the regularities of information by parsing documents. In the literature, grammar rules are used frequently (Amano & Asada, 2002, 2003; E. Green & Krishnamoorthy, 1995a, 1995c; Komfeld & Wattecamps, 1998; Long, Dale, & Cassidy, 2005; Rahgozar & Cooperman, 1996).

Unlike deterministic algorithms, which use pre-defined thresholds, machine learning approaches use adaptive methods to find the most optimal thresholds based on some training data. During the learning phase, the algorithms examine both accepted and rejected samples, and learn to recognise their differences. During the experimental phase, the algorithms accept or reject a hypothesis depending on whether it has features that are more similar to the accepted samples or to the rejected ones. Machine learning algorithms are widely used in natural language processing. The table processing literature shows the use of several machine learning techniques.

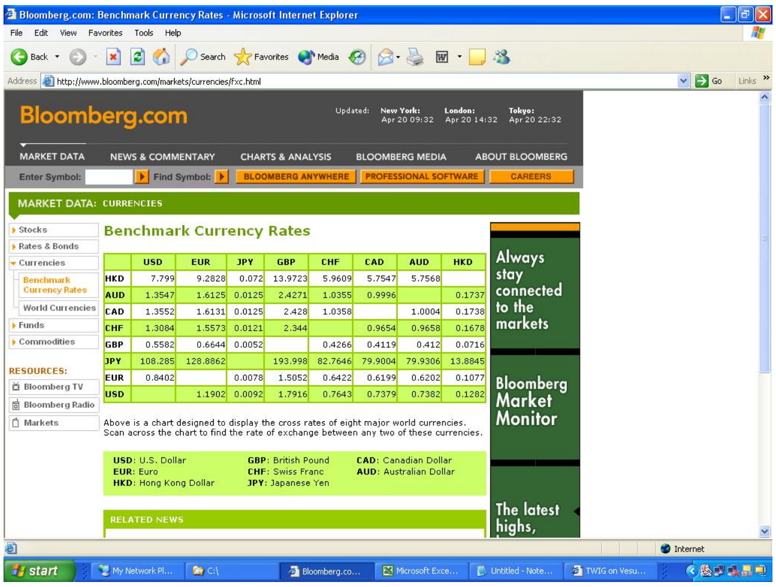
For example, neural networks are used to recognise table boundaries, table columns and rows (Ng et al., 1999), and support vector machines (SVM) are used to differentiate ⟨TABLE⟩ tags that are used to format text from ⟨TABLE⟩ tags that are used for genuine tables in HTML documents (Y. Wang & Hu, 2002b). Other machine learning techniques include: k-nearest neighbour (K-NN) approaches to identify keywords that are frequently used in tables (Y. Wang & Hu, 2002a) and to recognise table columns, rows and cells (Hu, Kashi, Lopresti, & Wilfong, 2000b, 2001b; Hu et al., 2002); information entropy value to find unique content embedded in table tags (S. H. Lin & Ho, 2002); and maximum entropy models, as well as decision trees, to recognise table boundaries, table columns and rows (Ng et al., 1999), and to differentiate ⟨TABLE⟩ tags used for formatting purposes from ⟨TABLE⟩ tags used for genuine tables in HTML documents (Y. Wang & Hu, 2002a, 2002b).

While feature selection has been recognised as an important factor in determining accuracies, it has been reported that no one learning algorithm clearly outperforms others (Ng et al., 1999). Comprehensive descriptions of machine learning theory and its applications to natural language processing are covered by many authors (see, for example, Mitchell (1997); Michalski, Bratko, and Kubat (1998); Manning and Schutze (1999)). It has been found that, for the same table processing task, learning algorithms outperform deterministic algorithms by about 10%. For example, the accuracy of detecting table boundaries using deterministic algorithms is about 70%, whereas the C4.5 decision tree algorithm and neural networks algorithm can achieve accuracies of between 85% and 95% for the same task (Ng et al., 1999).

2.4.2.2 Extracting Tables from HTML Documents

In HTML documents, tables are mainly created by using matched pairs of ⟨TABLE⟩ tags, although it is also possible to include pre-formatted tables within the ⟨PRE⟩ tags. Inside a pair of ⟨TABLE⟩ tags, table rows and table cells are marked by the matched pairs of ⟨TR⟩ and ⟨TD⟩ tags.

Table tags are widely used in HTML documents: a website search engine in Taiwan reported that almost 70% of web pages contain ⟨TABLE⟩ tags, and there were between 3 to 4.42 ⟨TABLE⟩ tags per page (S. H. Lin & Ho, 2002). Compared to extracting tables from plain text documents, extracting HTML tables might seem a bit easier because of the hint from the ⟨TABLE⟩ markup tags. However, the presence of the table tags does not guarantee genuine tables because table tags are widely used for formatting purposes (H.-H. Chen et al., 2000; Hurst, 2001b; Lopresti & Nagy, 1999a; Y. Wang & Hu, 2002b, 2002a; Tengli et al., 2004). The low percentage of ⟨TABLE⟩ tags being used for creating genuine tables was noted by a number of independent researchers. Y. Wang and Hu (2002b) examined a collection of 1,393 HTML documents from hundreds of different websites spanning across different industries, and they found that only 1,740 (15.16%) of 11,477 ⟨TABLE⟩ tags were used for genuine tables. A survey of 1372 HTML documents in the Asian airline industry showed that about a third of the table tags were used for genuine tables. On average, 2.35 ⟨TABLE⟩ tags were used per document, but only 0.67 ⟨TABLE⟩ tags were used for genuine tables (H.-H. Chen et al., 2000).



Benchmark Currency Rates

	USD	EUR	JPY	GBP	CHF	CAD	AUD	HKD
HKD	7.799	9.2828	0.072	13.9723	5.9609	5.7547	5.7568	
AUD	1.3547	1.6125	0.0125	2.4271	1.0355	0.9996		0.1737
CAD	1.3552	1.6131	0.0125	2.428	1.0358		1.0004	0.1738
CHF	1.3084	1.5573	0.0121	2.344		0.9654	0.9658	0.1678
GBP	0.5582	0.6644	0.0052		0.4266	0.4119	0.412	0.0716
JPY	108.285	128.8862		193.998	82.7646	79.9004	79.9306	13.8845
EUR	0.8402		0.0078	1.5052	0.6422	0.6199	0.6202	0.1077
USD		1.1902	0.0092	1.7916	0.7643	0.7379	0.7382	0.1282

Above is a chart designed to display the cross rates of eight major world currencies. Scan across the chart to find the rate of exchange between any two of these currencies.

USD: U.S. Dollar	**GBP**: British Pound	**CAD**: Canadian Dollar
EUR: Euro	**CHF**: Swiss Franc	**AUD**: Australian Dollar
HKD: Hong Kong Dollar	**JPY**: Japanese Yen	

Figure 2.18: Tables in HTML documents. This web page contains 33 ⟨TABLE⟩ tags, but only 1 of them is used for a real table. Source: `http://www.bloomberg.com/markets/currencies/fxc.html`. Last accessed on 28 November 2009.

In addition to the problem of ⟨TABLE⟩ tags being used for formatting purposes, missing close tags in HTML documents also create problems (Hurst, 2001b). In an attempt to work out the correct locations of tables from 1,393 HTML documents collected from 200 websites, over 250 files were found to have unmatched table tags (Y. Wang & Hu, 2002b). This causes difficulties in calculating the number of rows and columns in a table.

In their research aimed at extracting HTML tables and presenting tabular data as attribute-pairs, H.-H. Chen et al. (2000) consider a table hypothesis as a formatting table if the number of long strings it contains exceeds a certain threshold. They use this information because their study shows that table cells are much shorter than full sentences outside tables, as over 70% of table cells in a corpus of documents from the airline domain contain less than 10 words. A table hypothesis is also considered to be a formatting table if the numbers of hyper-links, or forms or pictures it contains, exceed certain thresholds. However, a table hypothesis is accepted as a genuine table if the majority of cells in a column or row are of the same data type (such as numbers, percentage, time and date, or names of people, places and organisations), or if the majority of cells in a column or row share some common words or strings (such as 'am', 'pm' or the $ or % sign). The authors conducted three independent experiments on the same test data: 1,372 HTML pages from websites in the Asian airline industry. In their first experiment, they accepted or rejected table hypotheses purely based on the number of common characters in neighbouring cells. This gave them only about 55% accuracy. In their second experiment, they added named entities into the acceptance process, but this still did not improve the accuracy very much (from 55.5% to 55.66%). In their third experiment, a hypothesized table was accepted if the numbers of common characters or named entities in neighbouring cells exceeded certain thresholds, or if the number of numeric characters in a table was above a certain threshold. This acceptance condition improved the result significantly: from about 55.5% to 86.5%. In all three experiments, thresholds are set to some fixed values based on the authors' experiences with the data.

Y. Wang and Hu (2002a, 2002b) use 16 features (7 layout features, 8 content type features and 1 word group feature) that they believe to be capable of providing significant separation between genuine tables and formatting tables. For each table (genuine or not), the following information is recorded.

1. Layout features
   - the average number of columns, rows and their standard deviations.
   - the average overall cell length and its standard deviation.
   - the average cumulative length consistency, which measures the cell length consistency along either row or column directions.

2. The content type features are 7 HTML object types: images, forms, hyperlink, alphabetical characters, numeric characters, empty, other objects in each table, and a content type consistency score.

3. A word group feature that measures which words are more likely to be contained in genuine tables, and which ones are not.

After obtaining the feature vectors for each table (genuine or not) from some training data, Y. Wang and Hu (2002a, 2002b) conducted a number of experiments to test the relevance of the three feature sets as well as the effectiveness of two machine learning methods (the decision tree and the SVM methods). By switching on and off certain feature sets, they were able to produce different feature sets: layout features only; content type features only; layout and content type features; and layout, content type and work group features. Their test data consists of 1,393 HTML pages collected from about 200 websites. There are a total of 14,609 ⟨TABLE⟩ tags in these HTML pages, and 1,740 of these ⟨TABLE⟩ tags are used for genuine tables. Their experiments showed that the layout features alone could deliver about 87.7% accuracy (f-measure), content type feature alone could deliver about 93% accuracy, and layout and content type features together could deliver over 95% accuracy. Their experiments also showed that the decision tree and SVM methods produce compatible results (there are no significant advantages of one over the other) as the decision tree produced 95.88% accuracy and the SVM produced 95.89% accuracy.

Although HTML includes markup for headers, it is not consistently used. Using the features identified by Y. Wang and Hu (2002a, 2002b); Tengli et al. (2004) went further by identifying table headers from data cells after re-implementing a program to filter formatting tables. Their approach was to first build a set of header words from the table headers in the training data. Later, when it came to deciding the table headers, a column (or a row) would be identified as a header column/row if the number of header words it contained exceeded a certain threshold. Once a new header was identified, the set of header words would be expanded by adding the unseen words from the header. The expanded header words would then be used to identify new table headers. Testing on about 250 tables mainly extracted from `http://www.commondataset.org`, they were able to identify about 91% of tables correctly.

S. H. Lin and Ho (2002) observed that most websites presented their web pages with repeating information (such as company logos, navigation panels, copyright and privacy announcements and advertisement banners), and the repeating information was usually enclosed within ⟨TABLE⟩ tags. Using this as the hint, they developed a program, called InfoDiscoverer, to filter this particular type of formatting tables. InfoDiscoverer was able to achieve at least 96% accuracy when tested by 26,518 web pages from 13 websites.

2.4.2.3 Extracting Tables from Plain Text Documents

Tables in plain text documents are formed by aligning cell contents, blank lines, sequences of whitespace characters and punctuation characters appropriately. Unlike HTML tables, in which boundaries are indicated by the ⟨TABLE⟩ tags and table structures are indicated by the ⟨TR⟩ and ⟨TD⟩ tags, tables in plain text documents do not have the same markup hints. Although the structural complexities that are the result of the variety of table formats (see discussion in Section 2.4.2.1) are not unique to plain text documents, detecting tables from plain text documents is more challenging due to the lack of hints from markup tags. For example, there is a variety of characters that a ruling line can be built upon. Although characters such as space, tab, dot(.), asterisk(*), dash (‘−’), and equal sign (=) are frequently used, there are no universal rules on what characters to use, and tables could use one or more types of characters as a separator (Ng et al., 1999). The wide range of layout formats adds additional complexities to the table recognition task.

Long tables are often split into parts to accommodate page size restrictions; Semantic analysis is required to check whether tables are split (Lopresti & Nagy, 1999a). In contrast to the long table case, tables could be sparse and cause non-alignment between cells. The irregular character strings and extra whitespace in tables could fool algorithms into thinking that there are multiple tables when in fact there is only one table (Hirayama, 1995). A more common example is the existence of multi-line cells, spanned cells and embedded tables in tables. The presence of data items that do not fit on a single line of a cell complicates the analysis of both paper and electronic tables. In such a case, it is necessary to distinguish line-wrapped data from multiple rows of cells (Lopresti & Nagy, 1999a). Spanned cells provide flexibility for users to design tables, but they make automatic table interpretation more challenging (H.-H. Chen et al., 2000). The presence of sequences of whitespace characters does not guarantee the presence of a table: in an extreme case, two-column articles use sequences of whitespace characters to separate columns.

Although the presence of consecutive sequences of whitespace characters does not guarantee that a table is present, finding tables in plain text documents still relies heavily on the horizontal and vertical alignment of whitespace characters. Once a table hypothesis is identified and the table structure is abstracted out, the acceptance / rejection criteria for plain text documents is very similar to the one for HTML documents. For example, H.-H. Chen et al. (2000) accept/reject table hypotheses based on string similarity, numeric character similarity and named entity similarity in neighbouring cells. Similarly, Hurst and Douglas (1997b) accept/reject table hypotheses based primarily on two measures: the ratio between the number of alphabetic characters and the number of numeric characters (the alpha-numeric ratio), and the ratio of differences between string lengths and the total string length (the string length ratio).

The problem of table extraction can be broken down into three subproblems: recognising table boundaries, column boundaries and row boundaries (see Ng et al. (1999)). In their table identification works, they treat each of these sub-problems as a classification problem. For each sub-problem, they identify a set of layout features from their training data, construct a decision tree based on the features, and apply the decision tree to classify their test data. To solve the table boundary identification sub-problem, for example, they measure 9 attributes including the number of leading space characters, the first non-space character and the number of two or more contiguous space characters in a text line. For each text line, a vector of 27 features representing the attributes from the immediately preceding line, the current line, and the immediately following line is constructed. The vectors are then used to identify table regions, which would have some recurring layout patterns.

Ng et al. (1999) observe that column boundaries are characterised by transitions from vertically connected space or punctuation characters to vertically connected alphanumeric characters. Thus, they use features that reflect the transition of character types across vertical lines to give an indication of column boundaries. The feature set consists of six features: the first three are derived from comparing the immediately preceding vertical line and the current vertical line, and the last three are derived from comparing the current line with the following line. In each case, the three features are the percentage of characters with the space-to-space character transitions, the percentage of characters with the non-space-character-to-space-character transitions, and the percentage of characters with the space-character-to-non-space-character transitions. Their features for detecting row boundaries reflect their observation that rows tend to record repetitive or similar data. If two lines are similar, then they belong to two separate rows; otherwise, they belong to the same row. Similarity is measured by character type transitions. Four features are used for measuring character type transitions: the percentages of space-to-space transition, non-space-to-space transition, space-to-non-space transitions between two adjacent characters, and the ratio between the index of the first non-space character and the table width (in most cases, this is the index of the last non-space character).

Similar to Y. Wang and Hu (2002b)'s work where the effectiveness of different learning algorithms were compared, Ng et al. (1999) also compared the effectiveness of two machine learning algorithms (the C4.5 decision tree and the backpropagation algorithm). They used a deterministic algorithm on a set of 100 Wall Street Journal test data. Their findings were that there was not a significant difference between the two machine learning algorithms. This matched the conclusion that can be drawn the experiment in Y. Wang and Hu (2002b). However, there is a significant difference between using the machine learning algorithm and the deterministic algorithm: the machine learning algorithms deliver 85% - 95% accuracies, whereas the deterministic algorithm delivers about 70% accuracy.

Independent of Ng et al.'s work, Pinto et al. (2003); Tubbs and Embley (2002) also developed programs to locate table boundaries, to identify column-row structures and to associate data cells with their headers in plain text documents. In their work, information used in identifying cell functions and inter-cell relationships can be divided into two categories: geometric layout information and language information. Geometric layout information refers to the physical locations of certain components. For example, table titles are usually placed in the areas above or below the table body (the column/row structure), table headers are usually found in the first row and first column, and data cells are usually located in the following columns/rows. Language information refers to the choice of words, word types and the number words in table cells. For example, it has been observed that the number of words in table titles is usually greater than the number of words contained in table headers, and the number of alphabet characters in table headers is usually greater than the number of alphabet characters in data cells. Data cells usually contain numeric data, and for those data cells that belong to the same column/row, they are usually similar in length and are of the same data type Hu et al. (2000b); Hu, Kashi, Lopresti, and Wilfong (2001b); Hu et al. (2002); Pinto et al. (2003); Wei, Croft, and McCallum (2006). When searching for table headers, Pinto et al. (2003); Wei et al. (2006) observed that if the first column/row of a table contained certain keywords, such as 'Month', 'Year', and 'Name', then the column/row is a header column/row. In general, layout information has wider applicabilities, whereas language information is more specific to an environment or to a set of data.

Pinto et al. (2003); Wei et al. (2006) also compared the effectiveness of different learning algorithms on cell functions, and their findings were somewhat different from the findings by Ng et al. (1999), who did not find much difference between learning algorithms (more specifically, the decision tree and the backpropagation algorithm), whereas Pinto et al. (2003); Wei et al. (2006) reported a significant difference between using the conditional random fields (CRF) algorithm and the Hindden Markov Models (HMM). Pinto et al. (2003) tested sixty-two plain text documents extracted from `http://www.FedStats.gov`, and found that the HMM algorithm could only achieve about 65% accuracy in identifying table lines, the Maximum Entropy algorithm could achieve about 89% accuracy (which is compatible with the results by Ng et al. (1999)), and CRF could achieve about 91.5% accuracy. It is worth noting that the features used by Ng et al. (1999) were purely layout features, and almost all of the features (except for one linguistic feature that measured the number of common strings for identifying table headers) used by Pinto were also layout features.

In the development of the TINTIN system, Pyreddy and Croft (1997) identified tables by checking for whitespace alignment in blocks of contiguous lines of text. After testing a set of fifty tables (out of 6,509, or 0.77%, tables in their database) extracted from 100 documents, the reported result was that 987 out of 1,005 table lines were extracted correctly. There were only 18 false positive (type I) errors and a 54 false negative (type II) errors. Given that the system only used layout features to identify table lines, the reported performance seemed very satisfactory. However, the reported result might be biased for two reasons: first, the test sample is very small - only 50 tables in 100 documents are used for a 6-year Wall Street Journal database. It is not clear how the test data were sampled. Second, the high performance might be due to the evaluation method used in their experiment, because the paper reports that the errors are spread out widely and there is no table missed entirely. This suggests that the errors occur in many tables, but each table has only a small number of errors. If precision and recall rates are calculated at the table level, then the performance was expected to drop dramatically. Having said this, the reported result is interesting, because it provides an insight into how systems perform by only using layout features to detect tables.

Hu et al. (2000a) treat table detection as an optimisation problem. For an input document, they first partition the document into blocks with similar layout structures, then for each block they calculate a score for it to be a table. A scoring function will assign a score to each block to reflect its layout consistency. With all these scores, it is possible to use a global optimisation strategy to find a set of tables that result in the maximum total score. When deciding how many blocks a document has, they use a greedy algorithm. For each block, if a higher score can be achieved by expanding the table upward to include the previous line, or downward to include the following line, they would do so. To decide whether a block should be expanded to cover a new line, they measured the sum of the weighted similarity between the new line and each line in the table. The similarity score for any two lines is calculated as the total number of vertically aligned whitespace characters (excluding the leading and trailing whitespace alignments) minus the total number of whitespace and non-whitespace alignments shared between the two lines. The weight is a function of the distance between two lines: the weight decreases as the distance increases. When this algorithm was tested using 25 ASCII documents, it achieved 88% accuracy.

To summarise, both layout and language features are used in detecting tables, and the following features are commonly used when looking for tables and table structures in plain text documents.

Layout features used in table detection

- The number of space and punctuation characters. This feature is used because sequences of space or punctuation characters are often used to delineate table and table cell boundaries in text documents (Laurentini & Viada, 1992; Ng et al., 1999; Nielson & Barrett, 2003; Pinto et al., 2003; Ramel et al., 2003; Tubbs & Embley, 2002; Wei et al., 2006).
- The number of leading space characters. This feature is used because the number of leading space characters tend to be constant, and this is one of the recurring characteristics tables often have (Ng et al., 1999; Pinto et al., 2003; Wei et al., 2006).
- The number of vertically aligned space and non-space characters. This feature is used because the degree of vertical whitespace / non-space

character alignment is high in tables: a whitespace character is usually vertically aligned with another whitespace character, and a non-whitespace character is usually vertically connected to another non-whitespace character (Hu et al., (2000a, 2000b; 2001a, 2001b; 2002); Ramel et al., 2003; Hurst, 2003; Tubbs & Embley, 2002; Y. Wang, Phillips, & Haralick, 2002).

- The ratio between alphabetical and numeric characters. This feature is used because the ratios are low in tables (Hurst & Douglas, 1997b).
- The ratio of total large vertical blank block areas over table areas. This feature is used because tables often contain horizontal and vertical blank blocks and the areas occupied by whitespace characters in tables are relatively large (Y. Wang et al., 2001).
- The locations where the character types change. This feature is used because table column boundaries are often indicated by the change of alphanumeric characters to space or punctuation characters (Ng et al., 1999).

Language features used in table detection

- The number of cells containing long strings. This feature is used because table cells are much shorter than full sentences outside tables, as study shows that over 70% of table cells contain less than 10 words (H.-H. Chen et al., 2000). This feature is used to accept table hypotheses by H.-H. Chen et al. (2000); Embley et al. (2002); Hu et al. (2000a); Hurst and Douglas (1997b); Ramel et al. (2003); Tengli et al. (2004); Y. Wang and Hu (2002b, 2002a).
- The data type consistency. This feature is used because most columns or rows in tables contain the same data types such as date, time, percentage, people, places and organisation names (H.-H. Chen et al., 2000; Embley et al., 2002; Hu et al., 2000a; Hu, Kashi, Lopresti, & Wilfong, 2001b; Hu et al., 2002, 2000b; Hurst & Douglas, 1997b; Kieninger, 1998; Tengli et al., 2004; Y. Wang & Hu, 2002a, 2002b).
- Whether there are shared words in columns or rows. This feature is used because columns or rows in tables tend to share some common words (H.-H. Chen et al., 2000).

- Whether there are header cells. This feature is used because tables tend to have headers. Header cells differ from data cells in two main ways: geometrically, header cells are usually placed in the first column or in the first row if there are no table titles, or they are placed after table titles and before data rows if there are table titles (Pinto et al., 2003; Wei et al., 2006; Tubbs & Embley, 2002). In terms of content, many researchers find that certain keywords, such as ‘Year’, ‘Name’, month abbreviations, and year strings, are commonly placed in table headers (Embley et al., 2002; Pinto et al., 2003; Wei et al., 2006).

2.4.2.4 Extracting Tables from Document Images

Document images are made up of pixels. Tables in this document type are visual effects created by digitising process such as scanning. The process of finding tables from document images is more complex than finding tables from markup documents or from plain text documents. For example, one of the complexities exists in document images but not in plain text document is the use of irregular fonts. At the very least, table analysis on document images can only be started after converting a pixel representation to a more abstract representation in terms of characters. This process is error prone by itself, because document images, which are commonly obtained from scanning procedures, might contain noise and skewed images.

Skewed images have to be corrected before the horizontal and vertical lines in the original documents can be represented in vectors. A gradient based approach for skew correction was presented by Xi and Lee (1998).

The process of converting a pixel representation to a higher representation format involves a binarisation step and a noise filtering step. Because pixels have different colors, the binarisation step converts each pixel’s colour to either black or white. Noise may be introduced to images by the scanning process or during the binarisation process. The noise filtering step is to remove the noise by adding missing pixels, and removing extra pixels. Extra pixels can be detected if a group of black pixels are surrounded by white pixels, and its size is smaller than a certain threshold (say half of the average width of the characters). Similarly, missing pixels can be detected if a small (say less than half of the average width of characters) group of white pixels is detected, and it is surrounded by some black pixels (Abu-Tarif, 1998).

There are two common approaches for processing tables in document images: to manipulate the source document directly by looking for the regularity in pixel arrangement, or to recover the ASCII text from the source document using OCR tools and deal with the documents as text documents. OCR systems recover the original documents from rendered images. There are two main problems associated with the second approach. Firstly, OCR tools do not have 100% reliabilities, and their errors would be propagated to the table analysis stage. The author of this dissertation notice that most OCR systems convert large space areas, which are often appear between table cells, to a single space character. By doing this, the column-row structures in document images are lost. Secondly, certain typographic information (such as the font and the colour of the text) in the original documents is lost. Modern OCR systems have gone beyond the simple transformation of document images to plain sequences of words by having built-in abilities to recover tables (Kieninger, 1998; Zanibbi et al., 2004). Hundreds of thousands of tables are recovered each day from paper documents which layouts are known in advance. Examples of these documents are telephone bills, medical, insurance, and state income tax forms using OCR technologies (Lopresti & Nagy, 1999a; Peterman, Chang, & Alam, 1997; Shamailian, Baird, & Wood, 1997). However, for documents that contain tables at locations that are unknown, and for tables whose structures are not known (for example, the cells are uniformly spaced), the OCR results are still need to be improved.

Y. Wang et al. (2001) used large horizontal and vertical blank blocks to identify potential tables and table structures. After identifying a table and its structure, they then calculated the consistency score. The scores took into account three elements: the ratio of total blank block areas (both horizontal and vertical blank blocks) over the entire table area, the maximum y-coordinate difference between cells in a row, and the accumulated difference between the justification of all columns. If the score for the consistency is greater than 0.5, then the table hypothesis is accepted. This method of table detection was tested using a total of 1125 machine printed documents, which included 565 real-world documents, and it achieved about 90% accuracy.

This background-analysis-based table identification method was subsequently improved by adding an optimization step to the process. In this step, Y. Wang et al. (2002) treated the table detection problem as a probability optimization problem. Given a table hypothesis, they compared the probabilities of several alternatives: to accept the table hypothesis as a table, to grow the table to include its upper/lower neighbour blocks, and to reject the table hypothesis as a table and to merge it with its upper/lower non-table blocks to form larger non-table blocks. The alternative that gave the biggest improvement upon the initial page segmentation probability would be selected as the result. This process was repeated until there was no further improvement that could be made. The additional step boosted the performance from about 90% to about 95%.

Nielson and Barrett (2003) developed a table structure recognition algorithm based on a document's horizontal and vertical profiles. The horizontal and vertical profiles were obtained by projecting the (non-background) pixels that makes up of a document image onto the x-axis and y-axis respectively. Table boundaries are detected from the 'peaks' of a document's horizontal and vertical profiles, because table boundaries are typically unbroken ruling lines, which generate more projected pixels than other elements could. This detection method worked particularly well for tables that used explicit grids to separate table cells, because the pixels in the grids boosted the value of the projection function. Tested using the 1841 and 1881 British census and 1870 United States census forms, this method delivered extraction results of about 90%.

Hu, Kashi, Lopresti, and Wilfong (2001a) also did an experiment on extracting tables from single-column document images. Instead of applying OCR to the documents, they pre-processed them and extracted only the bounding box for each word using the tool developed by Baird (1992). Hence, their algorithm did not assume any form of delimiter, because after the pre-processing step, the inputs to their system were the geometrical positions of the bounding boxes. Tables were detected based on the amount of whitespace correlation in the document. As an example, column identification was achieved by using a clustering technique: first, each bounding box for a word was classified into a single cluster; and second, two clusters with the minimum inter-cluster distance (defined as the Euclidian distance) were merged into a new cluster. The merging process was repeated until there was only one cluster, and a tree representing the cluster structure was generated along the way. The root of the tree represented a document, and each tree branch represented a cluster in the document. Relying on the information that the spacing between table columns tended to be similar across a table, columns were identified by traversing the cluster tree and finding branches that had similar cluster distances. Testing on 110 pages from the University of Washington I CD-ROM database, Hu, Kashi, Lopresti, and Wilfong (2001a) were able to achieve about 91% accuracy.

Similar to Hu, Kashi, Lopresti, and Wilfong (2001a) 's approach of using word bounding boxes as input, Kieninger (1998) produced a table structure recognition system called T-Recs (Table REcCognition System). Unlike Hu, Kashi, Lopresti, and Wilfong (2001a)'s approach where separator information (ruling lines or spacing) is used to detect tables, T-Recs detects table columns based on a bottom-up word clustering method: initially, an arbitrary bounding box for a word forms a block; later, the algorithm recursively expands the block to cover all bounding boxes that are vertically aligned (fully aligned or partially aligned) neighbours. In the identified columns, a bounding box that does not have an upper or a lower neighbour is identified as a header cell. Lines containing more than one header cell would be used as a hint for table boundaries.

There are special types of document images called table-form documents (also referred to as 'form documents'). These documents attract a reasonable amount of research interest (see, for example, Amano and Asada (2002, 2003); Amano et al. (2004); Belaid and Belaid (1999); J. S. Chen and Tseng (1996); Clark (1987); Duygulu et al. (1998); Duygulu and Atalay (2000); Hori and Doermann (1995); Turolla and Belaid (1996); C. Lin and Hsiao (1998); Watanabe, Luo, and Sugie (1993, 1994); Watanabe et al. (1995); Xi and Lee (1998)). An entire table-form document is a single table. When detecting table-form structures, most systems take advantage of the main characteristic of form documents - fields are rectangular, and they are often enclosed by horizontal and vertical lines called bounding boxes.

Hori and Doermann (1995) presented a different way of analysing the structure of table-form documents: after a table-form document is scanned as a binary image in which black pixels are 1 and white pixels are 0, the contours of objects are extracted from the scanned image. Each contour of an object is represented by a bounding box, and the geometric locations of the boxes are used to analyze the structures of the table-form documents.

When Tubbs and Embley (2002) tried to interpret data from genealogical microfilm table-form documents, they too used existing OCR tools to convert the document images into XML documents. The results of the pre-processing step allowed them to detect table-form document structures using the geometric layout information. Once tables were extracted, they used string matching method to find pre-defined concepts and markup table cells with pre-defined strings.

2.4.3 Table Interpretation

After working out table locations and structures in documents, the next natural progression is to determine the meaning of the contents of tables. In order to achieve *table interpretation*, one fundamental question has to be answered - what counts as table interpretation? Although table interpretation is the ultimate goal of table processing, and it is considered to be the most challenging table processing task (Lopresti & Nagy, 1999a; Zanibbi et al., 2004), currently there is no agreement on what table interpretation should be. Hurst (1999b) proposes that table interpretation is about finding models that can describe what is true about our world based on the information presented in tables. This definition provides an idealistic goal of table interpretation. However, each piece of research is focused on a specialised area in practice. The literature shows that there are many levels and forms of table interpretation.

Interpreting tables by recovering the abstract models of tables
Embley et al. (2006) defines that a table is understood if the sets of labels, domains, and the mapping functions are known.

Interpreting tables by displaying pre-defined concepts
In the work done by Tijerino et al. (2003); H.-H. Chen et al. (2000);

Wong, Martinez, and Cavedon (2009), the objective of table interpretation is to extract pre-defined information types from tables. The pre-defined information includes names of customers, numeric value of customer spending, population of countries and named entities in tables. Machines display their *interpretation* by showing all the pre-defined concepts from a given table.

Interpreting tables by showing the reading order of a table and by showing the attribute-value association

A table could potentially hold a large amount of data, and Guthrie, Weber, and Kimmerly (1993) specifies that accessing individual cells is the most basic form of using tables. To be able to access cells, the orientations and the reading order of a table must be known. H.-H. Chen et al. (2000) used cell similarities to judge the orientation of a table: if there are many similarities between cells in the same column/row, then the reading order for that particular table is likely to be top-down (or left-to-right). From a table's orientation, they were able to decide header rows (or columns) and associate header cells with their corresponding value cells (called attribute-value association). Attribute-value shows an unordered list of unique attributes with their associated values. Layout similarities, such as string lengths, data types and shared words between table cells, are often used to detect attribute-value pairs (H.-H. Chen et al., 2000; E. Green & Krishnamoorthy, 1995a; Hurst, 1999b), although ontological knowledge has also been used (Yoshida et al., 2001). Identifying attribute-value pairs is a fundamental step for many other table interpretation tasks. After identifying attribute-value pairs, H.-H. Chen et al. (2000) re-write tables in a two-column format: the first column being the attribute column, and the second column being the value column. Hurst (1999b) explored judging the domains of tables based on the contents of attribute cells; and Yoshida et al. (2001) used the identified attribute-value pairs to classify and to merge tables.

Interpreting tables by identifying cell functions

Table cells are the basic elements for interpretation, and identifying cell functions is the most popular form of table interpretation found in the literature. Cell function identification is mainly about finding the index columns (or rows), finding table titles, headers (and hence non-headers), and footnotes (Pinto et al., 2003; Tubbs & Embley, 2002; Wei et al., 2006).

Interpreting tables by populating databases and filling in pre-defined templates

Populating databases and filling in pre-defined table-like templates is probably the most practical form of table interpretation. The rationale behind this form of table interpretation is that tables in documents have relational information structures similar to database relationships: a table can be thought of as a set of n-tuples expressing a multi-dimensional relationship among a number of domains of values, mapped onto a two-dimensional representation for columns/rows (Douglas & Hurst, 1996; Hurst, 1999b). More specifically, a header in a table can often be mapped to an attribute name in a database table. Since tables do not usually contain duplicated records, each row in a table can be used as a unique key in a database table (Lopresti & Nagy, 1999a). The most representative work in this area involves filling in pre-defined templates with information from tables (Douglas & Hurst, 1996).

Interpreting tables by categorising tables and by combining related tables and present them in a single table

Combining related tables is an advanced form of table interpretation. By combining related tables, the amount of time and space in storing and searching for data is reduced (Lopresti & Nagy, 1999a; Watanabe et al., 1995). The general approach is to enumerate all attribute-value pairs in tables, eliminate the redundant pairs, and re-render the information in a new table.

Marchionini, Hert, Liddy, and Shneiderman (2000) showed that tables from different sources could be combined and summarised. Based on 14 pre-defined topics (such as child care) they summarised the contents of tables containing statistical data from 200 different websites. The summary information included the data type, and the number of websites containing the data.

After identifying attribute-value pairs in tables, Yoshida et al. (2001) took the work a step further. Based on the attribute-value pairs contained in tables, they grouped related tables together before merging them into tables without repeating information.

Interpreting tables by providing answers to questions based on table contents
TINTIN is a table retrieval system that automatically detects tables from plain text documents. When users query the detected tables using the English language, TINTIN provides answers to the questions based on the content of the tables (Pyreddy & Croft, 1997).

Unlike table extraction and table structure recognition tasks, table interpretation tasks are medium independent. That is, whether a table is extracted from a plain text document, an HTML document, or from a document image has very little impact on the interpretation tasks.

Identifying pre-defined concepts, cell functions, and the reading order of a table can be generalised as establishing the relationships among table components, or between table components and information outside tables. 'Table interpretation' in this dissertation is about identifying such relationships (see Chapter 6 for an example). Through identifying relationships, abstract models of tables can also be found (see Section 6.5).

2.4.4 Overall Architecture

In terms of the overall architecture, the procedure programming model has been widely used, regardless of the sources of inputs. Existing algorithms (adaptive or otherwise) were implemented as modules in the progression of table detection activities. For example, when attempting to extract the attribute-value relationships among table cells from HTML documents, input documents went through five processing steps before tabular data was presented: the hypertext processing step extracted text embedded in ⟨TABLE⟩ tags; the extracted text was then analysed further by the filtering step, which filtered ⟨TABLE⟩ tags that were used for formatting purposes using heuristic rules; the remaining text was further analysed by a table recognition step, the results of which were sent to the table interpretation step; finally, tables were displayed as attribute-value pairs by the presentation module developed by H.-H. Chen et al. (2000).

Similarily, when determining the non-redundant content information based on the ⟨TABLE⟩ tags in HTML documents, a cascading model was used. Pages from a website were extracted using a crawler. The extracted HTML pages were sent to a module, which identified content blocks partitioned by the ⟨TABLE⟩ tags. The contents in each block were input to a module, which calculated the word frequencies for each block. The word frequencies in each block were then analysed by a classifier, which distinguished redundant and non-redundant blocks (S. H. Lin & Ho, 2002).

When developing an automatic table ground truth generation system, the first step was to identify lines and words used in document images. A statistical method was then used to identify table regions. Finally, an x-y cut method was applied to identify table structures (Y. Wang et al., 2001). Similar procedural models where activities progress from one phase to another have been used by several other researchers (see, for example, Amano and Asada (2003); Hu et al. (2000a); Y. Wang et al. (2002); Tengli et al. (2004); Y. Wang et al. (2001)).

2.5 Evaluation of Table Layout Analysis Tasks

Evaluation is about comparing expected answers and experimental results based on some test data. Despite the prevalence of tables, there is currently no one accepted test corpus. This is partly due to the lack of agreement on how tables should be marked up. As a result, a variety of test data have been used for evaluating table analysis algorithms. For example, the Wall Street Journal (WSJ) was used as the test data in the work done by Pyreddy and Croft (1997); Ng et al. (1999). 1372 HTML pages from five Asian airline companies (China Airline, Eva Airline, Mandarin Airline, Singapore Airline and Fareast Airline) were used to test the table extraction algorithm developed by H.-H. Chen et al. (2000). When identifying table lines, Pinto et al. (2003) collected a set of documents from a crawl of `http://www.FedStats.gov`. Fifty-two documents containing 31,915 lines, of which 5,764 were table lines, became the training set. The development set featured six documents containing 5,817 text lines, of which 3,607 were table lines, and the test set featured 62 documents containing 26,947 lines, of which 5,916 were table lines.

On a smaller scale, X. Wang (1996) used 886 tables from five unspecified sources to test a table rendering algorithm. Thirteen Taiwanese news websites from `http://nse.yam.com` were used to test this algorithm's performance in discovering non-redundant content blocks (S. H. Lin & Ho, 2002). When differentiating formatting tables from genuine tables, 157 HTML pages from university websites were used (Tengli et al., 2004). When testing a medium-independent table extraction algorithm, a set of 25 ASCII documents and a set of 25 document images of unspecified sources were used (Hu et al., 2000a) . A collection of 851 tables (part of the Federal Register database distributed on CD-ROM by the National Institute of Standards and Technology) was used to test the work done by Garris, Janet, and Klein (1999). When deriving an abstract geometric model of a table from a physical representation, 22 documents of 5 companies' annual reports containing 662 tables were used (Hurst, 2003). In the attempt to assign table cells to their proper place in the logical structure of tables, 29 tables consisting of 91 domains formed a development set, and 4 tables consisting of 13 domains were held back as a test set (Hurst & Douglas, 1997b).

Most of the above mentioned test data was selected from a single source. The downside of using domain specific test data, especially the ones that have been professionally written and edited (such as WSJ), as a single source of test data is that the data are unlikely to be diverse enough to represent the varieties of tables in the real world (Lopresti & Nagy, 1999a). To overcome the deficiency of using domain specific test data, Y. Wang and Hu (2002a, 2002b) tried to collect test data from as many different varieties as possible from the web. They composed a set of keywords that were likely to indicate documents containing tables and used these keywords to retrieve web pages using the Google search engine. Using the keywords of 'table', 'stock', 'bonds', 'figure', 'schedule', 'weather', 'score', 'service', 'results', and ‘value’, 2,851 web pages were downloaded and 1,393 of them, collected from 200 websites, contained one or more tables.

Apart from using real data, synthetic data that was generated from a program was also used. In the experiments conducted by Y. Wang et al. (2001, 2002), 1,125 machine printed, noise free pages were used. These include 565 documents from business and law books, and the remaining pages were synthetic data automatically generated by a system developed by the authors. The system took tables and non-tables (the correctness of which has been verified) as the inputs, divided the inputs into small units, mixed the units together while adding a small degree of controlled variations, and output them.

The synthetic data discussed above was claimed to be highly accurate and did not require much human effort when identifying expected answers from test data (Y. Wang et al., 2001). Apart from the synthetic test data, expected answers for other test data were constructed in the following way: a preliminary set of expected answers was first constructed by programs, followed by manual verification and correction processes, which incorporate human judgement and are possibly assisted by GUI (Graphical User Interface) (H.-H. Chen et al., 2000; Hurst, 2003; Hurst & Douglas, 1997b; Penn, Hu, Luo, & McDonald, 2001; Pinto et al., 2003; Tengli et al., 2004; Y. Wang & Hu, 2002a, 2002b; Yoshida et al., 2001).

In either case, there has been no report on how human judgement was employed in constructing expected answers. In particular, there was no discussion on the guidelines of what counts as a table, whether the annotators should be different from the program designers, and what happens if human annotators disagree with each other’s judgement.

Recall and precision have been the mainstream evaluation measures for layout analysis. For table layout analysis tasks, recall is defined as the number of answers that are correctly identified divided by the number of expected answers, and precision is defined as the number of answers that are correctly identified divided by the total number of answers (correct or otherwise) a system produces. Depending on the tasks, the number of correct answers can be counted at the table, row, column or cell level. For example, when evaluating his table structure derivation algorithm, Hurst (2003) quoted recall and precision at both the table and the cell levels. In the literature, recall and precision are commonly used as the indicators for the efficiencies of algorithms (see, for example, Ng et al. (1999); Hu et al. (2000a); Hu, Kashi, Lopresti, and Wilfong (2001a); H.-H. Chen et al. (2000); Komfeld and Wattecamps (1998); S. H. Lin and Ho (2002)).

Although recall and precision can give an indication of the efficiency of a system, the downside of these measures is that they fail to differentiate the seriousness of errors, because minor errors (such as missing only one line in detecting a table) and major errors (such as missing almost the whole table) result in lower recall and precision in the same way. This deficiency is addressed in Section 4.4.

2.6 Summary

Researchers have gone a long way towards finding methods to extract and understand tables. Despite their efforts, there are areas that still need to be improved. Table representation methods is one such area. As shown in Section 2.2.2, existing table representations are very restrictive because they encode only the layout structures of tables, and cannot depict any semantic relationships between table components. This restriction makes it hard to perform high level processing work. As Thompson states: 'table markup contains a great deal of information about what a table looks like . . . but very little about how the table relates the entries . . . [This] prevents me from doing automated context-based data retrieval or extraction' (Thompson, 1996 cited in Hurst & Douglas, 1997b).

Secondly, at present there is no common markup scheme shared between the systems, there is no agreement on what the expected answers should be, and there is no common evaluation standard that is adopted by all researchers in the table processing community. As a result, it is hard to make systems communicate with each other, it is hard to evaluate systems using a common set of test data, and it is hard to discover the strengths and weaknesses of the algorithms used. Having a set of table markup guidelines that are commonly agreed upon could help to decide what the expected answers should be. Additionally, a common markup specification and an evaluation method that can reveal the sizes of errors could help to compare the strength of algorithms.

Many table processing systems have been produced over the years. These systems differ a lot in terms of the problems they try to solve, the information they rely on, and the techniques they employ. Most systems are very effective in solving the problems they are designed for by using a single source (or a small number of sources) of information. However, their applicabilities are usually very narrow because they fail to bring in a sufficient number of knowledge sources in their problem-solving processes. This dissertation takes the view that these problems do not need to be dealt with independently. Rather, they should work together to address table analysis in a larger domain. The rest of the dissertation proposes a framework that allows existing systems (both structural and semantic analysis systems) to work co-operatively towards the goal of improving table analysis.

‘

3

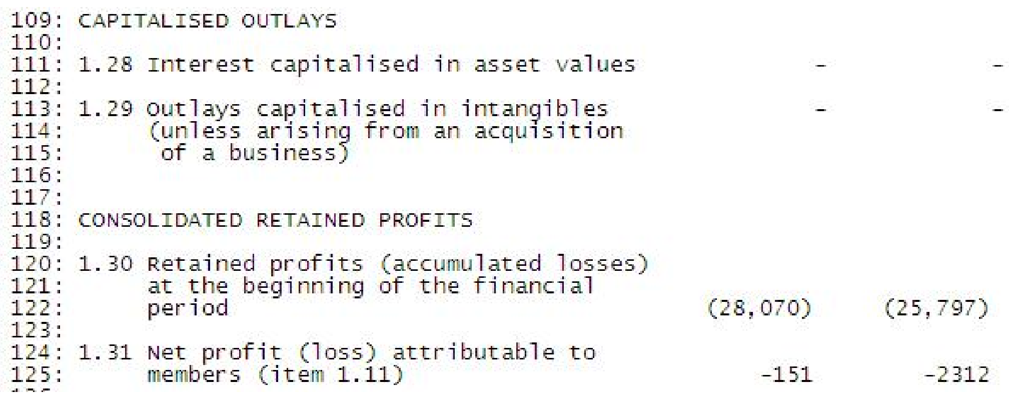
An RDF-Based Blackboard Framework for Table Analysis

3.1 Introduction

Table analysis is a complex problem for two reasons. The first is due to the unpredictable nature of printed documents: it is not possible to predict exactly what the next word or sentence or paragraph would be based on current content. The choice of words, the length of paragraphs and the way that characters (especially punctuation characters and space characters) are used are different from documents to documents. Because of the unpredictable nature, document attributes, such as whether a document contains tables, cannot be detected with 100% accuracy.

The second reason is that if a document contains tables, there are no rules or information that can reliably lead to the detection of table locations and table structures. This can be illustrated by the real examples in Figures 3.1 and 3.2. The table in Figure 3.1 allows two consecutive blank lines to be in a table. However, the table in Figure 3.2 does the opposite; two consecutive blank lines are used to separate adjacent tables. In general, a rule that works for some cases would fail for others for a wide range of reasons. The unpredictability of tables and the unreliability of information for finding tables make table analysis a complex task.

It has been discovered that solving complex problems often requires combining multiple kinds of knowledge, and the combined effect of multiple sources of knowledge can often identify the single most credible conclu-



```
109: CAPITALISED OUTLAYS
110:
111: 1.28 Interest capitalised in asset values              -           -
112:
113: 1.29 Outlays capitalised in intangibles                -           -
114:      (unless arising from an acquisition
115:       of a business)
116:
117:
118: CONSOLIDATED RETAINED PROFITS
119:
120: 1.30 Retained profits (accumulated losses)
121:      at the beginning of the financial
122:      period                                    (28,070)    (25,797)
123:
124: 1.31 Net profit (loss) attributable to
125:      members (item 1.11)                           -151       -2312
```

Figure 3.1: Two consecutive blank lines are used within a table (lines 116 and 117).

sion (Erman, Hayes-Roth, Lesser, & Reddy, 1980). The idea of bringing in multiple knowledge sources using agent-oriented programming architectures has been conceived and explored since the late 1980s. There have been successful applications of this idea (see examples in Bradshaw (1997); Bordini, Dastani, Dix, and Seghrouchni (2005)), and the conclusion was that multi-agent architectures were most suitable for building systems for solving complex problems, because by using agents (small programs that work autonomously) systems became more modular, more changeable, and more adaptive (see discussions in Jennings (2001); Padgham and Winikoff (2004)). In the table processing literature, experiments showed that table identification systems that made use of both layout and language knowledge sources outperformed systems that used only layout knowledge sources (see discussion in H.-H. Chen et al. (2000); Hurst (2001b); Pinto et al. (2003); Y. Wang and Hu (2002b)). This chapter discusses how the *blackboard architecture*, which integrates multiple *agents* (also known as *experts* or *knowledge sources*) into a problem solving process, can help to solve table analysis problems.

```
161: 2.4 Extraordinary items           -         -         -          -
162:                (details)
163:
164: 2.5 Total extraordinary
165:     items                       N/A       N/A       N/A        N/A
166:
167:
168: COMPARISON OF HALF YEAR PROFITS              Current     Previous
169: (Preliminary final report only)                year        year
170:                                              AUD000       AUD000
171: 3.1  Consolidated profit (loss) from
172:      ordinary activities after tax
173:      attributable to members reported
174:      for the 1st half year (item 1.22
175:      in the half yearly report)                (436)            1
176:
177: 3.2  Consolidated profit (loss)
178:      from ordinary activities after tax
179:      attributable to members for the 2nd
180:      half year                                   285      (2,313)
```

Figure 3.2: Two consecutive blank lines are used to separate two tables (lines 166 and 167).

3.2 Motivations for Using the Blackboard Architecture

The blackboard architecture was first published in 1980 when the HEARSAY-II speech understanding system was built. It can be understood as a global memory space (the blackboard) being provided to a group of agents who try to solve a problem collaboratively by writing their knowledge on the blackboard. The overall solution is found by combining the solutions contributed by individual agents. Because of the use of agents, the blackboard architecture has the ability to bring multiple knowledge sources, including partial knowledge, to a problem solving process. Using this architecture, the Hearsay-II speech understanding system was able to interpret 90% of the continuous speech sentences from a 1000-word vocabulary. Compared to the performance of other speech recognition systems at that time (such as the system developed by Systems Development Corporation that had about 25% accuracy, and the *Hear What I Mean* system that reached about 45% accuracy, this level of accuracy was considered to be very satisfactory (Hirschberg, March 2008). What has been observed was that the speech recognition problem and the table analysis problems share many common characteristics: there is not a single algorithm that can deterministically produce all the solutions to the problems; the input data has no fixed structures, and it is often noisy, ambiguous and incomplete; and finding solutions to each problem involves creating a (large) space of candidate solutions. Since the speech understanding problem was solved with a high level of accuracy under the blackboard architecture, it was hypothesised that the blackboard architecture can also help improve the performance of table processing tasks.

Agents need to interact with each other by exchanging messages, and there are two methods. The first is the open communication method where messages are posted in a common area and are available for all other agents. This open communication method is represented by the blackboard architecture (see Chapter 4 in Brenner, Zarnekow, and Wittig (1998)). The second is the message-passing method, where messages are directly exchanged between two agents except for broadcasting. This communication method is used in message-passing agent-based systems. In this dissertation, the second method is referred to as a *non-blackboard* or *subroutine-like archi-*

tectures. Although non-blackboard architectures can also bring multiple knowledge sources to a problem solving process, there are two fundamental differences between the blackboard architecture and a non-blackboard architecture. The first is the control strategy for the running sequences of agents (or modules), and the second is the communication among agents.

Agents in the subroutine-like architecture directly call each other. This requires built-in knowledge about the agents and their relationships in a system when introducing new agents. The blackboard architecture, on the other hand, does not have such a requirement, as agents can simply be added to a problem solving process and they will work out their own running sequences. This is because agents in the blackboard architecture bear the opportunistic and knowledge sharing characteristics simultaneously. Being opportunistic, as soon as a blackboard contains enough information for an agent to produce results (even partial results), it will do so without delay; and being knowledge sharing, an agent will publish all its knowledge on the blackboard. After the first agent shares its knowledge on the blackboard, subsequent agents activate themselves by the information available on the blackboard, rather than by explicit calls from other agents or some central sequencing mechanism.

The other major difference between the blackboard architecture and the subroutine-like architecture is the data communication among agents (or modules). In the subroutine-like architecture where an agent calls another, the caller provides the called agent with those data it deems relevant. In the blackboard architecture, agents are provided with the blackboard, which contains all data known to all the agents. One agent can use data created by previous agents' activations without the creators of the data having to know which agent will use the data and without the user agent having to know which agents might be able to create the data. Comparing the two approaches, the subroutine-like architecture makes it harder to introduce new agents without adopting the existing agents' internal data representation. Erman et al. (1980) point out, the subroutine-like architecture, in general, is suitable for systems that are relatively stable, where the need for introducing new agents is expected to occur very rarely. The blackboard architecture is more suitable for systems in which the type, number and interaction patterns of agents (or modules) change frequently.

When trying to identify tables and their structures in plain text documents, Ng et al. (1999) decomposed the problem into three steps: boundaries identification first of table, then row and column boundaries within the identified tables. These three steps are run in a subroutine-like architecture. Under the blackboard architecture, these three steps can be implemented as three independent problem-solving agents: one looks for tables, one looks for columns and one looks for rows. Unlike the rigid structure used in Ng et al.,'s experiments, there is no need to schedule the running sequence of the agents, as their running sequences are determined by the availability of the information they require to complete their jobs. For example, the column-finding agent can identify columns in an input document before the table-finding agent identifies the table. Comparing the two approaches, the blackboard architecture has the advantage of delivering potentially higher accuracy for the following reason. Under the subroutine-like architecture, if a table is not detected in the first step, then its structure will not be decided by the column-finding and row-finding steps. That is, the recall error will not have a chance to be fixed. In the blackboard architecture, because agents are run independently, it is possible to identify columns and rows before tables are identified. If a table cannot be identified by the table-finding agent alone, the columns and rows contained in the table can still have a chance to be identified by the column-finding and row-finding agents. Moreover, the columns and rows found by the column-finding and row-finding agents in this example can give hints to the table-finding agent that there is a table in the input document. This kind of agent co-operation is seen in the experiments covered in this dissertation, and is discussed in the performance analysis in Section 5.6.

The two fundamental differences discussed above make it easy to add or to remove knowledge sources under the blackboard architecture. This, in turn, results in several other benefits that make the blackboard architecture suitable for integrating diverse types of problem-solving agents. The first benefit is to encourage the development of diverse and independent knowledge sources. Separating the diverse sources of knowledge into independent programs allows different people to create, test, and modify programs independently and concurrently (Erman et al., 1980).

The second benefit is the inclusion of partial knowledge in a problem-solving process. When a complete solution is not yet available, finding partial solutions can still be feasible. Agents who have partial solutions to the problem can make a contribution by sharing their knowledge under the blackboard architecture. The shared knowledge could be useful to other agents in finding the overall solution (Erman et al., 1980). For example, agents that know how to find table-begin boundaries (but not table-end boundaries) can participate in the table boundary identification process.

The third benefit is the support of development of programs with a wide range of programming paradigms. In this dissertation, agents communicate with each other via the blackboard. As a result, each agent is free to use any type of internal problem solving architecture with arbitrary levels of complexity. As can be seen in Chapter 5, adding agent #9 (see Figure 5.2) to the problem solving process, for example, has no effect on how other agents work.

The fourth benefit is that the blackboard architecture provides ease of experimentation and evaluation of individual agents. Because agents are not rigidly tied together, it makes it easy to experiment with different configurations and measure their combined effectiveness by calculating the overall performance difference between including and excluding certain agents from the problem-solving process. Chapter 5 provides a number of comparisons on table boundary identification performance over various configuration of agents representing different heuristics.

Until now, semantic analysis can take place only after table structures are fully identified. This approach fails to recognise that there are many levels of semantic analysis, and that certain types of semantic analysis can help structural analysis. For example, determining the word type (such as number, currency, percentage, named entity, and non-numeric words) for each word in a document is a type of semantic analysis. This analysis can lead to (or increase the confidence of) the discovery of tables, especially when adjacent text lines contain the same number of numeric words. Under the blackboard architecture, this analysis does not have to wait until after tables are recognised. The blackboard architecture can integrate structural analysis and semantic analysis together, because agents are run independently and any agent can be added to a problem-solving process.

It is worth mentioning that although the blackboard architecture promotes the use of multiple agents, it retains the software decomposition ability. For example, the table boundary identification problem can be decomposed into two sub-problems: first to classify table lines and non-table lines, and then to identify the table boundaries. For the first sub-problem, multiple agents are allowed to find the solution together. Once the first sub-problem is solved, the next group of agents come along to solve the second sub-problem. As Buteau (1990) points out, the blackboard architecture is considered to be one of the most well-studied approaches to problem factorization. The rest of the chapter discusses an RDF-based blackboard for table analysis.

3.3 The Content of the Blackboard

As a domain independent software framework, the blackboard architecture was used in a number of applications (see, for example, Erman et al. (1980); Campos and Macedo (1992); Hayes-Roth, Pfleger, Lalanda, Morignot, and Balabanovic (1995); Wu and Lee (1993); McClain (2004)). Like all other blackboard systems, these applications must make a number of design decisions before they can be implemented. These include what can be written on the blackboard, how to decide on the structure of the blackboard, how to direct the problem solving process towards finding solutions to the overall goal, and how to detect and resolve conflicts among agents. The rest of the section discusses these design decisions for the blackboard system used in this dissertation.

3.3.1 Motivations for Using RDF

When trying to solve a problem through common effort, agents need to agree on a common knowledge representation so that they can communicate. When searching for a suitable knowledge representation for the agent-based approach to table analysis, the author of this dissertation considered a number of requirements. The first one is that the knowledge representation must allow agents to refer to, among many resources, table structures, table contents, and the locations of table cells, rows and columns.

The second requirement is that the knowledge representation must support an incremental problem solving pattern. Under the blackboard architecture, overall solutions are constructed step by step at multiple levels of abstraction. As cooperation takes place at each level of abstraction, the knowledge representation must support the agents' communication in the entire process.

The third requirement is that the knowledge representation must support inference. This is because when an agent reads information on the blackboard, it might want to derive new knowledge based on the blackboard content and its internal knowledge.

The fourth requirement is that the knowledge representation must be flexible enough to allow information, especially partial solutions, to be inserted as they become available. This is because in the blackboard architecture, the running order of the agents and the knowledge they contribute towards the final answers are unknown in advance.

In this dissertation, RDF (Resource Description Framework) has been chosen as the knowledge representation, and the content of the blackboard has to be RDF statements using a predefined vocabulary and concepts. There are several reasons for choosing an RDF as the knowledge representation scheme. The foremost reason is that RDF allows a table's structure to be stated as a set of explicit statements, and it has the ability to refer to low-level content, such as the positions of characters in documents, as well as high-level content, such as descriptions of tables, at the same time. For example, the statement, *the character at line 9 and column 50 is 'c'*, and the statement, *the table contains a summation calculation*, can both be represented using RDF syntax, which is discussed in Section 3.3.2.

The second reason is that RDF has the ability to allow any relationships between any parts of tables to be established. This ability makes RDF suitable for integrating table structural analysis with table interpretation analysis, as statements about geometric relations between elements and statements about functional relations can be expressed at the same time using RDF. For example, the statement, *'cell A is directly above cell B'*, and the statement, *'cell A is the header for cell B'*, can both be encoded in an RDF graph. This ability can help to reveal the abstract model of a table. As discussed in Chapter 2, abstract tables are expressed as multiple relationships between table components, and these relationships eventually lead to understanding the meaning of the data in tables.

The third reason is the availability of tools. Since RDF has been adopted as the standard of knowledge representation by the semantic web community, there are many RDF support tools available. The experiments covered in this dissertation use JENA, a Java implementation of RDF, that comes with a general inference engine for creating new statements and a SPARQL search engine for making information searching easy.

Finally, RDF matches the characteristics of the blackboard architecture very well. The RDF is essentially a directed graph, and this gives RDF the advantage, that the order of statements entered into the graph has no effect on the overall solutions. One of the characteristics of the blackboard architecture is that the running sequence of the agents is not deterministic at design time. This characteristic is matched by the fact that RDF makes no restriction on the order of statements entered by agents, as each statement eventually becomes part of a directed graph, regardless of its order of creation.

3.3.2 RDF Statements

An RDF statement can encode any relation between any two noun resources. For example, an RDF statement can be used to express *'line 327 in 50931.txt is a non-table line'*. In this example, the subject noun resource is *'line 327 in 50931.txt'*, the object noun resource is *'non-table line'*, and their relation is *'is'*. Subject, relation and object are the three components in an RDF statement. Noun resources represent anything that can be identified, although they can also represent abstract objects (see, for example, the blank node in Figure 6.3).

There are a number of ways to encode an RDF statement. For example, it can be printed as a 3-tuple; or it can be written as an XML file; or it can be printed as a directed graph. The directed graph format and the N3 format are disscussed here, because these RDF formats are used in this dissertation[1]. The directed graph representation for the above statement is given in Figure 3.3. Both the subject and the object are referenced by Uniform Resource Identifiers (URIs). The object noun resource is assigned a URI of `http://www.ics.mq.edu.au/~vanessa/Document#NonTableLine`. Because URIs are rather long and cumbersome, a shorthand, such as `Doc:NonTableLine`, is often used. Here, the part before the ':' ('Doc') represents a namespace, and the part after the ':' ('NonTableLine') is a local name in the 'Doc' namespace. For clarity reasons, the rest of dissertation uses shorthand for referring to URIs, and namespaces are sometimes omitted if doing so does not create ambiguity.

By convention, URIs are written inside elipses, and string literals are written inside rectangles (see example in Figure 3.4). Each vertex in an RDF graph represents a noun resource, and each edge represents a relation resource. Two adjacent vertices and the edge between them denote a simple statement. The subject of the statement is the noun resource from which the edge originates, the object of the statement is the noun object at which the edge terminates, and the edge denotes the relationship between the subject and the object. It is worth noting that since each statement is encoded as a connected pair of vertices in a directed graph, blank nodes (which are also referred to as anonymous resources) can be used as subjects or objects in statements.

To write an RDF statement in the N3 format, one only needs to enumerate the three components (each included in a pair of angle brackets) in the order of subject, relation and object followed by a period. For example, the above statement can be written in the following N3 format.

[1]For details of other RDF encoding schemes, please consult `http://www.w3.org`.

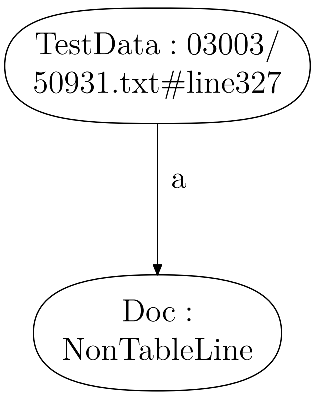


Figure 3.3: A statement under the RDF representation

```
@prefix Doc:  <http://www.ics.mq.edu.au/~vanessa/Document#> .
        <TestData:03003/50931.txt#line327> a <Doc:NonTableLine>.
```

When writing multiple statements that share the same subject, one only needs to write the subject once in the first statement. Subsequent statements contain relations and objects only, and the statements are separated by semicolons, with the last statement followed by a period.

A compound statement is represented by joining two or more simple statements. For example, the compound statement 'the content of the cell in the 2^{nd} column and the 8^{th} row is 'total current assets" could be broken down into the following four statements.

```
<theCell> <rdf:type> <doc:cell>;
          <tableModel:columnIndex> 2;
          <tableModel:rowIndex> 8;
          <doc:content> "total current assets".
```

Figure 3.4 shows compound statements that are represented in an RDF graph. In English language terms, the graph can be read as *document 119474.txt contains two tables. The first table contains cell(1,2) and cell(2,2), and these two cells have content strings of 'cash' and '1167' respectively.*

Using RDF, relations between table elements can also be expressed. For example, one might wish to represent 'cell(3, 18) is the sum of cell(3, 8) and

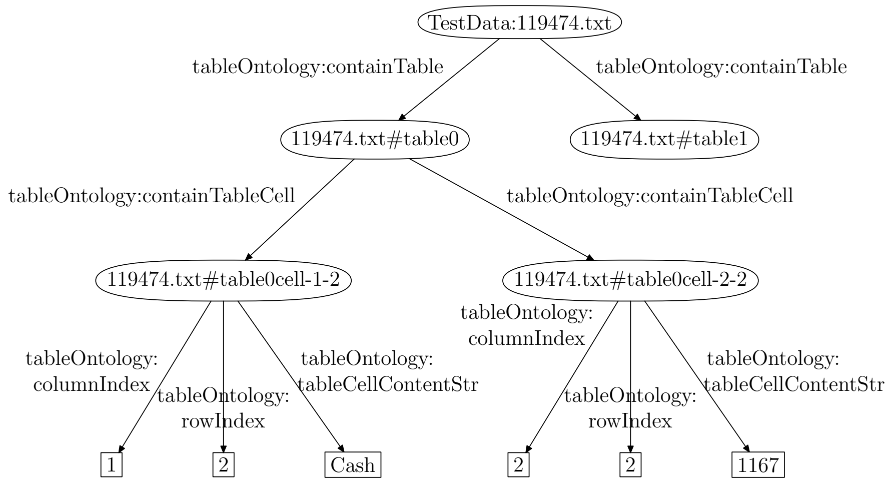

TestData:119474.txt
tableOntology:containTable
tableOntology:containTable
119474.txt#table0
119474.txt#table1
tableOntology:containTableCell
tableOntology:containTableCell
119474.txt#table0cell-1-2
119474.txt#table0cell-2-2
tableOntology: columnIndex
tableOntology: rowIndex
tableOntology: tableCellContentStr
tableOntology: columnIndex
tableOntology: rowIndex
tableOntology: tableCellContentStr
1
2
Cash
2
2
1167

Figure 3.4: An RDF representation for a document segment

cell(3, 17).' RDF provides three container data types (the *bag*, the *sequence* and the *alternative*), which can be used to describe groups of objects. A bag represents a group of unordered resources or literals. In the above example, cell(3, 18) is equal to the sum of the bag elements, which include cell(3, 8) and cell(3, 17). Figure 6.3 illustrates how a summation calculation is represented using RDF. In contrast to the bag data type, a sequence represents a group of ordered resources or literals. An alternative can be understood as *'one of the members'*. A container resource is uniquely identified by its URI, and it has an rdf:type property whose value is one of the predefined resources *rdf:Bag*, *rdf:Seq*, or *rdf:Alt*. Members of a container are described using RDF statements with subjects being the container URI, objects being the members' URIs, and the appropriate ordinal membership properties, such as 'rdf:_1', describing the relationships between subjects and objects. For example, the ordinal membership property for the first member of the container is *'rdf:_1'*, the second member of the container is *'rdf:_2'* and so on. The *bag* container data type is used for describing the sum calculations in tables in Chapter 6.

Since graphs do not allow duplicated vertices and edges, no duplicated statements can exist in RDF. However, it is possible to have different edges, labelled by different predicate URIs, between any pair of vertices.

3.3.3 Table Annotation

In order to regulate communication among agents and facilitate their co-operation, a method for referring to document content has to be agreed upon. In the table processing literature, extracted text strings have been used. For example, when marking up expected answers, the method of adding XML tags around extracted text strings, such as the one in Figure 3.5, was used in a number of publications (see, for example, Pyreddy and Croft (1997); Tengli et al. (2004); Wei et al. (2006)). The disadvantage of

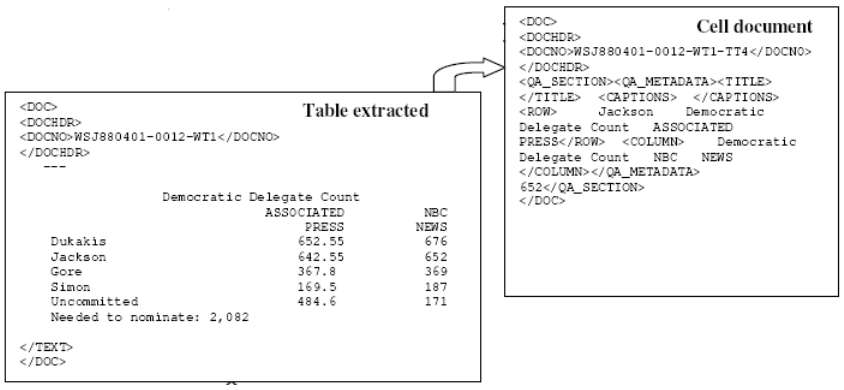


Table extracted

```
<DOC>
<DOCHDR>
<DOCNO>WSJ880401-0012-WT1</DOCNO>
</DOCHDR>
   ---

                Democratic Delegate Count
                           ASSOCIATED         NBC
                                PRESS        NEWS
    Dukakis                    652.55         676
    Jackson                    642.55         652
    Gore                       367.8          369
    Simon                      169.5          187
    Uncommitted                484.6          171
    Needed to nominate: 2,082

</TEXT>
</DOC>
```

Cell document

```
<DOC>
<DOCHDR>
<DOCNO>WSJ880401-0012-WT1-TT4</DOCNO>
</DOCHDR>
<QA_SECTION><QA_METADATA><TITLE>
</TITLE>  <CAPTIONS>  </CAPTIONS>
<ROW>     Jackson    Democratic
Delegate Count   ASSOCIATED
PRESS</ROW>  <COLUMN>    Democratic
Delegate Count   NBC   NEWS
</COLUMN></QA_METADATA>
652</QA_SECTION>
</DOC>
```

Figure 3.5: A table annotation method used in a number of table processing systems found in the literature.

using extracted text strings is that the locations of the tables in the original documents cannot be found from the expected answers. Additionally, identical tables in original documents result in multiple copies of the tables in the expected answers. If a system detects only one of the identical tables, it is difficult to tell which table is detected correctly and which table is missed.

To overcome these problems, this dissertation uses the stand-off annotation method discussed by Ide and Romary (2004, 2006). According to the stand-off annotation method, a string is referenced by the positions of its first and last characters, and the position of a character is addressed by byte offset. There are two popular ways to calculate the byte offset of a character. One is to use the sequence of the character in a document as the position.

The other is to use the line and column indices of the character. The latter is chosen in this dissertation because of a slight advantage: it makes it easy to reference vertically aligned strings, which frequently occur in tables.

The RDF table representation has advantages over the tree, graph, grid and dual hierarchy representations found in the literature. Although the x-y tree representation of tables was widely used (see Section 2.2.2), the graph representation suggested by Amano and Asada (2002, 2003) was a big step forward towards a more versatile representation, because a tree can be represented by a graph, but the reverse is not necessarily true (see Section 2.2.2). The downside of this graph representation was that it only allowed for ten pre-defined edge types. This problem does not exist in the RDF table representation, because RDF allows any relations to be expressed. For example, *table numbered 2 has 3 rows and 4 columns* can be represented by the following four statements under the RDF representation.

```
<:t>   a <GeneralVocabulary:Table>;
        <tableModel:index> 2;
        <tableModel:numberOfRows> 3;
        <tableModel:numberOfColumns> 4.
```

The table structure cannot be directly expressed by using only the ten pre-defined edge types in the graph model proposed by Amano and Asada (2002, 2003). On the other hand, the following RDF statements about the geometric relationships between cells represent the same table in Figure 2.7.

```
<box1> <LeftOf>  <box2>, <box5>, <box8>;
       <boundaryFullyContain> <box2>, <box5>, <box8>;

<box2> <LeftOf>  <box3>;
       <equalBoundary> <box3>;

<box3> <LeftOf>  <box4>;
       <equalBoundary> <box4>;

<box5> <LeftOf>  <box3>, <box6>, <box9>;
       <equalBoundary> <box6>;
       <below> <box2>, <box3>;
       <boundaryFullyContain> <box2>;
       <boundaryPartiallyOverlap> <box3>, <box9>;
```

```
<box6> <LeftOf>  <box7>;
       <equalBoundary> <box3>, <box7>;
       <below> <box3>;

<box8> <LeftOf>  <box9>;
       <equalBoundary> <box5>, <box9>;
       <below> <box5>;

<box9> <LeftOf>  <box10>;
       <equalBoundary> <box10>;
       <below> <box5>, <box6>;
       <boundaryFullyContain> <box6>.
```

3.3.4 Inference in RDF

Given a set of RDF statements, additional statements can be deduced based on some rules. As stated previously, a Java implementation of RDF, called JENA, is used in this dissertation. In JENA, inference is completed by modules called *reasoners* (Jena 2 Inference support (n.d.)). During the inference process, a reasoner is attached to an RDF graph to create a new RDF graph, which includes statements in the original RDF graph and the newly derived statements. As a result, queries to the new RDF graph return not only those statements that were present in the original graph but also the deduced statements. There are a number of predefined reasoners, such as the transitive reasoners and the symmetric reasoners, included in JENA. In this dissertation, JENA's general purpose reasoners, which can derive additional RDF statements based on user-defined rules, have been used. With a general purpose reasoner, it is possible to construct inference rules to encode knowledge such as *'a text line cannot be a table line and a not-a-table line at the same time'*. This piece of knowledge is represented by the following inference rule.

$$\begin{aligned}
&(?s\ a\ NotTableLine),\\
&(?s\ a\ TableLine)\\
&->\\
&(?s\ contain\ ConflictedStatements)
\end{aligned} \tag{3.1}$$

Inference plays an important role in this dissertation in detecting conflicts, which can be very subtle.

$$(?s\ a\ NotTableLine\_hasForbiddenChar) \tag{3.2}$$

$$(?s\ a\ EndOfTableRegion) \tag{3.3}$$

$$(NonTableLine\_hasForbiddenChar\ subClassOf\ NonTableLine) \tag{3.4}$$

$$\begin{aligned}&(?s\ a\ EndOfTableRegion)\\ &->\\ &(?s\ a\ TableLine)\end{aligned} \tag{3.5}$$

The statement in (3.4) denotes that *NonTableLine_hasForbiddenChar* is a subclass of *NonTableLine*. Hence, any objects that are of the *NonTableLine_hasForbiddenChar* type are also of the *NonTableLine* type. The statements in (3.5) state that *if a line marks a table-end boundary, it must be a table line*. The statement in (3.6) can be deduced from (3.2) and (3.4), and the statement in (3.7) can be deduced from (3.3) and (3.5). Statements (3.6) and (3.7) show a conflict according to (3.1).

$$(?s\ a\ NonTableLine) \tag{3.6}$$

$$(?s\ a\ TableLine) \tag{3.7}$$

Before detecting conflicting statements from the RDF graph on the blackboard, the inference rules listed in Appendix J are applied to the RDF graph to derive implicit statements posted by the agents. Conflicting statements are searched from the original RDF graph and the newly derived statements.

3.4 Directing the Problem-Solving Process

Blackboard systems can be classified as goal-driven or agenda-driven (see Carver and Lesser (1992)). In the goal-driven architecture, hypotheses are mapped onto goals, each goal is sub-divided into sub-goals, and a sub-goal is mapped onto one or more agents. In the agenda-driven architecture, agents are self-activating; as soon as an agent finds that it can make a contribution, it activates itself. The blackboard system in this dissertation is both goal-driven and agenda-driven. It is goal-driven because goals are decomposed into sub-goals, and each sub-goal is achieved by agents who have partial solutions. The problem solving process is directed by finding incremental solutions to sub-goals that make up of the overall goal. For example, the table boundary identification problem is decomposed into five sub-problems including one pre-processing step and one post-processing step in Chapter 5. The final combination step involves a single agent combining the solutions to the sub-problems to form an overall solution.

Within each sub-problem, agents are agenda-driven. Agents check if they can produce partial solutions based on the blackboard content. If there is enough information for an agent to contribute a partial solution, it will do so without delay. Figure 3.6 illustrates the general problem-solving pattern for each sub-goal. The pattern is characterised by a problem being attempted by multiple (possibly heterogeneous) agents during the *discussion phase*, where agents contribute their knowledge to the solution of the problem, followed by the *conflict resolution phase*, where a single agent resolves conflicting opinions produced by the agents.

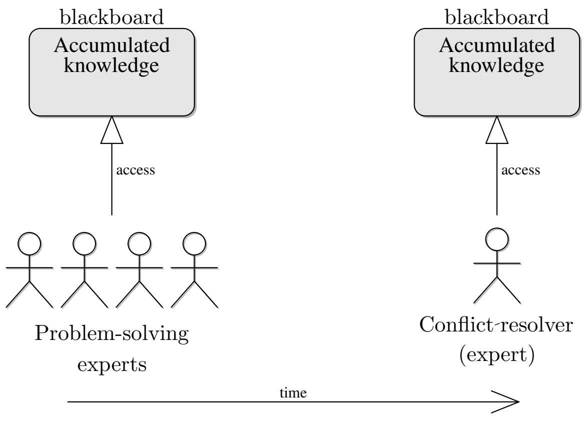


Figure 3.6: The general problem-solving pattern under the blackboard architecture: a group of agents discussing the solutions, followed by a single agent resolving conflicts and combing evidence to form a solution.

During the discussion phase, each agent has at least one opportunity to contribute its insights. This opportunity is given to an agent when the scheduler passes the blackboard around. In practice, each agent has more than one opportunity to contribute its solutions, because they are in a feedback loop of discussion. Once a discussion phase is ended, the problem solving process moves on to the next phase, and the agents involved will no longer have the opportunity to add more content. However, this can be overturned by adding a new phase involving the same agents. For example, suppose phase 1 involves agents A and B, and phase 2 involves agents C and D. During phase 1, agents A and B will be given sufficient opportunities to contribute their ideas. In general, the blackboard will not be passed onto agents A and B again once phase 1 is completed. If for some reason, agents A and B must run again after phase 2, then it can be achieved by adding phase 1 as the following phase after phase 2. Details of the conflict resolution phase are provided in Section 3.5.

3.4.1 Agent Scheduling

On the surface, the agenda-driven architecture seems to have no need for a controlling mechanism, as agents are self-activating in that architecture. However, as Carver and Lesser (1992) point out, agents cannot be activated as soon as information is available on a single-processor machine. Besides, the downside with 'absolutely no control' is that problems become intractable if systems try to activate all agents. Therefore, a blackboard system must have some agent scheduling mechanism.

In the Hearsay-II system, if an agent finds that the precondition for its activation is satisfied, it will inform a controlling unit about its actions. The actions are then rated, and the most highly rated one is performed (Erman & Lesser, 1990; Erman et al., 1980; Carver & Lesser, 1992).

In a recent project where McKenzie, Preece, and Gray (2006b, 2006a) used the blackboard architecture to solve a constraint problem, agents write their actions on a task panel (a reserved area in the blackboard) if they decide that they can make new contributions. The task panel is used by a controller for granting access to agents. This controlling mechanism ('register action first' then 'wait for access') is a key characteristic of the BB1 blackboard architecture. The BB1 architecture, invented by Hayes-Roth (1984) in 1983, consists of 2 blackboards: the domain blackboard and the control blackboard. The domain blackboard is equivalent to the normal blackboard discussed above, and it is used for solving the actual problems. The control blackboard contains information about the activation of the agents. When an agent finds that it can respond to a certain situation, it writes a message (called a bid) on the control blackboard. The bid is a signal to the rest of the system that an agent considers itself to be useful. A bid contains information about the characteristics of the agent, the agent's opinion of the consequence (or importance) of being activated, the amount of execution time it needs, and the expected quality, accuracy and the level of details of the results it produces (Reklaitis, Sunol, Rippin, & Hortacsu, 1996).

In this dissertation, the running order of the agents is sequenced by a control unit called *scheduler*, whose main functions are to mediate among agents competing to access the blackboard and to drive the problem solving process towards finding the overall solution. The scheduler maintains a list of participating agents. When a sub-goal is achieved and the problem solving process progresses to the next stage, the scheduler de-registers existing agents and allows new agents to participate in a new task.

A key idea behind the blackboard model is that problem solving should be opportunistic (see Carver and Lesser (1992)). This means that when deciding which agent should have the blackboard, the scheduler should assign the blackboard access to the agent that can make the 'best progress' toward delivering the overall goal, given the available data and the intermediate state of problem solving. The agent who can deliver the 'best progress' is determined by the amount of progress it can contribute relative to the computational costs of the agent. In practice, rating agents based on their potential contribution is difficult, and therefore is usually completed in an ad hoc manner. In this dissertation, the scheduler passes the blackboard to the registered agents in a round-robin fashion until no new assertions are made by any agent. If the blackboard is given to an agent that can make RDF statements based on the blackboard content, the agent writes the statements on the blackboard before it returns it to the scheduler. However, if the blackboard happens to be given to an agent who does not have immediate knowledge to share, then the agent would simply return the blackboard back to the scheduler. After the scheduler gives the blackboard to other agents, more knowledge is accumulated by the blackboard, and the agent will have more chances to produce new insights in the next round of blackboard passing.

When granting the blackboard access rights to an agent, the scheduler calls the agent's *accept()* function (each agent is required to implement such a function). Figure 3.7 shows the implementation of the scheduler.

```
public void start(Blackboard blackboard)
{
    int i;

    {
        // the old element counters
        long numOfSubjects_prev = blackboard.getNumOfSubjects();
        long numOfObjects_prev = blackboard.getNumOfObjects();
        long numOfStatements_prev = blackboard.getNumOfStatements();
        // the new element counters after passing the blackboard
        long numOfSubjects;
        long numOfObjects;
        long numOfStatements;
        boolean needToPassTheBlackboardAround = true;
        KnowledgeSourceBase theKS;

        while(needToPassTheBlackboardAround)
        {
            // *** pass the blackboard around in round robin fashion
            for(i=0; i<ksList.size(); i++)
            {
                theKS = (KnowledgeSourceBase)ksList.get(i);
                blackboard = theKS.accept(blackboard);
            }

            // *** decide if the blackboard should be passed around
            numOfSubjects = blackboard.getNumOfSubjects();
            numOfObjects = blackboard.getNumOfObjects();
            numOfStatements = blackboard.getNumOfStatements();
            needToPassTheBlackboardAround =
                    !((numOfSubjects_prev == numOfSubjects)
                    && (numOfObjects_prev == numOfObjects)
                    && (numOfStatements_prev == numOfStatements));

            if(needToPassTheBlackboardAround)
            {
                numOfSubjects_prev = numOfSubjects;
                numOfObjects_prev = numOfObjects;
                numOfStatements_prev = numOfStatements;
            }
        }
    }
}
```

Figure 3.7: A simple implementation of the scheduler

3.4.2 Message Deletion

One design decision that has to be made is to decide whether or not to allow agents to delete messages from the blackboard. There are pros and cons associated with each decision. On the one hand, if deletion is not allowed, then the blackboard content might grow to a size that is too big to be efficient. It would take too long for agents to search for the information that is relevant to them. Some messages are only useful in solving a sub-problem but are irrelevant to solving the subsequent problems. In this case, it is not harmful to remove them from the blackboard to increase space and time efficiencies. Sometimes a blackboard may contain erroneous information. By allowing deletion, errors can be removed from the blackboard. On the other hand, allowing deletion can cause loss of information, and this may result in agents entering into one type of infinite loop: a message that is written by an agent is deleted by another agent, and the absence of the message makes the first agent write the message again. If deletion is not allowed, and if an agent disagrees with any messages written on the blackboard, the expert could mark the statement being disagreed upon and the reasons for the disagreement. In this case, the process would not go into an infinite loop. The conflicting opinions could remain on the blackboard and could be resolved by an independent agent at a later stage. Both strategies (allowing deletion and forbidding deletion) are seen in blackboard systems. For example, in order to avoid loss of information, agents are not allowed to delete content from the blackboard in the AWB+B project (McKenzie et al., 2006b). However, because a robot has limited storage space, deletion is necessary, and therefore, allowed, in the robotic mapping project (McClain, 2004).

To balance the pros and cons, and to be consistent with the conflict resolving strategy discussed in Section 3.5, the blackboard content deletion policy used in this dissertation is to allow the conflict resolver (an agent) to delete conflicting statements, but the same privilege is not given to other agents. If an agent disagrees with what is written on a blackboard, instead of removing the statements that it disagrees with it can explicitly negate the content by adding a contradictory statement. The conflicting statements will then be resolved by the conflict resolver agent later in the conflict resolution phase.

3.4.3 Detecting the End of a Problem-Solving Process

For a given problem, the number of solutions can be 0, 1 or more than 1. A problem faced by a general blackboard system is how to detect that a problem has no solution or its solution has been found so that the problem-solving process terminates. In this dissertation, agents are assumed to run in finite durations, and detecting the end of a problem-solving process for the overall goal involves detecting whether the solution for each sub-goal is found or not.

In the blackboard architecture where agents are required to register their actions with a controlling unit before they can be activated, the end of a problem-solving process is reached if the controlling unit has no registered potential contribution. This termination detection method is used by McKenzie et al. (2006a). In their system, the registered actions are kept in a task panel. If the task panel is empty after a cycle of examining the blackboard by each agent, then the controlling unit will decide that problem solving process has reached an end, at which point it will despatch the final agent to look for the overall solution from the blackboard.

In this dissertation, the end of a problem-solving process is detected if the scheduler has passed the blackboard to each agent, and there are no new statements produced in a complete round of blackboard passing. In order to decide whether there is new information written on the blackboard, and whether there is any information being deleted from the blackboard, the scheduler can take a snapshot of the blackboard before giving it to an agent. Upon receiving the blackboard from an agent, the scheduler can compare the two versions of the blackboard statement by statement, and decide on the changes made to the blackboard. If an agent deletes information from the blackboard when it is not supposed to, the scheduler can restore the blackboard content before passing it to the next agent. If the number of new statements on the blackboard has increased since the blackboard was given to the agent, then the agent has made some new contribution. Otherwise, the agent makes no new contribution.

Although taking a snapshot of the blackboard and comparing the blackboard content can help the scheduler decide whether new changes have been made to the blackboard, such an operation is costly, and it can reduce the system's performance to an unacceptable level if the blackboard contains a large number of statements. In this dissertation, the scheduler approximates this operation by counting three numbers on the blackboard: the number of statements (that is the number of edges), the number of subjects (that is the number of vertices where the edges start), and the number of objects (that is the number of vertices where the edges end). The scheduler infers that there are no new statements if these numbers are unchanged during a new round of blackboard passing. A new round of blackboard passing starts whenever the scheduler encounters an agent that cannot make any new contribution, and a complete round is reached if the blackboard has been passed to each registered agent since a new round started.

It is worth noting that the way that the scheduler decides that there is no new information is simple but inefficient. It is simple because agents do not need to check whether the information they are producing has already been written on the blackboard or not, and agents do not need to know anything about termination; they just need to focus on doing their tasks. However, the termination detection method is inefficient because computational time is wasted at least in the final round of passing the blackboard when agents do their work as usual, but none of them produces any new insights.

The inefficiency could become problematic if computationally expensive agents are used in the problem solving process. It is possible that there are *independent agents* in a problem solving process. The amount of information produced by an independent agent does not vary as a result of the existence of other agents. To improve efficiency, it is recommended that independent agents (especially if they are computationally expensive) be separated from other agents, and the blackboard be given to each of these agents once. An example of using this separation strategy can be found in sub-tasks 1 and 2 as described in Section 5.2. Upon receiving the blackboard, an independent agent is activated and run once.

3.5 Detecting and Resolving Conflicts

While agents address the same problem independently, it is possible that they will disagree. Conflict resolution is one of the characteristics of a blackboard system, and being able to resolve conflicts is critical in order to apply the blackboard framework to table analysis.

There are a number of ways to resolve conflicts, and the typical ones are the 'trust-no-one', 'trust all', 'trust-the-most-reliable-agent', 'voting' and the 'weighted voting' strategies. The 'trust-no-one' strategy resolves conflicts by removing any conflicting statements. It treats conflicting statements as if they were not said. The 'trust all' strategy is at the other end of the spectrum. It tries to evaluate the consequences of the conflicting sides and choose the best one. This strategy is used in the AWB+B project, where conflicting statements are left on the blackboard and remain unresolved. If any parts of the contradictions are required in the composition of a solution, then they are used (McKenzie et al., 2006b).

The 'trust-the-most-reliable-agent' strategy resolves conflicts based on a weighting scheme; if multiple agents are involved in a conflict, the agent with the highest credibility rating prevails. This strategy is used in the robotic mapping project designed by McClain (2004). In their project, a robot would be in a conflicting state if an agent tells it to move forward to explore an unknown territory, and at the same time, another agent tries to instruct the robot to move backward to avoid collision with a physical object. McClain uses a dynamic weight assignment scheme. Depending on the state of their system, the weight assigned to each agent changes frequently. For example, the *object avoidance* agent would normally have a low weight unless the robot is within 3 inches of an object. This allows the agent who has most expertise in the conflicting area to take control of the situation should the system enter into a conflicting state.

Under the 'weighted average' strategy, decisions are made based on a weighted average credibility rating, which is calculated as the sum of products of each involved agent's weight and the score for its decision. Suppose that a text line is said to be a table line by agent A, and at the same time, the same text line is said to be a non-table line by agents B and C. If the score for a table line is 1, the score for a non-table line is -1, and the weights for agents A, B and C are 0.6, 0.3 and 0.2 respectively, then the weighted average credibility rating is equal to 0.1. If the weighted average credibility rating is greater than a particular threshold (say 0), then the conflict is resolved into a table line. Otherwise, the conflict is resolved into a non-table line. There are two special cases of the weighted average strategy. The first one is the case where every agent has an equal weight. When this happens, the weighted average strategy is equivalent to a voting system. The second special case occurs when one agent has the weight of 1 and all other agents have the weight of 0. Instead of assigning weights to agents, weights could also be assigned to the reasons behind agents' judgements.

In this dissertation, conflicts can occur in two situations: during the text line classification process, a text line could be classified as a table-line and a non-table line at the same time; and in the table boundary identification step, a text line could be identified as a table-begin and a table-end boundary at the same time. In the first situation, conflicts are resolved by using the 'trust-no-one' strategy. That is, if a text line is said to be a table-line and a non-table line at the same time, then both the table-line status and the non-table line status will be removed and the text line becomes an unclassified line. Table-lines can only appear in table-blocks, and non-table lines must appear outside table-blocks. Unclassified lines can appear in table-blocks as well as non-table blocks. By resolving conflicts this way, the conflict resolver is not biased towards either side of a conflict. Whether the line is eventually included in a table or not depends on subsequent processing steps. This simple strategy is ideal when the reliability of the agents is unknown.

In the second conflicting situation where a text line is said to be a table-begin and a table-end boundary at the same time, conflicts are resolved by using the 'trust all' strategy combined with the hypothesis-generation-evaluation-selection approach. When a conflict arises, each side of the conflict is used to generate table hypotheses. That is, the line is used as a table-begin boundary and a table-end boundary to construct table hypotheses, and these hypotheses will be evaluated and will compete to be selected as a final answer (see discussion in Section 5.4). There are two possible

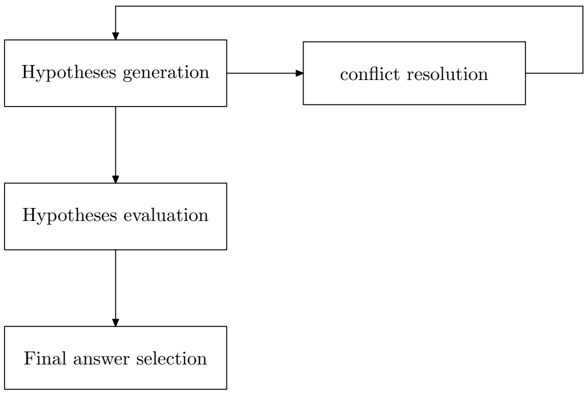


Figure 3.8: Disagreement among agents are resolved by the hypothesis-generation-evaluation-selection approach.

outcomes for resolving the conflicts: none of the hypotheses involving the conflicting sides is chosen as a final answer, because none of the hypotheses can maximise the overall score; or only one hypothesis involving the conflicts is chosen as a final answer, because the final solution assembler (agent #14) ensures that accepted hypotheses do not overlap (see Section 5.4.2). The overall approach is illustrated by Figure 3.8. After a group of agents is used to identify table boundaries in the beginning of a problem-solving process, table hypotheses are constructed based on agents' opinions about the locations of table boundaries [2]. After the table hypotheses are obtained, the agent that is responsible for selecting the final results has the option of accepting the hypotheses that are proposed by the most trustworthy agent(s). Since the final result selecting agent does not have information about the credibilities of other agents, it introduces an evaluation step, where a number of independent agents express their opinions about how much 'table-like' a table hypothesis is. Based on this information, the final result selecting agent selects the most credible solutions. Chapter 5 shows how this approach is used in finding the final results for the table identification problem.

[2]Agents do not propose table hypotheses directly. Instead, they propose table-begin and table-end boundaries, and hypotheses are constructed in such a way that every possible combination is covered.

3.6 The Structure of the Blackboard

Efficiency of information searching has been the main focus when designing the structure of a blackboard. As a problem-solving process proceeds, a blackboard accumulates more and more information. The time required to check the blackboard content can be unacceptably long[3], as agents have to search through a large amount of information on every blackboard access to determine what is of interest to them. An agent seldom wants to read all the information on a blackboard. Rather, it only wants to know the new information, or information it is interested in. One way of dealing with the problem is to partition the blackboard into regions (or layers). Each agent is assigned to one or more regions (a region can be assigned to more than one agent), and an agent only needs to observe the regions to which it is assigned in order to find the information of interest. The layered blackboard structure was used in the Hearsay-II project and in the AWB+B project, the goal of which was to find a solution to form workgroups that meet certain constraints (Erman & Lesser, 1990; Erman et al., 1980; McKenzie et al., 2006b, 2006a). In these projects, the data on the blackboard is structured hierarchically into abstraction levels, and each layer encapsulates relevant information through data abstraction. That is, each layer hides the detail of the layer below it.

Interestingly, a distributed blackboard structure was seen in a robot system where hardware resources is limited. McClain (2004) designed a mapping system where a group of robots walked around a building, and tried to construct a map of a floor. In this system, each robot maintains a copy the blackboard, and the entire blackboard was stored in a single file. With multiple copies of the blackboard existing at the same time, a synchronisation step was required.

The blackboard system used in this dissertation is not partitioned. However, searching for information can still be achieved efficiently due to the support of a query engine, and the factorisation of commonly used functions. In this dissertation, a query language, named SPARQL, was used to let agents

[3]It can take an agent more than 2 hours to search for a statement from a blackboard containing over 500,000 RDF statements in a low-end computer.

```
SELECT DISTINCT  ?tableBoundaryLineIndex
WHERE {
    ?tableBoundaryLineUri  Document.isTableLine "" .
    ?tableBoundaryLineUri  Document.hasLineIndex
                            ?tableBoundaryLineIndex .
    ?tableBoundaryLineUri  ?tableBoundaryProperty  "" .
    FILTER((?tableBoundaryProperty = <Doc:BeginningOfTableRegion_final>)
            || (?tableBoundaryProperty = <Doc:EndOfTableRegion_final>) )
}
 ORDER BY ?tableBoundaryLineIndex
```

Figure 3.9: An example of SPARQL query to select table boundary lines from RDF graphs

retrieve and manipulate data by specifying what should be retrieved rather than how the data should be retrieved. For example, Figure 3.9 shows how agents can select table boundary lines (both the table-begin and table-end lines) from an RDF graph.

Certain information is searched frequently. In this dissertation, search queries for this type of information are optimised and provided to agents in the format of functions. Factorising commonly used functions and making them available to agents increases agents' efficiencies, as it allows agents to focus on solving the problem rather than finding the information they need. Using functions has another advantage, which is to retain control of what agents can do with the blackboard. This is the underlying purpose of some blackboard systems that require agents' interaction with blackboards, which are completed via function calls (see, for example, McKenzie et al. (2006b, 2006b); McClain (2004)). Overall, a non-partitioned RDF-based blackboard is used for solving table analysis problems in this dissertation.

3.7 Summary

This chapter has discussed the advantages of applying the blackboard architecture to table analysis, the technical problems faced by blackboard systems, and how an RDF-based blackboard architecture solve the problems by providing an environment whereby multiple agents working towards goals improve the process of table analysis.

Table processing is an active research area that involves many subproblems, and new algorithms solving parts of the problem emerge all the time. As new algorithms become available, the blackboard architecture allows them to be implemented as agents and be easily added to the problem-solving process. Chapters 5 and 6 provide a concrete realisation of the RDF-based blackboard framework, and show how various sources of evidence can be used together in solving a table processing problem.

4 Table Identification Guidelines and Evaluation

4.1 Introduction

Before conducting any table identification experiments, two decisions must be made: how to identify a correct table when marking up the expected answers, and how should experimental results be compared against expected answers (for example, how should a partially correct experimental result be reported). This chapter provides answers to these questions. The first half of this chapter provides a set of table structure identification guidelines, so that ambiguities are reduced when constructing expected answers. The second half of this chapter defines a set of measures for evaluating table structural analysis tasks informatively, so that the extent of the effectiveness of systems is disclosed. It also discusses the challenges associated with evaluating table interpretation tasks.

4.2 Motivations for Table Identification Guidelines

Common sense definitions of tables are ambiguous in identifying tables and table structures. Figure 4.1 provides an example of confusing areas that often divide annotators. Feedback from colleagues shows three different opinions on how to recognise the text between lines 8 and 43 in Figure 4.1. The first one is that the text does not contain a table, because there are no clear column-row structures marking the column boundaries as well as the relationships between headers and their corresponding data. The second opinion is that the text contains a two-column table, due to the double space column separator in each line. The third opinion is that the text contains a three-column table, because each text line contains data with recurring patterns that would be re-written in a three-column table (with the first column as an index, the second column being the names of companies, and the third column featuring the integers showing the 'SHARES PER CREATION UNIT FOR 31/10/2001', suggested in line 8). Without guidelines, there would be more than one version of correct answers, which makes it impossible to evaluate experimental answers.

```
Date:31/10/2001

Opening Units on Issue 800,001.00
Applications 0.00
Redemptions 0.00
Ending Units on Issue 800,001.00

INDEX BASKET SHARES PER CREATION UNIT FOR 31/10/2001

AGL  AUSTRALIAN GAS LIGHT 2340
ALL  ARISTOCRAT LEISURE 2935
AMC  AMCOR LIMITED 4199
AMP  AMP LIMITED 7275
ANZ  AUSTRALIA & NZ BANK 9680
AXA  AXA ASIA PACIFIC 8597
BHP  BHP LIMITED 24077
BIL  BRAMBLES INDUSTRIES 6257
CBA COMMONWEALTH BANK. 8134
CCL  COCA-COLA AMATIL 4445
CML  COLES MYER LTD. 5756
CPU  COMPUTERSHARE 3575
CSL  CSL LIMITED 1031
CSR  CSR LIMITED 6100
FGL FOSTER'S GROUP 13205
FXJ  FAIRFAX (JOHN) 4780
GMF GOODMAN FIELDER 8354
GPT  GENERAL PROP. TRUST 12046
HVN  HARVEY NORMAN 4967
LLC  LEND LEASE CORP. 2797
MAY  MAYNE NICKLESS LTD 5400
MBL  MACQUARIE BANK LTD 1281
MGR  MIRVAC GROUP 4009
MIG  MACQUARIE INFRA. 10195
MIM  M.I.M. HOLDINGS LTD 11302
NAB  NATIONAL AUST. BANK 10084
NCP  NEWS CORPORATION 13613
NCPDP NEWS CORPORATION 14154
NDY  NORMANDY MINING 11604
NRM  NRMA INSURANCE GROUP 9102
SGPNF  STOCKLAND GROUP (NF) 266
WFTNG  WESTFIELD TRUST (NG) 206
SGT SINGAPORE TELECOMMUN 20449
SUNNC  SUNCORP METWAY (NC) 927
```

Figure 4.1: Table or not table: an example of ambiguities in text.

Errors in documents often cause confusion when locating tables. For example, lines 16, 24 and 30 in Figure 4.2 contain errors, because the content (the '-' character) is supposed to be the same as in the lines immediately preceding them (blanks). Since the errors partially destroy the column-row structure, confusion arises when deciding whether there is a table. Similar to the previous example, there are three different opinions on whether the lines between 4 to 30 in Figure 4.2 form a table: the first opinion is that lines 4 to 30 should not be recognized as a table because there is no column-row structure in the block; the second opinion is that lines 4 to 30 contain a table, because the column-row structure can easily be recognised despite the error; and the third opinion is that a table exists only between lines 4 to 14, because after line 14, the column-row structure is destroyed.

```
PAYMENTS TO DIRECTORS OF THE ENTITY AND ASSOCIATES OF THE DIRECTORS
PAYMENTS TO RELATED ENTITIES AND ASSOCIATES OF THE RELATED ENTITIES

 Current Quarter
 AUD

1.23 Aggregate amount of payments to
 the parties included in item 1.2                            -

1.24 Aggregate amount of loans to the
 parties included in item 1.10                               -

1.25 Explanation necessary for an understanding
     of the transactions


 -

NON-CASH FINANCING AND INVESTING ACTIVITIES

2.1  Details of financing and investing transactions which have had a
     material effect on consolidated assets and liabilities but did
     not involve cash flows

 -

2.2  Details of outlays made by other entities to establish or
     increase their share in projects in which the reporting entity
     has an interest

 -


FINANCING FACILITIES AVAILABLE
Add notes as necessary for an understanding of the position.
```

Figure 4.2: An example of errors in tables

The document in Appendix E illustrates the ambiguous nature of identifying table components. For example, without guidelines, it is difficult to decide

1. whether lines 1 and 3 should be recognised as part of a table. The reason for them to be included in a table is that they look like a table title summarising the table content. The reasons against them to be included as part of a table are that they do not have the same layout pattern (that is each row is indexed, and each row contains numeric numbers) as the rest of the table does; they do not seem to be a table title - especially line 3 (in which text is surrounded by a pair of brackets); and even if they form a table title, they are not part of a table because a table title is not a part of a table.

2. whether lines 5 to 11 should be recognised as two table cells (which are *'CURRENT PERIOD AUD000'* and *'PREVIOUS CORRESPONDING PERIOD AUD000'*) or as four table cells (which are *'CURRENT PERIOD'* and *'PREVIOUS CORRESPONDING PERIOD'* in one row, and *'AUD000'* and *'AUD000'* in a separate row). The argument for putting them in two cells is that they are two headers for two columns. The argument for extracting the string *'AUD000'* out to the following row is because the strings are linguistic constituents that serve a function on their own. In this case, they indicate that the numbers in the column are quoted as thousands of Australian dollars.

3. whether the strings *'PROFIT RESTATED TO EXCLUDE'* in line 159 and the string *'AMORTISATION OF GOODWILL'* in line 161 should be recognised as a table title or not. The reason for them to be classified as a table title is because they provide a summarisation of the table content. The reason for them not to be classified as a table title is because they appear in the same physical lines as the header cells. The same confusion exists for the word *'RATIOS'* in line 773.

4. whether the content at line 159 and thereafter should be classified as a continuation of the previous table, or as a separate table. The reason for the content to be considered as a continuation of the previous table is because of the indices *1.24* in line 169, *1.25* in line 179 and so on. These indices extend those from the previous table. The reason against

the content to be classified as a separate table is because of the table headers in lines 159 to 165. Since a table should not have headers that are far apart, the content should be classified as a separate table.

5. whether line 439 should be classified as a table title or not. The reason for it to be classified as a table title is that it summarises the table content between lines 449 and 491. The reason for it not to be classified as a table title is because three non-table lines (lines 443 to 447) are standing between the line and the table content.

6. whether line 559 should be part of a table title or not. Syntactically, it should be part of the table title because of its position in the table. However, semantically, it should not be part of the table title because a table title should summarise the table content, and it should not contain a reference pointer.

7. whether lines 819 to 837 should be recognised as a separate column from lines 811 to 815. The argument for lines 819 to 837 to be a separate column is that the indices '(a)' in line 819, '(b)' in line 823, and '(c)' in line 829, perform an independent function in a table. However, the argument against the lines to be recognised as a separate column is their physical alignment with lines 811 to 815.

8. whether the content between line 845 and line 853 should be classified as a table or not. The argument for them to be a table is that there is a table header, and there is a clear vertical alignment between the header and the data. The arguments for the content not to be classified as a table are that such a table is not semantically complete, as there is only one row in the table.

As the literature shows (see Section 2.2.3), tables at the physical level are seldom formally defined. In most cases, column-row structures are treated as tables. As can be seen from HTML documents, treating column-row structures as tables is wrong in many cases. In HTML documents, tables are embedded in table tags (⟨TABLE⟩). However, table tags are also used to render parts of documents into column-row structures to enhance visual clarity. Compared to these two usages of table tags, only a small subset of column-row structures are tables (H.-H. Chen et al., 2000). Overall, a column-row structure is a necessary, but not a sufficient, condition for a table. Given the number and the types of ambiguities that could occur in deciding the existence of tables, it is not practical to provide a concise definition to resolve the ambiguities. Rather, table identification guidelines are needed to resolve conflicting opinions.

4.3 Table Identification Guidelines

In order to reduce the ambiguity and the degree of freedom when determining the presence of tables and table structures in documents, the following guidelines, which are used in determining the expected answers for the table recognition experiments in this dissertation, are proposed. The guidelines, in principle, view a table as an interpretable column-row structure. For plain text documents, a stream of characters is a not a table unless it forms a column-row structure, and it can be interpreted within a context.

1. A table is a collection of data which fits in a $n \times m$ column-row structure consisting of horizontally and vertically aligned rectangular areas, with n being the number of rows and m being the number of columns satisfying $n \geq 2$, $m \geq 2$ and n and m cannot be equal to 2 at the same time. The restrictions that n and m must be greater than or equal to 2 and not both equal to 2 at the same time ensure that a table has more than one data column/row after the headers. If a column-row structure contains only one data column/row, then the structure is considered as a list rather than a table. According to this guideline, lines 845 to 853 in the document in Appendix E should not be recognised as a table.

2. Empty cells can occur in tables. However, a table should not contain an empty row or an empty column. That is, there should be at least one data cell in a column or row.

3. Each non-empty table cell must be a complete linguistic constituent such as a number, word, phrase, sentence, symbol or picture.

4. If a table contains headers, then the headers should be recognised as part of a table. Unless for the reason of gaining clarity, where table headers are re-displayed in the body of a long table, header rows should

appear in the top part of the table, and header columns should appear in the left-most part of the table. This guideline makes the recognition of a merged table undesirable. According to this guideline, the content at line 159 and thereafter in the document in Appendix E should be recognised as a separate table. For the same reason, lines 5 to 11 should be recognised as a table header spanning over four table cells (which are *'CURRENT PERIOD'* and *'PREVIOUS CORRESPONDING PERIOD'* in one row, and *'AUD000'* and *'AUD000'* in a separate row).

5. If a table contains headers, then each header must form the same relationship with the non-header data in the same column or row. If a table does not contain headers, the relationships between the data must be meaningful. The definition of 'meaningful relationships' among data is purposely not provided; or more precisely, it is left to be defined by usage. In practice, common relationships include header-data relationships, index-content relationships, attribute-value relationships, and class-instance relationships.

6. A column-row structure should be identified as a table if one of the following conditions is met.

   (a) The structure contains header columns (or rows), and the relationship between the header in each column/row and the data cell is clear. All the non-header cells in the same column/row should share some common properties. Ideally, the header columns (or rows) describe the common properties shared by the data.

   (b) The data in most columns (or rows) share some common words that are not stopwords, or the data in most columns (or rows) belong to the same data type such as date, integer or named entity.

7. Only those strings that contribute to the table content should be included in table bodies. Blank lines, ruling lines, column and row delimiters, and leading and trailing whitespaces should not be recognised as parts of a table.

8. A table cell can be neither an incomplete linguistic constituent (such as half of a word or picture), nor a composite linguistic constituent

(such as two numbers). The principle behind this guideline is that information in non-empty table cells should be interpretable. For example, numbers cannot be split between cells if connected by a relational or mathematical operator. According to this guideline, line 15 in Appendix E should be split into four segments: '1.1', 'Sales (or equivalent operating) revenue', '4,139' and '8,027'.

9. A table cell should contain as much information as possible. For example, if a line contains the word 'Total', and directly below this word is the word 'Expenses', then instead of having two cells ('Total' and 'Expenses'), one cell ('Total Expenses') should be recognised if doing so makes the table cell contain a complete linguistic constituent.

10. A table should contain the largest possible number of lines. That is, if line n to m is a table, and lines n to $m + k$ (where $k > 0$) is also a table, then the latter is preferred. This guideline makes partial table detection undesirable.

11. A spanned cell (horizontally or vertically spanned) should have the same semantic or linguistic relationship with the data cells in each column/row it spans over.

12. No column or row should be repeated in the same table. Repeating columns or rows should be merged together. This guideline is particularly applicable to long (or wide) tables that run across multiple pages. Long (or wide) table headers are sometimes repeated for clarity. When this happens, the expected answers should not contain multiple split tables. Instead, parts of the table from different pages should be joined together to form one table, and the table headers should not be repeated again in the expected answers.

13. Auxiliary components such as titles and footnotes should be recognised separately. That is, auxiliary components should not be in the markup of the expected answers. According to this guideline, lines 1 and 3 in the document in Appendix E should not be included in the expected answers. For the same reason, the string 'PROFIT RESTATED TO EXCLUDE' in line 159, the string 'AMORTISATION OF GOODWILL' in line 161 and the string 'EXPORATION AND

EVALUATION EXPENDITURE CAPITALISED' in line 439 should not be included in the expected answers.

With these guidelines, the ambiguities in deciding expected answers are reduced. The next section discusses how table analysis tasks should be evaluated.

4.4 Evaluation for Table Layout Analysis Tasks

4.4.1 Insufficiencies of Mainstream Measures

The main challenge in evaluating table layout analysis tasks is to decide on the measures that can provide feedback on the areas where systems can be improved. So far, recall and precision have been the mainstream reporting measures. Although these measures provide an overall measure for the accuracy of systems, they cannot give feedback on the types of mistakes made by systems.

Firstly, using just recall and precision does not indicate the seriousness of the errors, because recall and precision are calculated based on the number of errors, rather than the sizes of detection areas. For example, when trying to identify table boundaries, both missing only one line and missing an entire table result in 1 error count according to the current practice in the table processing literature. The former case (missing only one line) is less serious than the latter case (missing an entire table), and recall and precision do not differentiate between these two cases. Knowing about partial errors is desirable for many systems, and quoting only recall and precision does not provide sufficient information.

Secondly, recall and precision do not provide information on the types of errors. For example, when a table is not being detected correctly, one of these situations could occur: the table is entirely missed, or only parts of the table are detected, or parts of the table (together with some out-of-table content) are detected. Recall and precision do not differentiate between these situations.

The third reason is that the strengths and weaknesses of a table analysis system cannot be seen from recall and precision. In general, the more complex a table analysis task is, the more steps (sub-tasks) it involves, the more errors can be propagated from one step to the next, and the lower the expected recall and precision are. Suppose that a system wants to identify table boundaries, followed by a step working out the row boundaries. In such a system, errors produced from the table boundary identification step are propagated to the next step. By looking at just recall and precision for the overall performance, one cannot determine which part of the system produces most errors. However, if errors are shown at table-level, row-level, and cell-level, then the strengths and weaknesses of each part of a system can be revealed. The following section discusses supplementary evaluation metrics for table layout analysis.

4.4.2 A Granular Table Layout Evaluation Method

The drawbacks of using summative measures, such as recall and precision, in evaluation were noticed in the table research community. Effort has been made to improve evaluation reports by itemising error types. This includes reporting merging and splitting errors in addition to quoting recall and precision (see, for example, ,Hu, Kashi, Lopresti, Nagy, and Wilfong (2001); Hu, Kashi, Lopresti, and Wilfong (2001b); Hu et al. (2002); Kboubi, Chabi, and Ahmed (2005); Kieninger and Dengel (2005)). However, the proposed solutions fall short of providing measures for evaluating results that are partially correct.

For input documents that contain at least one table, the difference between experimental results and expected answers can be found by comparing the corresponding geometric regions, which might be polygons. Due to efficiency, comparing geometric regions has been replaced in the literature by comparing the bounding boxes of the regions (see discussions in Thulke, Margner, and Dengel (1999); Das, Saha, and Chanda (March 2002); Antonacopoulos, Karatzas, and Bridson (2006); Long, Cassidy, and Dale (2006); Antonacopoulos and Bridson (2007); Bridson and Antonacopoulos (2008)). Given two bounding boxes R_1 (representing an extracted table in the experimental result) and R_2 (representing a table in the expected answers), there are five possible overlapping patterns: R_1 and R_2 are 100% overlapping (see Figure 4.3), R_1 fully contains R_2 (see Figure 4.4), R_2 fully contains R_1 (see Figure 4.5), R_1 partially intersects with R_2 (see Figure 4.6), and R_1 does not intersect with R_2 at all (see Figure 4.7). The first overlapping pattern corresponds to a correct answer, and each other overlapping patterns corresponds to an error type (see Figures 4.4, 4.5, 4.6 and 4.7). To overcome the drawbacks of using only summative measures, and to take advantage of the efficiency and the informative nature associated with comparing bounding boxes, the following recommendations are made in addition to reporting recall and precision.

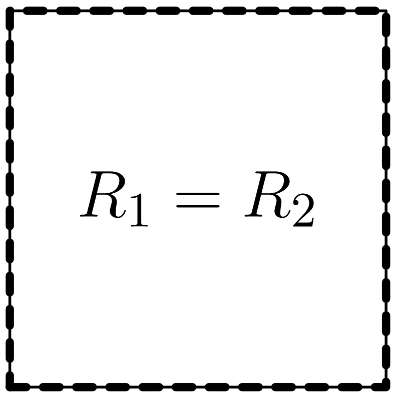


Figure 4.3: Experimental results and expected answers are 100% overlapping. This overlapping pattern is most desirable, because it indicates that the experimental result exactly matches with the expected answer.

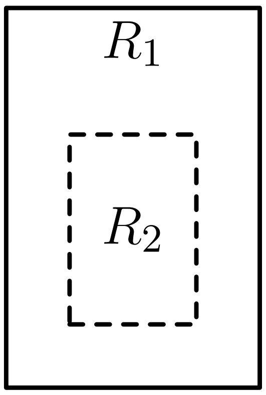


Figure 4.4: Experimental results fully contain expected answers. This overlapping pattern corresponds with the merging error type, where an element appearing in the ground-truth is fully contained within an element in the experimental result.

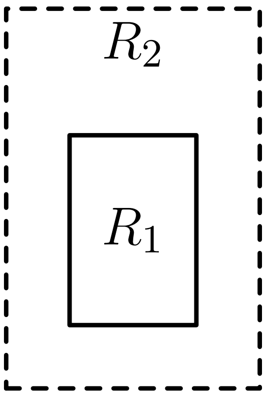


Figure 4.5: expected answers fully contain experimental results. This overlapping pattern corresponds with the splitting error type, where an element appearing in the experimental result is fully contained within an element in the ground-truth data.

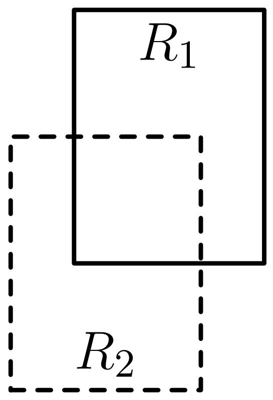


Figure 4.6: Experimental results partially intersects with expected answers. This overlapping pattern corresponds with the partial overlapping error type, where an element in the ground-truth data partially overlaps with an element in the experimental result.

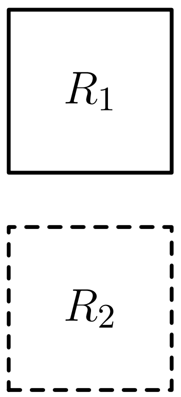


Figure 4.7: Experimental results and expected answers do not overlap at all. This overlapping pattern corresponds with two error types: the insertion error and the deletion error. An insertion error occurs when an element appears in the experimental result, and does not appear in the ground-truth markup. A deletion error occurs when an element appears in the ground-truth, and does not appear in the experimental result.

Recommendation 1: the number of test cases used for evaluation, and the portion of the test cases that do not contain tables should be disclosed. Test documents that do not contain tables can only test insertion errors; test results from this kind of document can either be that no table is detected, or one or more tables are inserted. These measures give an indication of the complexity and the completeness of the test data.

Recommendation 2: for input documents that do not contain tables, the size of insertion errors, which is calculated as the number of inserted tables divided by the number of documents, should be disclosed. This measure indicates the average number of tables (or the number of table lines) inserted per document.

Recommendation 3: in order to give feedack on the types of errors a system makes, the following measures should be disclosed at table-level, row-level [1] and cell-level: the total number of bounding boxes identified, the number of bounding boxes that are correctly identified, the number of insertion errors, the number of deletion errors, the number of proper subset errors (whereby parts of expected tables are detected without introducing non-table contents), the number of proper superset errors (whereby the whole expected table together with some non-table content is detected), and the number of interset errors (whereby parts of expected tables together with some non-table contents are detected). Additionally, in order to measure the overall effectiveness of a table layout analysis system, the percentage of areas that is correctly detected should be reported. The percentage is calculated as the overlapping area in the bounding boxes representing the expected answers and the experimental results divided by the total area of the two should be reported. In other words, the percentage is the ratio of the intersection of the expected and experimental bounding boxes to their union. The measures in this recommendation provide information on the distribution of errors based on the error types. Appendix K provides examples of the recommended measures.

[1]It is unnecessary to report errors at column-level, because any layout error that affects a column will also be reflected at the row-level.

4.4.3 Layout Annotation for Multi-level Evaluation

In order to compare experimental results against expected answers, a common format for encoding information must decided. This annotation specification determines the information encoding format used to represent both experimental results and expected answers.

The evaluation method discussed in Section 4.4.2 uses the overlapping pattern of bounding boxes to decide error types and error sizes. It turns out that the stand-off annotation method discussed in Section 3.3.3 suits this evaluation method very well: because table contents are referenced by their physical locations (line numbers and column numbers) in the input documents, a bounding box for each table (or a row, a column, or a cell) can be marked up using the following XML encoding specification.

1. A URI for a document is the namespace followed by a local name string that can uniquely identify the document. For example, `TestData:03003/50931.txt` is a valid URI for a document.
2. A URI for a table in a document is the document URI followed by the string ‘table’ followed by a unique integer. For example, `TestData:03003/50931.txt#table4` is a valid URI for a table.
3. A URI for a row in a table is the table URI followed by the string ‘row’ followed by a unique integer.
4. A URI for a table cell is the table URI followed by the string ‘cell’, an integer x, the ‘-’ character, and another integer y. The combination of x and y must be unique. For example, `TestData:03003/50931.txt#table4cell-2-14` is a valid URI for a table cell. Although it is common to use table indices or cell indices in URIs, it is incorrect to assume that this is always the case, and it is not recommended to derive table indices and cell indices from URIs. It is preferable to encode table locations and cell locations as explicit RDF statements.
5. A URI for a line inside a table cell is the cell URI followed by the string ‘line’ followed by a unique integer.

6. A data cell may contain one or more lines. Each line is specified by the positions of its first and last characters.

7. At the highest level, a markup document contains a pair of ⟨Doc⟩ tags (that is, ⟨Doc⟩ is the root element). Within a pair of ⟨Doc⟩ tags, there is an arbitrary number of ⟨Table⟩ elements, which contain one or more rows. Within each pair of ⟨Row⟩ tags, there are one or more ⟨Cell⟩ elements. A cell element may, in turn, contain one or more ⟨Line⟩ elements, which cannot contain any sub-elements.

8. The markup for a non-spanned cell is to include each of its lines in a pair of ⟨Line⟩ tags.

9. The markup for spanned cells is very similar to the markup for non-spanned cells, except that additional spanning properties are specified within the ⟨Cell⟩ elements. If a cell is horizontally spanned, then both the *SpanColStartIndex* and *SpanColEndIndex* properties must specified. If a cell is vertically spanned, then both the *SpanRowStartIndex* and *SpanRowEndIndex* properties must specified.

10. There is no child element (especially the ⟨Line⟩ element) for empty cells. That is, an empty cell is marked by a matched pair of ⟨Cell⟩ and ⟨/Cell⟩ tags with a cell ID as a property. For example, the ⟨Cell cellID="table2cell-0-1" cellType="emptyCell"⟩ ⟨/Cell⟩ is a valid markup for an empty cell.

The XML schema for this annotation specification is provided in Appendix F. Appendix H shows an example of a markup up file that follows the specification.

Given an XML file containing the expected answers and an XML file containing table extraction results, the evaluation process begins by verifying that the files meet the annotation specification. It does this by applying a parser that implements the context-free grammar that is listed in Appendix D to the files. If the files can be parsed by the parser successfully, then every data item is enclosed between an open tag and a close tag. By traversing the parsed trees, a number of conditions can be checked. These include whether there are illegal tags (that is, tags that are not 'DOC', 'TABLE', 'ROW' or 'LINE') in the files; whether the open tags match with their corresponding close tags; and whether the locations of the table lines are fully specified.

After confirming that the file containing expected answers and the file containing experimental results meet the annotation specification, the parsed trees are traversed again for determining the bounding boxes for tables, rows and cells in the corresponding files. These bounding boxes can then be used for evaluating table layout analysis performance at the table-level, row-level and cell-level.

4.5 Test Data Used in This Dissertation

The experiments covered in Chapters 5 and 6 were conducted on the test data sampled from the ASX corpus, which contains financial reports (in plain text format) submitted to the ASX by public companies since 2000. There are 19 types of financial reports with a variety of complexity. For example, the takeover offers are short documents that usually contain no tables. The company annual reports are documents that usually contain complex tables in which structures cannot be easily agreed upon by human annotators. Each document type contains one or more sub-types. The documents in the corpus are organised by the year they were produced. Within the same year, the documents are organised into directories according to the type and the sub-type to which they belong. Table 4.1 shows the information of the corpus from where test data is sampled. Since the test data and the training data are sampled from the years between 2000 to 2004, changes occurred after 2004 will not affect the experiments in this dissertation. It is believed that individuals and organisations can purchase the corpus from ASX.

There were two sets of test data sampled using two methods. The first set was sampled from the 2000 and 2004 reports respectively. The sampling method is to randomly take a fixed percentage (3%) from each directory. If 3% of the files in a directory is less than one, then it would take one file. This sampling method balances two factors: each document type must be included in the sample (for this reason, it samples from each directory), and

Year	Number of documents	Number of directories	Size of documents (bytes)
2000	918	47	4.95M
2001	873	120	10.6M
2002	589	126	10.1M
2003	88	65	0.35M
2004	482	95	1.1M

Table 4.1: ASX corpus information 2000 - 2004

the sample reflects the document composition (the more files a directory has, the more samples it has). The second test data set consists of 6 plain text documents that are manually selected from the ASX corpus. The documents were selected because they contain tables that are considered (by the author of this dissertation) to be relatively difficult to detect: some do not have columns that are perfectly aligned, some occupy multiple pages, some neighboring tables are physically so close together that their boundaries are not obvious, some contain multi-line cells and spanned cells, some contain empty rows, and some table widthes are wider than the document width and the tables have to be split into two parts. The 10 documents in the training data are also manually selected, and their level of complexities is compatible with the one in test set #2. The details of the test data and the training data are listed in Figure 4.2.

Test set samples	Number of documents not containing tables	Number of documents containing tables	Total number of documents
Test set #1	275	71	346
Test set #2	0	6	6
Training data	0	10	10
Total	275	87	362

Table 4.2: Table distribution in test data

After the test data were sampled, and before any experiment was conducted, expected answers were decided from the test data by a single human annotator, who was asked to observe the table identification guidelines discussed in Section 4.3. The annotator had no specialised knowledge about table processing. The annotator looked for tables from documents in the test sets, and used a markup tool to markup the location of each table using the stand-off annotation method discussed in Section 3.3.3. The markup task was completed in two phases over a period of 9 months. The first phase covered 106 documents from test set #1 and all of the 6 documents from test set #2. The second phase covered the rest of the 240 documents from test set #1.

4.6 Summary

This chapter provides solutions to two problems preventing direct comparison between table layout identification systems. Firstly, by providing a set of guidelines for recognising tables and an annotation specification for marking up tables, this chapter reduces the degree of freedom in constructing and encoding expected answers for table layout analysis tasks. Secondly, the multi-level evaluation report measures discussed in this chapter provide more fine-grained insights of the scope and effectiveness of table layout identification systems. Together, they pave the way for directly comparing the effectiveness of table layout analysis systems. With this ability, research activities such as competitions can be promoted.

5 Applying the Blackboard Framework to Table Boundary Identification

5.1 Introduction

The blackboard architecture provides a general framework to make it easy to incorporate a wide range of knowledge sources in the problem-solving process. Using the RDF-based blackboard framework discussed in Chapter 3, this chapter discusses table boundary identification experiments using multiple knowledge sources, including selected ones from the literature. It compares the results of programs running in two conditions - using heuristics in the traditional subroutine-like architecture and using multiple agents in the blackboard architecture. Through the experiments, the feasibility and the benefits associated with using the blackboard framework are illustrated.

5.2 Task Overview

The task for the experiments in this chapter is to identify tables and their locations in plain text documents. This task is accomplished by specifying the line indices of each identified table's boundaries (both *begin-boundaries* and *end-boundaries*). This task was chosen because the literature shows that it is a complex task (it contains all the challenges mentioned in Section 3.1), and all proposed algorithms experience some difficulties in detecting tables from an arbitrary set of documents.

Because the task is too complex to be achieved in a single step, it is decomposed into simpler, solvable sub-tasks. The decomposition strategy used in this chapter is to classify each line in an input document as a table-line, or a non-table line if there is evidence for the classification. If there is no evidence for the classification, the line remains in the unclassified category. After that, tables (and hence table boundaries) are identified from consecutive table-lines (permitting sparse unclassified lines to occasionally appear in tables). This approach is made clear by the example in Figure 5.1, which shows the line indices and the content of an input document in the first two columns. The third column shows the line classification results. For example, lines 29, 56, 57 and 58 are classified as non-table lines, because they are long text lines that do not contain two or more consecutive space characters. Lines 31, 32, and 34 are classified as table lines because they contain wide spaces that are frequently used as column delimiters in tables. There is no opinion about how lines 35, 39 and 43 should be classified, and these lines are remained in the unclassified category. Column 4 shows the table boundary identification results and the reasons for the decisions. For example, line 31 is identified as a table-begin boundary for three different reasons: it contains named entities that typically appear in table headers, it is at the point where a block of table-lines appears after a block of non-table lines, and it is the first line that does not contain numeric data but appears on aligned numeric sequences (likely to be the header of numeric columns). The last two reasons are also used to classify line 44 as a table-begin boundary. Line 41 is classified as a table-end boundary, because it lies between two table-begin boundaries (lines 31 and 44), and it maximises the length of the first table.

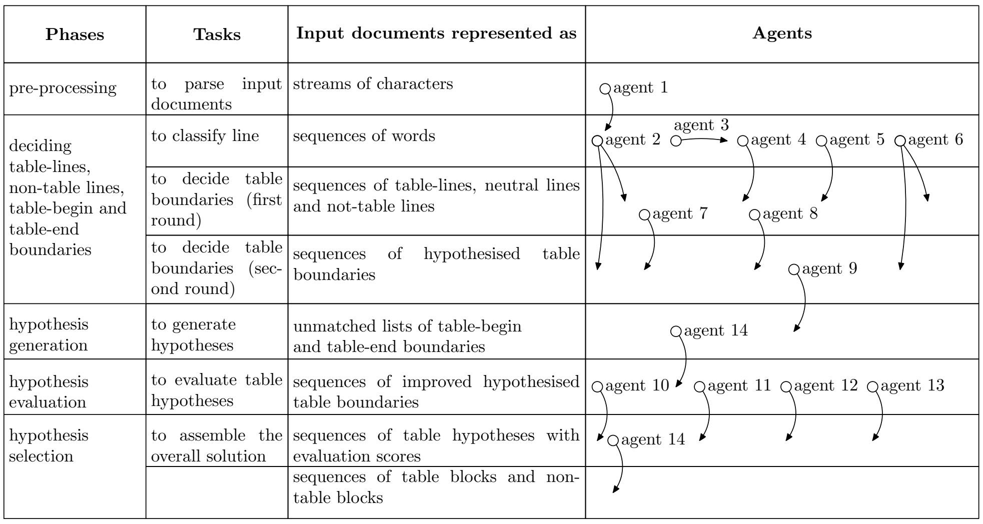
1	(A) GEOGRAPHICAL SEGMENTS - YEAR ENDED 30 JUNE 2000	✠	
2			
3	AUSTRALIA PHILIPPINES MALAYSIA HONG KONG	§	♣▽♠
4	$,000 $,000 $,000 $,000	§	
5			
6	Sales to customers 62,066 1,029 371 1,609	§	
7	outside economic entity		
8	Inter-segment sales 746	§	
9	Other Revenue 584 16 3	§	
10	63,396 1,045 371 1,612	§	
11	Operating Profit/(Loss)		
12	after Tax 2,445 (414) 58 649	§	
13	Total Assets 49,952 855 325 1,725	§	♡
14			
15	(table cont.)		
16	ELIMINATION CONSOLIDATED	§	♣♠
17	$,000 $,000	§	
18			
19	Sales to customers - 65,075	§	
20	outside economic entity		
21	Inter-segment sales (746)	§	
22	Other Revenue 603	§	
23	(746) 65,678	§	
24	Operating Profit/(Loss)		
25	after Tax 2,738	§	
26	Total Assets (15,847) 37,010	§	♡
27			
28			
29	(B) GEOGRAPHICAL SEGMENTS - YEAR ENDED 30 JUNE 1999	✠	
30			
31	AUSTRALIA PHILIPPINES MALAYSIA HONG KONG	§	♣▽♠
32	$,000 $,000 $,000 $,000	§	
33			
34	Sales to customers 40,235 1,797 135 124	§	
35	outside economic entity		
36	Inter-segment sales 254 48	§	
37	Other Revenue 127 4	§	
38	40,616 1,849 135 124	§	
39	Operating Profit/ (Loss)		
40	after Tax 1,757 92 (301) 2	§	
41	Total Assets 40,967 787 417 163	§	♡
42			
43	(table cont.)		
44	ELIMINATION CONSOLIDATED	§	♣ ♠
45	$,000 $,000	§	
46			
47	Sales to customers - 42,291	§	
48	outside economic entity		
49	Inter-segment sales (302)	§	
50	Other Revenue 131	§	
51	(302) 42,422	§	
52	Operating Profit/(Loss)		
53	after Tax - 1,550	§	
54	Total Assets (21,576) 20,758	§	△
55			
56	The Principal activity of the CIL Group consists of engineering,	✠	
57	scientific and project management services in the earth sciences,	✠	
58	water and natural resources sectors throughout Australia and	✠	

Figure 5.1: Table boundary identification steps

§: the line is classified as a table-line, because it contains column delimiters.
✠: the line is classified as a not-table line, because it is a long text line.
♣: the line is identified as a table-begin boundary, because it contains keywords that are frequently appeared in table headers.
♠: the line is identified as a table-begin boundary, because it does not contain any numeric data, and it is the first table-line above some aligned numeric data.
▽: the line is identified as a table-begin boundary, because it is a table-line, its previous line is a non-table line, and its following line is a table-line.
△: the line is identified as a table-end boundary, because it is a non-table, its previous line is a table-line, and its following line is a non-table line.
♡: the line is identified as a table-end boundary, because there is no table-end boundary between two table-begin boundaries, and counting from the first table-begin boundary, it is the first table-line that is followed by a non-table line.

As discussed in Section 3.2, the main benefit of using the blackboard architecture is the ease of combining various knowledge sources. In order to show that both existing and future knowledge sources can be integrated in a single system, 16 knowledge sources, including 5 based on published algorithms from the literature, are used in the experiments discussed in this chapter. Figure 5.2 lists 14 of the 16 agents (the missing ones are the sub-property inference agent (agent #15) and the conflict resolver (agent #16)). These include the tokeniser (agent #1), the table finder based on context-free grammar (agent #2 implementing the algorithm described by the author of this dissertation. See Long et al. (2005)), the neighbouring text line identifier (agent #3), the non-table line finder based on line length (agent #4), the non-table line finder based on surrounding line types (agent #5), the table-line finder based on layout features (agent #6 implementing the algorithm described by Ng et al. (1999)), the table boundary identifier based on changes of line types (agent #7), the table-begin boundary identifer based on numeric columns and keywords (agent #8), the boundary line balancer (agent #9), the table evaluator based on column structures (agent #10 implementing the algorithms described by Hu, Kashi, Lopresti, and Wilfong (2001b, 2001a); Lopresti and Wilfong (2001)), the table evaluator based on line correlations (agent #11 implementing the algorithm described by Hu et al. (2000a)), the table evaluator based on header locations (agent #12 incorporating header information that is used in a number of works in the literature (see, for example, Embley et al. (2002); Pinto et al. (2003); Wei et al. (2006))), the table evaluator for finding split tables (agent #13), and the final solution assembler (agent #14). The reasons for including these knowledge sources in the experiments are that they are frequently used by various researchers in detecting tables; the literature has provided sufficient details for the information to be implemented as agents from their written descriptions; they represent various kinds of information (such as layout information and language information); and they cover various programming paradigms such as deterministic methods, machine learning, and grammar rule based algorithms.

As mentioned in Section 3.4, agents in the blackboard architecture can work out their running sequence based on the information on the blackboard. In this experiment, if agents #1 to #8 were put in a single discussion step,



Phases	**Tasks**	**Input documents represented as**	**Agents**
pre-processing	to parse input documents	streams of characters	agent 1
deciding table-lines, non-table lines, table-begin and table-end boundaries	to classify line	sequences of words	agent 2, agent 3, agent 4, agent 5, agent 6
	to decide table boundaries (first round)	sequences of table-lines, neutral lines and not-table lines	agent 7, agent 8
	to decide table boundaries (second round)	sequences of hypothesised table boundaries	agent 9
hypothesis generation	to generate hypotheses	unmatched lists of table-begin and table-end boundaries	agent 14
hypothesis evaluation	to evaluate table hypotheses	sequences of improved hypothesised table boundaries	agent 10, agent 11, agent 12, agent 13
hypothesis selection	to assemble the overall solution	sequences of table hypotheses with evaluation scores	agent 14
		sequences of table blocks and non-table blocks	

Figure 5.2: Participating agents and their levels of information processing. The levels where the arrows originate from indicate the input levels, and the levels where the arrows terminate indicate the output levels.

they could produce the table hypotheses. However, this would have been inefficient because of the way the scheduler detects the end of a process involving computationally expensive agents (see Section 3.4.3). To improve efficiency, the hypotheses generation step is further divided into sub-goals. A preprocessing step involving agent #1 is created, and agent #2 is separated out from other agents. The blackboard is passed to each of these two agents once, as opposed to being passed to each agent in a round robin fashion until no agent can infer any new information.

Initially, input documents are represented as streams of ASCII characters. As the problem-solving processes advance, input documents are represented at a number of progressing levels: the word level, the line level, and ultimately the table level. Figure 5.2 shows this decomposition in terms of levels of information needed to be processed. It also shows the agents involved at each level of information processing.

5.3 Functional Description of Agents

5.3.1 Processing Input Characters

Agent #1 - Tokeniser
Agent #1 is run as a preprocessing step, and its main goal is to post input documents, word by word, onto the blackboard, so that other agents can view the input documents. To do this, it tokenises input documents using the context-free grammar rule listed in Appendix A. The tokeniser produces tokens that correspond to the words in the input documents. The positions of the words (that is, their line and column indices) are given by the tokens, and this information is posted on the blackboard. Because the grammar contains regular expressions for identifying certain word types, such as 'positive decimal number' and 'percentage', the agent understands the semantics of certain words in input documents. This knowledge, such as the one listed below, is also written on the blackboard.

```
<file:110730.gsml#line45sequence8>
      a       Doc:SequenceOfCharacters ;
      Doc:beginColumn
              "60"^^<http://www.w3.org/2001/XMLSchema#int> ;
      Doc:beginLine
              "45"^^<http://www.w3.org/2001/XMLSchema#int> ;
      Doc:endColumn
              "64"^^<http://www.w3.org/2001/XMLSchema#int> ;
      Doc:endLine
              "45"^^<http://www.w3.org/2001/XMLSchema#int> ;
      Doc:hasContentString
              "1,692"^^<http://www.w3.org/2001/XMLSchema#string> ;
      Doc:hasSequenceType
              GeneralVocabulary:Integer, GeneralVocabulary:Positive ;
      Doc:sequenceIndexInLine
              "8"^^<http://www.w3.org/2001/XMLSchema#int> .
```

5.3.2 Processing Input Lines

Agent #2 - Table finder based on context-free grammar
Agent #2 identifies tables using the context-free grammar that is published by the author of this dissertation (see Long et al. (2005)). This agent represents the programming paradigm that uses context-free grammars, as used in the work done by Amano and Asada (2002, 2003); E. Green and Krishnamoorthy (1995c) for detecting tables. The agent processes an input document in two steps: first to classify

input lines, then to identify tables in documents.

In the first step, the agent parses documents using a parser based on the context-free grammar rules described in Appendix B. Due to the grammar, each input line of a document produces one of the three tokens: a blank line token, a ruling line token or a text line token. The parsed tree for a particular document is then traversed twice. In the first traversal, input lines are preliminarily classified using the following rule: blank lines and ruling lines are left in the unclassified category; text lines that contain certain substrings, such as ']]⟩' or XML tags, are classified as non-table lines because the lines are likely to be the file header; text lines containing sub-strings that are frequently used as table column delimiters (according to this agent, a column delimiter is defined as two or more consecutive space characters, tab characters, asterisk characters, or a single vertical bar character) are classified as table-lines; and all other lines are left to be unclassified, as the agent does not have enough information to make a decision.

In the second traversal, input documents are filtered internally using the following rule: each blank line is replaced by an empty line [1]; each ruling line is transformed to a uniform format by replacing each character with the '-' character; each unclassified text line or a non-table line is replaced by an empty line; and the content of each text line that has been identified as a table-line is unchanged. The effect of the filtering step is to make documents conform to a uniform format while preserving the content and the locations of the table-lines.

The filtered document is then parsed by a parser using the context-free grammar rules described in Appendix C. In essence, the grammar states that a document is comprised of an arbitrary number of tables and non-table regions, and a table region consists of at least three table-lines, which can be separated by up to two blank lines and a ruling line. The parser generates tokens representing table and non-table regions in the document. The first and the last lines of each table-region are identified as table-begin and table-end boundaries respectively, and the lines in between are identified as table-lines. By doing this, the unclassified lines in the first traversal step have another chance to be tagged as table-lines.

As the agent parses an input document, it gathers information such as the location and the length of each input line, the width and the length of the document,

[1]Visually, a blank line and an empty line are the same, as they contain no printed characters. However, an empty consists of a carriage return or the end-of-line non-printable character, whereas a blank line can have any number of space characters before the carriage return or the end-of-line non-printable character. If table rows are separated by blank lines, then each text line is a table row.

and the line indices for the first and the last text lines. This information, together with the locations of the detected tables, is posted on the blackboard by this agent.

Agent #3 - Neighbouring text line identifier
Agent #3 identifies neighbouring text lines. For each text line (that is, a line that is neither a blank line nor a ruling line) in a document, agent #3 identifies its following text lines and posts their separations (in terms of number of lines) on the blackboard. For the document shown in Figure 5.1, agent #3 decides that line 4's following text line is line 6, and their distance is 2. This information, which is encoded in the following two RDF statements, is useful to agents #4 and #8.

```
<line4> <Doc#nextTextLineUri> <line6>;
          <Doc#distanceToNextTextLine> 2.
```

Agent #4 - Non-table line finder based on line length
Agent #4 identifies non-table lines based on the following criteria.

1. A text line is classified as a non-table line if it does not contain three or more consecutive space characters, and its length is at least 90% [2] of the document width.
2. A text line and its immediately following text line are identified as non-table lines if neither contains three or more consecutive space characters, and their lengths are at least 50% [2] of the document width. Furthermore, any blank lines and ruling lines between the text lines are also identified as non-table lines.
3. An unclassified line is identified as a non-table line if its two immediately preceding text lines and its two immediately following text lines are also non-table lines. In other words, an unclassified line is judged to be a non-table line if it is in the middle of a non-table block.

Clearly, there is a form of co-operation between agents #3 and #4, as the findings by agent #3 can be used by agent #4.

Agent #5 - Non-table line finder based on surrounding line types
Agent #5 identifies non-table line based on surrounding input line types. This agent encodes the knowledge that a table-line should not appear in the middle of a non-table block. In particular, if a text line is tagged as a table-line, but its two immediately preceding text lines and its two immediately following text lines are

[2]These thresholds are set after viewing a small number of (about 5) documents from the training data set.

identified as non-table lines, then the agent will disagree with the previous judgement and tag the line as a non-table line.

Agent #6 - Table-line finder based on layout features

Agent #6 implements the layout features and the deterministic algorithm for identifying table lines published by Ng et al. (1999). According to this paper, a text line is classified as a table line if it is not a blank line, and if any of the following conditions is met: the ratio of the position of the first non-space character in the line to the length of the line exceeds some pre-determined threshold (0.25); or the line consists entirely of one special character, where a special character is any character that is neither a space character nor a alphanumeric character; or the line contains three or more segments, each consisting of two or more contiguous space characters; or the line contains two or more segments, each consisting of two or more contiguous separator characters, where a separator character is any of the following characters: '.', '*' or '-'. Since Ng et al.'s paper does not specified how a line should be classified when the conditions are not met, it is assumed that a line is classified as a non-table line (as opposed to leaving it in the unclassified category) when it does not meet any of the conditions. Based on this classification rule, a blank line is always classified as a non-table line. Because the published paper does not specify how to identify table boundaries, it is further assumed that a table is formed by the longest block of adjacent table-lines, and a table block should contain 2 or more table-lines. When deciding on a longest block of table-lines, a single blank line is allowed to appear between two table-lines. Information posted on the blackboard by this agent includes the classification of each input line, the table-begin and table-end boundaries of each table block.

Ng et al. used this information to establish the benchmark in their experiments. In this dissertation, the same information and the same algorithm is implemented as an agent. By applying the agent to the test data discussed in Section 4.5, a benchmark is obtained [3]. The benchmark performance is used to compare the performance of a subroutine like architecture (that is, a single agent not using the blackboard architecture) and the performance of using multiple agents in the proposed blackboard architecture. Additionally, the benchmark performance in this dissertation can also be compared with the benchmark performance published by Ng et al.

[3]During the process of obtaining the benchmark, except for the pre-processing agent (which writes the input document on the blackboard), all other agents were turned off.

5.3.3 Identifying Table Boundaries

Agent #7 - Table boundary identifier based on changes of line types
Agent #7 detects table boundaries based on the change of layout patterns in neighbouring lines and the appearance of certain keywords. In particular, it tests the following conditions to find table-begin and table-end boundaries [4].

1. A text line is a table-begin boundary if it is a table-line, and it is the first text line in the input document.
2. A text line is a table-end boundary if it is a table-line, and it is the last text line in the input document.
3. A text line is a table-begin boundary if it is a table-line, and one of the following two conditions is met: it contains one or more table header keywords (such as *'AUD000'*, *'\$000'*, *'\$thousand'*, *'\$million'* and dates), or one or more keywords appear in its following two text lines.
4. A text line is a table-begin boundary if at least one of the following conditions is met: there are vertically aligned wide space sequences and text sequences between the line and its following text line, or the line and its following text line have the same number of wide space sequences.
5. A text line is a table-begin boundary if it is a table-line, if its preceding two text lines are non-table lines, and if its following two text lines are table-lines.
6. A text line is a table-end boundary if it is a table-line, if its preceding two text lines are table-lines, and if its following two text lines are non-table lines.

Agent #8 - Table-begin boundary identifer based on numeric columns and keywords
Agent #8 identifies table-begin boundaries. It looks for vertically aligned numeric sequences in neighbouring text lines, and the first table-line that contains vertically aligned numeric sequences is identified as table-begin boundary. Additionally, a table-line is identified as a table-begin boundary if it does not contain any numeric sequences, and is directly above a block of table-lines that contain vertically aligned numeric sequences.

Agent #9 - Boundary line balancer
Because table-begin boundaries and table-end boundaries found in the first round do not necessarily appear in pairs (the alternating order) in the second round, agent #9 tries to correct some of the mistakes (if possible). It uses the knowledge that the first table-begin boundary must appear before the first table-end boundary,

[4]These rules were derived from observing some documents in the training set, and by the trial-and-error method.

and that the table-begin and table-end boundaries should appear in an alternating sequence. In particular, the agent takes one of the following actions.

1. For a given document, if the first boundary line is a table-end boundary, then it tries to find a table-begin boundary before the table-end boundary. It achieves this by searching for a table-line such that there is no non-table lines between the table-line and the table-end boundary, and the distance between the table-line and the table-end boundary is maximised.

2. For a given document, if the last boundary line is a table-begin boundary, the agent tries to find a table-end boundary after the table-begin boundary. It achieves this by searching for a table-line such that there are no non-table lines between the table-begin boundary and the table-line, and the distance between the table-begin boundary and the table-line is maximised.

3. If there are two consecutive table-begin boundaries, then the agent tries to find a table-end boundary between the two table-begin boundaries so that there will be matched pairs of table-begin and table-end boundaries. The principles behind this operation are that the table-end boundary must be tagged as a table-line in the previous step (i.e. it cannot be a neutral line or a non-table line); the table marked by the first table-begin boundary and the table-end boundary should not contain any non-table lines; the number of content lines in the table should be maximised; and the number of content lines should not be less than three. Using a similar principles, the agent tries to identify a table-begin boundary between two consecutive table-end boundaries.

It identifies a table-begin boundary between two consecutive table-end boundaries, or removes one of the two consecutive table-end boundaries if deletion is permitted; or it identifies a table-end boundary between two consecutive table-begin boundaries, or removes one of the two consecutive table-begin boundaries if deletion is permitted.

5.3.4 Generating Table Hypotheses

By the time agents # 2 to #9 have finished their discussions, the blackboard contains suggested table-begin and table-end boundaries, although they are not necessarily in matched pairs. Agent #14 constructs table hypotheses based on the proposed boundary lines. It does this by extracting two sets of boundary lines: the first set contains text lines marking the beginning of tables and the second set contains text lines marking the end of tables. Assuming that there are no nested tables, agent #14 builds a set of table hypotheses bounded by a table-begin line from the first set and a table-end line from the second set if the table-begin line

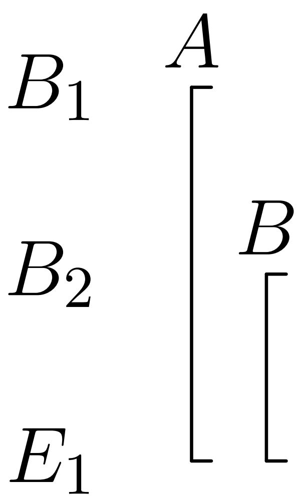


Figure 5.3: Two table-begin lines (B_1 and B_2) and one table-end line (E_1) are proposed. From these boundaries, two table hypotheses can be constructed: table A covering lines from B_1 to E_1, and table B covering lines from B_2 to E_1.

appears before the table-end line. Figure 5.3 illustrates a case where two table-begin lines and one table-end line are identified. In this case, two table hypotheses are identified: table A is bounded by the first table-begin line and the table-end line; and table B is bounded by the second table-begin line and the table-end line. Each table hypothesis is examined by the evaluators, which apply independent sources of knowledge in their evaluations.

5.3.5 Evaluating Table Hypotheses

Agent #10 - Table evaluator based on column structures
Agent #10 evaluates the column structure of a table hypothesis using the hierarchical clustering algorithm and a set of heuristic rules that was published by Hu, Kashi, Lopresti, and Wilfong (2001b, 2001a); Lopresti and Wilfong (2001). The motivation for using this agent is that if a table hypothesis is a genuine table, then it is expected to have a clear column structure; if a table hypothesis does not contain a table (i.e. a block of text lines), then it would not have a clear column structure.

The published algorithm contains two steps in deciding the column structure of a table. In the first step, all words in a table hypothesis are hierarchically clustered based on the following definitions and method.

- A word's position is represented by a vector $p = (s, e)$, where s and e are the word's starting and ending positions in a line.
- The distance between two words, w_1 and w_2, is defined as the Euclidean distance between their corresponding vectors. That is, the distance between w_1 and w_2 is $\sqrt{(s_1 - s_2)^2 + (e_1 - e_2)^2}$.
- The distance between two clusters of words is defined as the average of the distances between all pairs of words from each cluster.
- All words are hierarchically clustered using the following bottom-up method. Firstly, each word belongs to a single cluster, and each cluster is represented by a leaf node in a tree. Secondly, two clusters with the minimum inter-cluster distance are merged into a new cluster. The newly merged cluster is represented by an interior node with the two original clusters as its children. Finally, the process repeats until a single tree is formed.

In the second step, four heuristics are applied to the tree to find table columns. The heuristics can be summarised as the following: a table always contain 2 or more columns; and on average, the distance between adjacent columns should be at least 2 characters, unless there are many small gaps (less than 2 characters) between words from adjacent columns. In the experiments in this dissertation, two adjustments are made to these heuristics. The first adjustment is that there is no guarantee that a table hypothesis contains two or more columns. Given a block of input lines that is known to contain a table, the original algorithm tries to find at least two columns by splitting the root node. However, for a block of input lines that are included in a table hypothesis (not necessarily a table), it is not correct to assume that the hypothesis contains at least two columns. In the implementation, the root node has to go through the same test procedure as the non-root nodes in order to decide whether it should be split or not.

The second adjustment is made to the threshold that defines the distance between columns. According to the published paper, if an inter-cluster gap between the left and the right subtrees for a given node is greater than 2, then the node represents distinct table columns. Setting the threshold to a low level, such as 2, defeats the purpose of the heuristic, which is to separate two columns. Consider the smallest possible inter-cluster gap, where two one-letter words are separated by a single space. The coordinates for the words and the space would be $(i, i), (i + 1, i + 1), (i + 2, i + 2)$, where i is the begin-column index for the first one-letter word. In this case, the inter-cluster gap would be $\sqrt{2^2 + 2^2} = \sqrt{8} > 2$. In other words, the threshold of 2 would cause the smallest possible inter-cluster gap to be split. To overcome this problem, the threshold is set to be 3 times the median word length in a table hypothesis in the experiments in this dissertation.

The reason for setting the threshold to this level is that if a node represents two table columns, then there should be at least one word in each column, and there should be some spaces, which could be the same length as a word, between the columns. The total length of the words in each column and the space between them would be roughly equals to 3 words.

Additionally, the implementation goes one step further after columns are segmented. For each column, it counts the number of words consisting entirely of digits, the number of words consisting of entirely alpha-letters, and the total number of words. Information posted on the blackboard by this agent includes the number of columns in a table hypothesis, and the number of numeric words and the number of non-numeric words in each column.

Agent #11 - Table evaluator based on line correlations
Agent #11 implements the table scoring algorithm published by Hu et al. (2000a). The scores are based on line correlations and are calculated in the following way. First, vertically aligned characters are positively correlated (score 1) if they are both non-leading and non-trailing space characters; they are negatively correlated (score -1) if only one of them is a non-leading and non-trailing space character; and they are not correlated (score 0) if neither case applies. Second, without considering the line distance, the correlation of any two lines is calculated as the sum of scores of all vertically aligned characters. Obviously, the larger the distance between two lines, the less relevant their correlation is. For this reason, the correlation score for any two lines is discounted by their distance. The discount is measured by a distance weight, which decays exponentially as the distance increases. The distance weight is calculated by the following function.

$$\text{Distance weight } = \frac{1}{e^{d*\gamma}},$$

where d is the distance (measured as the number of lines) between two lines, and γ is a factor that controls how fast a correlation decays.

The distance between two adjacent lines is 0, and the distance weight is 1 in this case. The distance weight for lines that are one line apart is $\frac{1}{e^{\gamma}}$, and the distance weight for lines that are two lines apart is $\frac{1}{e^{2*\gamma}}$, and so on. Third, the score for a given a block of lines (line i to line j) is recursively calculated as the larger of the score for appending line j to the block covering lines i to $j-1$, and the score for prepending line i to the block covering lines $i+1$ to j. The score for appending or prepending a line to a block of lines is the score of the block plus the sum of line correlations between the new line and each line in the block. Setting γ to be 1, and without making the algorithm more sophisticated than what was published, agent

#11 implements this scoring algorithm. Overall, this agent scores a table hypothesis based on its column-row structure and the level of data consistency. Information posted on the blackboard by this agent includes a score measuring the correlation of character types among vertically aligned characters in a table hypothesis.

Agent #12 - Table evaluator based on header locations

Agent #12 reports the header information in a table hypothesis. The table processing literature shows that table headers provide useful hints for detecting tables. Certain keywords, such as 'Year', 'Name', month abbreviations, and year strings, are commonly used as the indication for the presence of table headers (see, for example, Embley et al. (2002); Pinto et al. (2003); Wei et al. (2006)), and table headers are usually placed in the beginning or the left hand side of a table (see, for example, Pinto et al. (2003); Wei et al. (2006); Tubbs and Embley (2002)). The motivation for using this agent is that if a table hypothesis contains table headers that are far apart, then it is possible that the hypothesis merges two or more tables together. The following information is written on the blackboard by agent #12.

- the number of header lines. A header line is defined as a line containing at least one sequence of three or more consecutive space characters, and at least one of the following keywords: the dollar sign, the percentage sign, a date, the word 'dollar', 'percentage', 'million', 'thousand', 'current', 'previous', 'quarter', 'year', 'month', or 'quarter' [5].
- the distance between header lines if the number of header lines is greater than one. A table might contain one or more header lines (for example a multi-line header), and if this is the case, the header lines are usually close to each other. If a table hypothesis contains header lines that are far apart, then it is likely that the table merges two smaller tables together.
- the maximum number of wide spaces in a table content line (that is, a text line that is neither a blank line nor a ruling line in a table), and the smallest end-index of the wide space. If the maximum number of wide spaces in a line is low, then the table hypothesis is unlikely to be a column-row structure, and therefore should be rejected.
- the number of table content lines and the percentage of such lines in a table hypothesis. If there are too many lines not containing any wide space, then the table hypothesis is unlikely to be a column-row structure, and therefore should be rejected.

[5] These table header keywords are chosen either because they appear in some documents in the traning set, or because they are reported to be commonly used in table headers in the literature.

Agent #13 - Table evaluator for finding split tables

Agent #13 posts the number of content lines in a table hypothesis on the blackboard. There are agents that inspect a table hypothesis' column-row structure (agents #6 and #10) and header structure (agent #12). However, good results from these agents do not necessarily mean that the hypothesis is perfect, because the hypothesis could be only a part of the target table one wants to detect. For example, if splitting errors occur in a detection process, and only the first half of a table is detected, then the table would still have a good structure and good headers. Agent #13 is introduced to the problem-solving process to penalise under-detected tables. For this purpose, it is not hard to understand that the score returned by this agent should be positively correlated to the number of content lines in a table hypothesis. The following score is used.

$$\text{score for not detecting split tables} = \begin{cases} 0 & \text{if } x <= 3, \\ 100 \times \dfrac{1}{1 + e^{-0.2 \times (x-10)}} & \text{otherwise.} \end{cases}$$

where x is the number of text lines contributing to the table content. This scoring method encourages systems to detect long tables, as a partially detected table (for example, the first half of a table) is not going to be scored as favorably as a full1 table.

In this section, the independent evaluating agents give their opinion about a table hypothesis. In the next step, this information is used by the Final Solution Assembler (agent #14 in Figure 5.2) to judge whether a table hypothesis should be accepted as a table, and whether the table should be included in the final result. Since the implementation of this agent requires a large amount of explanation, it is described in a separate section (Section 5.4).

5.4 Final Solution Assembly

Based on the information produced by the evaluators, the *Final Solution Assembler*'s task is to compile a list of tables as the final result. In order to achieve its goal, the agent first judges whether a table hypothesis is accepted as a table by examining if it has certain table properties. If the hypothesis is considered to be a table by this agent, then the table hypothesis will be included in the candidate solutions set, from which final solutions will be selected in the next step. The rest of the section describes these two steps in detail.

5.4.1 Scoring Table Hypothesis

Scoring table hypotheses is a critical step, because the acceptance and rejection of table hypotheses is based on the scores assigned to tables. Incorrect scoring (assigning a bad table with a high score or assigning a good table with a low score) would result in the acceptance of bad tables or the rejection of good ones. A good scoring system should give low scores to hypothesised tables whose areas do not exactly match those of the expected answers. Two situations should be penalised: (1) a table hypothesis is only a part of an expected answer (i.e. under-detection), and (2) a table hypothesis covers more than an expected answer (i.e. over-detection). Splitting errors (which split tables into two or more smaller tables) is a special case of the first situation, and merging errors (which merge two or more tables together) is a scenario belonging to the second case.

Certain table hypotheses should be rejected due to some obvious non-table characteristics. These characteristics include, for example, that there is only 1 content line or only 1 column in the hypothesis (because a structure is a list if it has only 1 column or 1 row); or there are more than 8 columns (which is not expected) in a hypothesis. These rejection conditions can be encoded as heuristic rules. The majority of the table hypotheses do not contain any of these characteristics, and since there is no algorithm that can precisely specify whether the hypotheses should be accepted, agent #14 *learns* the input/output functionality from examples. Agent #14 combines heuristic rules and learned knowledge when deciding whether a table hypothesis should be accepted or not.

In the previous section, information about a table hypothesis is written on the blackboard by the independent evaluating agents. The information is summarised in Figure 5.4. From this information, agent #14 *learns* the characteristics of tables by applying the J48 decision tree algorithm, which was first proposed by Quinlant (1993), to the training data listed in Table 4.2. The training data produce 640 table hypotheses, of which 539 are non-tables and 101 are tables. The training data and the J48 decision tree algorithm produce the decision tree listed in Figure 5.5.

Feature Symbol	Description
numOfTextLine-NotContainingWideSpace-OverNumOfTableLine	The ratio between the number of text lines not containing wide space and the number of table lines.
maxNumOfWideSpaceIn-TableLine-OverNumOfColumns	The maximum number of wide spaces in a table line over the number of columns.
averageDataType-ConsistencyInColumns	The average score for data type consistency in columns.
numOfLines-ContainingTypicalHeaders	The number of lines containing words that are frequently used in table headers.
distBetweenHeaderLines	The average distance between header lines (note: the average distance between headers is large for merging errors).
correlation-VerticallyAlignedChar	The average correlation score for vertically aligned character in a table hypothesis.
aveDist1stContent-LineToTableHeader	The average distance (measured by the number of lines) between the first content line and the table header.
numbOfContentLines	The number of content lines.
numOfColumns	The number of columns in a table hypothesis.
numOfTextLineNot-ContainingWideSpace	The number of text lines not containing wide spaces.
bestCaseDataType-ConsistencyInColumns	The score for the most data type consistent column.
worstCaseDataType-ConsistencyInColumns	The score for the least data type consistent column.

Figure 5.4: Features for identifying tables

```
aveDist1stContentLineToTableHeader <= 12
|   numOfContentLines <= 5
|   |   correlationVerticallyAlignedChar <= 14: FALSE
|   |   correlationVerticallyAlignedChar > 14
|   |   |   numOfLinesContainingTypicalHeaders <= 1: TRUE
|   |   |   numOfLinesContainingTypicalHeaders > 1: FALSE
|   numOfContentLines > 5
|   |   aveDist1stContentLineToTableHeader <= 2
|   |   |   worstCaseDataTypeConsistencyInColumns <= 0.8
|   |   |   |   numOfTextLineNotContainingWideSpace <= 6: TRUE
|   |   |   |   numOfTextLineNotContainingWideSpace > 6
|   |   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine <= 0.207254: TRUE
|   |   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine > 0.207254: FALSE
|   |   |   worstCaseDataTypeConsistencyInColumns > 0.8
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine <= 0.137931: TRUE
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine > 0.137931
|   |   |   |   |   numOfLinesContainingTypicalHeaders <= 1: FALSE
|   |   |   |   |   numOfLinesContainingTypicalHeaders > 1: TRUE
|   |   aveDist1stContentLineToTableHeader > 2
|   |   |   numOfLinesContainingTypicalHeaders <= 2
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine <= 0.129032
|   |   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine <= 0.117647
|   |   |   |   |   |   correlationVerticallyAlignedChar <= -2256: FALSE
|   |   |   |   |   |   correlationVerticallyAlignedChar > -2256: TRUE
|   |   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine > 0.117647: FALSE
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine > 0.129032: TRUE
|   |   |   numOfLinesContainingTypicalHeaders > 2: FALSE
aveDist1stContentLineToTableHeader > 12
|   distBetweenHeaderLines <= 13
|   |   numOfLinesContainingTypicalHeaders <= 1: TRUE
|   |   numOfLinesContainingTypicalHeaders > 1
|   |   |   distBetweenHeaderLines <= 12: FALSE
|   |   |   distBetweenHeaderLines > 12
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine <= 0.191011: TRUE
|   |   |   |   numOfTextLineNotContainingWideSpaceOverNumOfTableLine > 0.191011: FALSE
|   distBetweenHeaderLines > 13: FALSE
```

Figure 5.5: The learnt decision tree for accepting/rejecting table hypotheses.

To test the effectiveness of the decision, 7 test documents, including 6 that are randomly chosen from test sets #1 discussed in Section 4.5 and 1 randomly chosen from test set #2, are used. This test case selection method ensures that at least 1 test document contains long tables, while keeping the overall test cases relatively small. For each document, the line indices for all ruling lines and lines containing non-leading wide spaces (two or more consecutive space characters) are recorded. Test cases, spanning over the recorded line indices, are generated in such a way that at least 50% of the lines are included in the test cases. Using this method, test cases will include merged tables, split tables, whole tables, and non-table lines. The goal is to use these test cases to test the decision tree and see how well it rejects non-tables and how well it accepts good tables. 508 test cases are generated from 7 test documents. Evaluation shows that the decision tree correctly rejects 342 non-tables; incorrectly accepted 24 non-tables; correctly accepted 106 tables, which include 5 complete tables and 101 partial tables; and incorrectly rejects 36 tables, which include 2 complete tables and 34 partial tables. The results, which are summarised in Table 5.1, show that the decision tree is good at rejecting non-tables and accepting (partial) tables. However, it is not very good at differentiating partial tables from full tables.

Decision tree result	Num of Non-tables	Num of complete tables	Num of partial (proper subset of) tables
Accept	24	5	101
Reject	342	2	34

Table 5.1: The effectiveness of the decision tree in Figure 5.5.

The decision tree is applied to table hypotheses that are not rejected by the heuristic rules discussed above. If a hypothesis is accepted as a table according to the decision tree, then the hypothesis is assigned with a score that is equal to the number of content lines it has. Otherwise, it is assigned with an arbitrary low score (say -1000). The reason for using the number of content lines as the score for an accepted hypothesis is that this measure, in conjunction with the decision tree, can help to avoid under-detection and over-detection. Suppose that a table contains n lines, and there are three hypotheses (one is under-detection, one is perfect detection, and one is over-detection) competing to be chosen as the final answer. The over-detection scenario should be rejected by the decision tree, as the over-detected parts cause the hypothesis to have a bad structure. The decision tree cannot differentiate under-detection from perfect detection, as they both have good table structures. However, the perfect detection scenario has a higher number of content lines than the under-detection scenario. Therefore, using the number of content lines as the score can help the perfect detection scenario to be chosen as the final answer. After filtering out non-tables from the candidate solution set, the next step is to go through an optimisation process for selecting the final result.

5.4.2 Assembling Final Solutions

Final solutions are chosen from the candidate solution set, and the table hypothesis acceptance process can be summarised as maximising the sum of scores (i.e. the total number of content lines) of all accepted tables, subject to the condition that the accepted tables do not overlap. The the sum of scores, in this dissertation, is maximised by a global optimisation algorithm, which is described below.

5.4.2.1 Global Optimisation

To show how the global optimisation process works, let's first look at a simple example illustrated by Figure 5.3. In this example, the two table hypotheses cannot both be accepted at the same time because they overlap. If table A has a score of 70, and table B has a score of 60, then table A should be accepted and table B should be rejected.

Figure 5.6 illustrates a more complex example, where the table with the highest score should not be accepted. In this figure, two table-begin lines and three table-end lines are identified. These lines produce five table hypotheses: table A covering lines from B_1 to E_3; table B covering lines from B_1 to E_2; table C covering lines from B_1 to E_1; table D covering lines from B_2 to E_3; and table E covering lines from B_2 to E_2. Suppose that the decision tree has rejected table A, in this case, table A will have a negative score of -1000. Let's further suppose that the decision

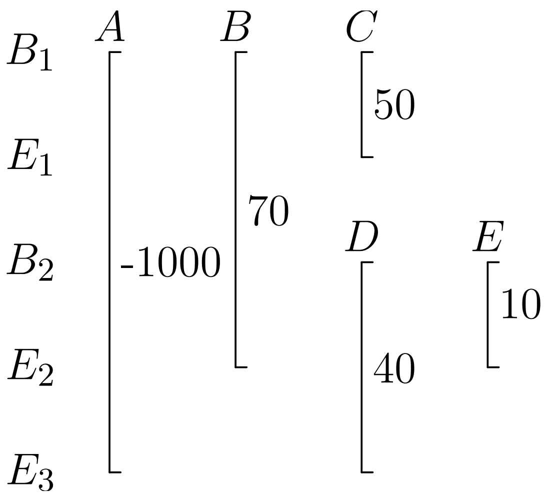


Figure 5.6: Two table-begin lines (B_1 and B_2) and three table-end lines (E_1, E_2 and E_2) are proposed. From these boundaries, five table hypotheses (table A covering lines from B_1 to E_3; table B covering lines from B_1 to E_2; table C covering lines from B_1 to E_1; table D covering lines from B_2 to E_3; and table E covering lines from B_2 to E_2) are competing to be included in the final result.

accepts all other tables, and tables B, C, D and E's scores are 70, 50, 40 and 10 respectively. Although table B has the highest score, it should not be accepted because its score is less than the total score of tables C and D. By accepting table D, tables A and E should be rejected, because they each overlaps with table D.

In this dissertation, the global optimisation algorithm is implemented by transforming the problem of finding the highest sum of scores to the problem of finding the maximum weight of a path in a weighted directed acyclic graph (weighted DAG). Here, the weight of a path is calculated as the sum of the weights of the selected edges. The global optimisation algorithm creates a DAG representing a list of table hypotheses using the following steps.

1. It creates a vertex for each table hypothesis that has a positive score. In addition to this, it also creates one artificial vertex, called a *source*.
2. The source is a vertex that has no incoming edges in the DAG. However, the algorithm creates an edge originating from the source to every other vertex representing a table hypothesis.
3. It creates edges only connecting non-source vertices representing non-overlapping tables. Each of these edges originates from the vertex with the earlier starting

position and terminates at the vertex with the latter.

4. Weights are assigned to edges. The weight for each edge is equal to the score assigned to the table represented by the vertex at which that edge terminates.

A DAG has at least one *sink*. A sink is a vertex that has no outgoing edges. The table hypotheses in Figure 5.6 are represented by the DAG in Figure 5.7. In this DAG, there are three sinks: vertices B, D and E. The final accepted list of hypotheses always includes a table represented by a sink. To find the maximum weight from the source to a sink in a weighted DAG, it is necessary to calculate the maximum accumulated weight for each vertex. The maximum accumulated weight for a vertex is calculated as follows: the accumulated weight for the source vertex is 0. For a non-source vertex, the accumulated weight is equal to the maximum accumulated weight of the vertices that link to it plus a weight for the incoming edge. Formally, let v be a vertex, let W_v be the set of vertices that have outgoing edges terminating at v, and let e_v be the weight of the incoming edges of v. Then the maximum accumulated weight for v, denoted by $maxWt(v)$, is calculated as follows:

$$maxWt(v) = \begin{cases} 0 & \text{if } v \text{ is the source vertex} \\ \max\limits_{w \in W_v} \{maxWt(w)\} + e_v & \text{otherwise} \end{cases}$$

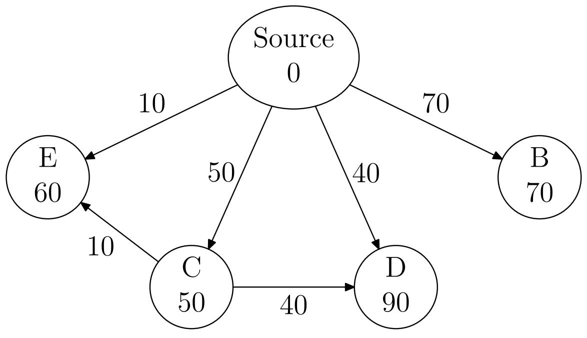


Figure 5.7: A DAG representing tables hypotheses in Figure 5.6. The accumulated weight for each table hypothesis is represented by the number written in each vertex. The vertices that are on the path with maximum weight correspond to the accepted table hypotheses. In this DAG, vertex D has the highest accumulated weight (90), and the path involves from source to vertex C, then vertex D. So tables C and D should be accepted.

5.4.2.2 Efficiency Analysis for the Global Optimisation

The number of table hypotheses is determined by the number of proposed table-begin and table-end lines and their positions. Let b and e be the number of proposed table-begin lines and table-end lines respectively. The minimum number of table hypotheses is 0 when each table-end line appears before any table-begin line; the maximum number of table hypotheses is $b \times e$ when each table-begin line appears before any table-end line; and in a special case where $b = e$, and the boundary lines follow the 'a table-begin line followed by a table-end line' alternating pattern, the number of table hypotheses is $\sum_{i=1}^{b} i = \frac{b(b+1)}{2}$.

In the case where each table-begin line appears before any table-end line, any pair of tables overlaps with each other. Only the hypothesized table with the highest score should be chosen in the final result; the rest of the table hypotheses should be rejected. In general, finding the maximum weight in a weighted graph is an NP-complete problem. However, because the graph is a DAG, the maximum accumulated weight can be calculated efficiently using a dynamic programming technique. The run time is $O(n^2)$, with n being the number of table hypotheses. To reduce the run time complexity, a local optimisation algorithm can also be used as an approximation of the global optimisation algorithm.

5.4.2.3 Local Optimisation

The local optimisation algorithm works by first sorting all the table hypotheses in descending order based on their scores. It then goes into the iteration process of accepting the first table (that is the table with the highest score) and rejecting all table hypotheses that overlap with the accepted table. After the iteration, a table hypothesis is either accepted or rejected. For the hypotheses listed in Figure 5.6, the local optimisation algorithm would first accept table B, because it is the hypothesis with the highest score. By accepting table B, all other table hypotheses will be rejected, because they each overlaps with table B.

The local optimisation guarantees that the table with the maximum score is chosen, and the global optimisation guarantees that the sums of scores are maximised. These become the two possible work modes for agent #14. Each of these work modes is evaluated, and the evaluation results show that there is no significant difference (in terms of the overall performance) between these two work modes.

At the end of a table identification process, the blackboard contains the information about the matched pair of table boundaries lines and the table content lines within each pair of boundaries. This information can be extracted from the blackboard and re-written to meet the XML specification discussed in Section 4.4.3.

5.5 Conflict Resolvers

Agent #15 - Sub-property inference agent

Conflicting statements can be produced by different agents during the problem solving process. For example, if a text line is said to be a table-line and a non-table line at the same time, or if a text line is identified as a table-begin boundary and a non-table line at the same time, then there are conflicts. In the first case, contradiction arises because a line is either in a table or not in a table. In the second case, contradiction arises because a line that marks the beginning of a table implies that the line is a table-line, and a line cannot be a table-line and a non-table line at the same time. Inference can help to reveal subtle contradictions.

Agent #15 assists the conflict resolver by inferring new statements based on the sub-class or sub-property relationships. In particular, it contributes new RDF statements based on the following inference rules using the vocabulary provided in Appendix I.

1. The *Doc:BeginningOfTableRegion* class is implied by any of its sub-classes, which are listed below.
   Doc:BeginningOfTable_atOrNearTableLnContaingTypicalHeaderStr,
   Doc:BeginningOfTable_changeOfLineType,
   Doc:BeginningOfTable_firstTextLineInDoc,
   Doc:BeginningOfTable_longestTableBlockBetweenTwoTableEndRegions,
   Doc:BeginningOfTable_firstTableBoundary,
   Doc:BeginningOfTable_1stLineInNumericColumn and
   Doc:BeginningOfTable_grammarRules.
2. The *Doc:EndOfTableRegion* class is implied by any of its sub-classes, which are listed below.
   Doc:EndOfTable_changeOfLineType,
   Doc:EndOfTable_lastTextLineInDoc,
   Doc:EndOfTable_longestTableBlockBetweenTwoTableBeginRegions,
   Doc:EndOfTable_lastTableBoundary,
   Doc:EndOfTable_lastLineInNumericColumn and
   Doc:EndOfTable_grammarRules.
3. The *Doc:NonTableLine* class is implied by any of its sub-classes, which are listed below.

Doc:NonTableLine_betweenLongTextLines,
Doc:NonTableLine_singleLongText,
Doc:NonTableLine_consecutiveLongText,
Doc:NonTableLine_noTableColumnDelimiterAndBetweenNonTableLines,
Doc:NonTableLine_hasForbiddenChar and
Doc:NonTableLine_precedingAndFollowingLinesAreNonTableLine.

4. The *Doc:TableLine* class is implied by two of its sub-classes, which are *Doc:TableLine_hasTableColumnDelimiter* and *Doc:TableLine_fallWithinTableRegion*.

As an example, agent #15 generates the the RDF statement of (⟨*file:110730.gsml#line45*⟩, *a, Doc:NonTableLine*) if it sees the following RDF statement from the blackboard.
(⟨*file:110730.gsml#line45*⟩, *a, Doc:NonTableLine_consecutiveLongText*)

Agent #16 - Conflict resolver
Conflicting opinions could arise among agents during a problem-solving process. After agent #15 finishes running, conflicts can be seen in two situations: a text line is identified as a table-line and a non-table line at the same time; and a text line is identified as a table-begin boundary and a table-end boundary at the same time. Conflicts are resolved by agent #16 in the process. The blackboard is given to this agent every time a group of agents finish discussing a problem. The reason this agent accesses the blackboard so often is because conflicts should be resolved as early as possible, otherwise, errors could be propagated into the remaining steps and cause more errors. The details of conflict resolution are discussed in Section 3.5.

5.6 Evaluation and Performance Analysis

In order to test the feasibility of the blackboard architecture and the effect of employing multiple agents to the problem solving process, three experiments are conducted. The first one is to create a benchmark by running only agent #6. The second and the third experiments allow multiple agents to run in the blackboard environment. The difference between the second and the third experiments is the work mode of the Final Solution Assembler (agent #14 in Figure 5.2). In the second experiment, agent #14 uses the global optimisation algorithm to maximise the total score of the accepted tables, whereas in the third experiment, agent #14 uses the local optimisation algorithm to maximise the total score.

The test data are the 2 test sets discussed in Section 4.5. The test data contain 352 documents, of which 275 do not contain any tables, and the rest of the 77 documents contains 170 tables, which include 1988 table rows and 7069 table cells. Appendix K lists the experimental results, and for clarity reasons the % area of bounding boxes being detected correctly at the table level is included in Table 5.2.

Agents involved	Optimisation method used by agent #14	Overall performance
The benchmark result produced by the deterministic method published by Ng et al. (1999)	NA	7.63%
All, but the benchmark, agents	global	64.95%
All, but the benchmark, agents	local	64.38%
All agents (including the benchmark agent)	global	70.82%
All agents (including the benchmark agent)	local	74.18%

Table 5.2: Percentage of areas that is correctly detected at table-level for the experimental results in Appendix K

A number of conclusions can be drawn from the experimental results. Firstly, test data have direct influence on the performance of table layout analysis systems. When tested by WSJ, the benchmark algorithm's overall result was about 70% (see Ng et al. (1999)). However, when tested by the test data used in this dissertation, the same algorithm has an overall result of 7.63%. Since the original WSJ test data is not available, detailed analysis on the errors cannot be performed. However, Appendix L provides some insights of the errors made by the benchmark algorithm on the ASX corpus.

Secondly, there is a significant advantage of using a diverse set of knowledge sources. By comparing the f-measures in Figure K.1 and the f-measures in other experiments listed in Appendix K, one can see that using only layout information, which is what the benchmark system does, produces an overall accuracy of only 7.63%. However, the performance is increased substantially (over 65%) when using both layout and language information. By comparing the f-measures in Figures K.2 and K.3 against the f-measures in Figures K.4 and K.5, one can see that although the benchmark agent produces only 7.63% accuracy, there is still an advantage of including this low performance agent, as adding it to the system can boost the performance from about 65% to over 70%.

Finally, there is no significant difference between using the global optimisation algorithm and the local optimisation algorithm in the final solution assembling step. By comparing the f-measure in Figure K.2 against the f-measure in Figure K.3, and the f-measure in Figure K.4 against the f-measure in Figure K.5, one can see that there is less than 5% difference in using the global optimisation algorithm and the local optimisation algorithm.

Given a question such as 'is this a table?', the answers could be positive, negative, or unknown (that is, neither positive nor negative). If this question is presented to multiple programs, and each program uses different sources of evidence to derive their results, then the relationships between any pair of conclusions must be one of the following: reinforcing (the two results are the same), conflicting (one has produced a positive result and the other has produced a negative result), or non-conflicting (one has produced a positive or a negative result, and the other does not know how to classify the lines). Complementary results can improve accuracies, because a table that is missed by one program would be picked up by another. Reinforcing results do not hurt performance. This is because they do not change the result; if a table is detected correctly by both programs, then it is still detected. If a table cannot be detected by either program, then it is still not detected. Conflicting opinions can create confusion. However, because the architecture was able to resolve conflicts relatively effectively, this drawback had limited effect. Overall, the experiments in this chapter show that multiple agents running within the blackboard framework can find more credible answers than a single agent could.

Although agents are independently designed, they co-operate with each other in achieving their common goals. For example, mutual co-operations are seen among agents when classifying input lines in Figure 5.8. Initially, agent #2 classifies line 1 as a non-table line, because it contains characters that are unlikely to appear in tables; line 2 as a table-line because it contains three or more space characters; and all other lines are left in the unclassified category, because the agent does not have any evidence to make a decision. When the process progresses to sub-task 1, the blackboard does not contain immediate information for agent #5 to be activated. However, agent #4 can contribute new information, as it decides that lines 3, 5, 6 and 7 are non-table lines due to their lengths. This new information activates agent #5 in the next round of blackboard passing. With the new information, agent #5 can decide that line 2 is a non-table line, because its preceding line and its following lines are non-table lines. That is, agent #5 disagrees with agent #2's decision. This disagreement will be resolved in the conflict resolution step later. But for now, the conflicting opinions are written on the blackboard. The output by agent #5 becomes a new feedback to agent #4, which further decides that line 4 is a non-table line, because it believes that an unclassified line should be moved to the non-table category if its two preceding lines and its two following lines are non-table lines. After this discovery, no agent can make any new contribution, and the discussion phase terminates. The net result is that with the exception of line 2 (which is classified as both table-line and non-table line), all other lines are

```
    <Text><![CDATA[
Canada Land Limited                             2004-06-11  ASX-SIGNAL-G
Canada Land today released its preliminary final report.
HOMEX - Sydney
The company said the revenue from ordinary activities was up 0.89 from
previous corresponding period (pcp) to $3,272,000 . The net profit was
up 28.38% from pcp to $9,264,000. No final dividend was declared.
```

Figure 5.8: Agent co-operations when classifying lines

classified as non-table lines.

5.7 Summary

This chapter presents a framework that allows heterogeneous table processing algorithms to work co-operatively towards the goal of improving a general table analysis task. Through experimentation, not only the feasibility of the framework, but also the benefits (which are discussed in Section 3.2) arising from its use have been confirmed. In particular, the most fundamental advantage, which is not having to schedule agents when they are added to the problem solving process, is confirmed. Adding or removing agents in the experiments is easy under the framework due to two factors. The first factor is that the blackboard provides a central place to hold the shared data. Any agent can use the data created by previous agents, and any agent can write information on the blackboard without knowing which agents will use the data. When an agent is added to a discussion phase, the round-robin scheduler guarantees that the blackboard will be passed on to the new agent. The second factor is that when an agent is removed from a discussion phase, the scheduler simply de-registers the agent, and the system does not need to worry about whether other agents will be affected.

One of the conclusions reached by Erman et al. (1980) through the Hearsay II project was that finding solutions to problems that involved ambiguities, noisy data, and imperfect or incomplete information often required simultaneously combining multiple kinds of knowledge, and the combined effect of several knowledge sources could often identify the single most credible solution. This conclusion is in agreement with the experiments discussed in this chapter, as several experiments allowing subsets of the agents to participate were conducted, and the results show that the best performance is the one that involves all agents. For example, agents #2 and #14 together can detect 65.54% of the tables (see Table K.6 for details); agents #4, and #7 cannot detect any table alone because agent #4 identifies non-table lines only, but they can detect 68.23% of the table areas when they work together with agent #6 (see Table K.7 for details). Overall, it is possible to improve table analysis performance by incorporating diverse knowledge sources.

6 Applying the Blackboard Framework To Table Interpretation Tasks

6.1 Introduction

The ultimate goal of table processing is to get machines to understand tables, because being able to do so makes other higher level data processing tasks possible, such as building question-answering applications and populating databases with correct results. Chapter 3 shows how the RDF-based blackboard framework could be used for improving table structural analysis. This chapter shows that the RDF-based blackboard framework can be used to support table interpretation tasks.

Tables are widely used in many domains, and the domain of a table has direct influence on how easily a table can be interpreted, as not all tables can be interpreted using only general knowledge. For instance, the Periodic Table of the Elements would require special knowledge to understand. Figure 6.1, which is extracted from a real document, gives another example that understanding certain tables require domain specific knowledge.

As was discussed in Section 2.4.3, the table processing literature contains many forms of 'table interpretation'. In this dissertation, the definition of 'table interpretation' is the ability to do the following:

- to express any relations between any table components (such as rows, columns and cells) in the same table.
- to express any relations between table components in different tables. This includes tables in the same document, and tables in different documents.
- to express any relations between table components and non-table components (i.e. text outside tables).

```
Results include:
J16  2.2m   @   1.03% Cu and from 219.8m
     2.0m   @   0.63% Cu and from 393.5m
J24  6.0m   @   0.87% Cu and from 72m
J25 10.24m  @   1.32% Cu and from 563.8m
J26  5.2m   @   3.29% Cu and from 179.8m (eastern zone)
J27  3.0m   @   0.75% Cu and from 110m
     2.5m   @   1.0% Cu and from 246.5m
    10.0m   @   1.40% Cu and from 264m
J28  8.4m   @   1.22% Cu and from 465m
J29  No assay data available.
```

Figure 6.1: An example showing that table interpretation requires domain knowledge.

Using identifying summation calculations in tables as an example, this chapter shows how a table interpretation goal is achieved under the RDF-based blackboard framework. In this experiment discussed in this chapter, the input data are tables, and the output data are summation calculations found in the tables. For example, Figure 6.2, which is extracted from the ASX corpus, contains the following summation calculations.

- row 4.6 is the sum of rows 4.1 to 4.5;
- row 4.15 is the sum of rows 4.7 to 4.14;
- row 4.16 is the sum of rows 4.6 and 4.15;
- row 4.21 is the sum of rows 4.17 to 4.20;
- row 4.26 is the sum of rows 4.22 to 4.25;
- row 4.27 is the sum of rows 4.26 and 4.21;
- row 4.32 is the sum of rows 4.29 to 4.31;
- row 4.34 is the sum of rows 4.32 to 4.33.

The calculations in this table illustrate the challenges in identifying summation calculations as follows: the number of calculations in a table is unknown; the location of the calculations in a table is unknown; the number of operands in a calculation is unknown; the operands included in a calculation are not necessarily adjacent to each other; some numeric data in a table does not participate in a calculation while other data could participate in more than one calculation; calculations could contain errors (including rounding errors); and the layout of calculations (listing sub-terms before showing the sum, or the other way around) is unknown.

		At end of current period AUD000	As in last annual report AUD000	As in last half yearly report AUD000
	CURRENT ASSETS			
4.1	Cash	1,167	1,214	1,104
4.2	Receivables	6	-	-
4.3	Investments	-	-	-
4.4	Inventories	-	-	-
4.5	Other (provide details if material)	-	-	-
4.6	Total current assets	1,173	1,214	1,104
	NON-CURRENT ASSETS			
4.7	Receivables	-	-	-
4.8	Investments	-	-	-
4.9	Inventories	-	-	-
4.10	Exploration and evaluation expenditure capitalised	-	-	-
4.11	Development properties (mining entities)	-	-	-
4.12	Other property, plant and equipment (net)	-	-	-
4.13	Intangibles (net)	-	-	-
4.14	Other (provide details if material)	-	-	-
4.15	Total non-current assets	0	0	0
4.16	Total assets	1,173	1,214	1,104
	CURRENT LIABILITIES			
4.17	Accounts payable	5	28	15
4.18	Borrowings	-	-	-
4.19	Provisions	-	-	-
4.20	Other (provide details if material)	-	-	-
4.21	Total current liabilities	5	28	15
	NON-CURRENT LIABILITIES			
4.22	Accounts payable	-	-	-
4.23	Borrowings	-	-	-
4.24	Provisions	-	-	-
4.25	Other (provide details if material)	-	-	-
4.26	Total non-current liabilities	0	0	0
4.27	Total liabilities	5	28	15
4.28	Net assets	1,168	1,186	1,089
	EQUITY			
4.29	Capital	806	806	806
4.30	Reserves	321	321	321
4.31	Retained profits (accumulated losses)	41	59	(38)
4.32	Equity attributable to members of the parent entity	1,168	1,186	1,089
4.33	Outside equity interests in controlled entities	-	-	-
4.34	Total equity	1,168	1,186	1,089

Figure 6.2: A balance sheet table

While it might be sufficient to convey information as an English-language sentence, the statement *'row 4.6 is the sum of rows 4.1 to 4.5'* is somewhat imprecise, because not every cell participates in calculations. For example, the two left-most cells in row 4.6 (the cell with the string of '4.6' and its right neighbouring cell with the string of 'Total current assets') do not participate in any calculation. The task in this chapter is to identify summation calculations that meet the following conditions.

1. Calculations have the top-down layout, which lists the operands that contribute to the total before listing their sums.
2. The operands that involved in a calculation must exist in the tables.
3. There must be linguistic evidence or domain knowledge to support the existence of the identified calculations. This requirement prevents finding unrelated data from tables that happen to make up a calculation.
4. For a calculation to be considered as identified, the sum, and the operands that make up of the sum, must be identified. Sums and operands are referenced by the table cell URIs.

6.2 Vocabulary for Summation Calculations

Before conducting the experiments, a vocabulary must be defined to support the task. A good design of vocabulary provides a complete, yet not redundant, set of concepts allowing agents to make statements about a particular domain. Unlike table layout analysis, which uses a fixed vocabulary for referring to the geometric locations of table components, table interpretation tasks use domain dependent vocabulary. In the experiment of finding summation calculations, the vocabulary contains arithmetic concepts such as *plus*, *total* and *sum of*. The full list of the vocabulary is provided in Appendix I.2.

To express a summation calculation such as *'cell(3, 8) is the sum of cell(3,3), cell(3,4), cell(3,5) cell(3,6) and cell(3,7)'*, three simple RDF statements are used. The first one states the sum relationship between the total line and an unnamed object; the second one states that the unnamed object is a bag of operands; and the third one enumerates the operands in the unnamed bag. Syntactically, these three statements can be combined into one statement. The above calculation can be represented by the following RDF statement, which graph representation is shown in Figure 6.3.

```
<cell(3, 8)> <Arithmetic#Sum>
    <bag(cell(3,3), cell(3,4), cell(3,5), cell(3,6), cell(3,7))>.
```

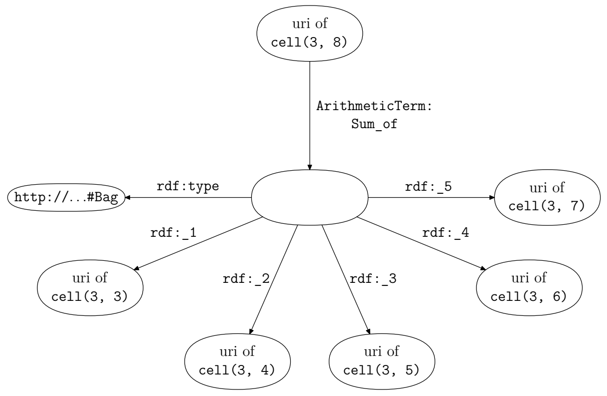


Figure 6.3: RDF statements representing a summation calculation

6.3 Identifying Summation Calculations

In order to identify summation calculations, the following three sub-tasks have been identified. The first one is to differentiate numeric and non-numeric data in tables. This is the most basic requirement for the task, as non-numeric data cannot participate in calculations. The second sub-task is to recognise index columns. The need for recognising index columns is because indices are often listed in numeric format, but they do not participate in calculations. The final task is to recognise summation calculations in tables, as this is the goal of the experiment, which is to illustrate the blackboard framework's ability to support table interpretation.

Similar to the table boundary identification task discussed in Chapter 5, the sub-tasks in this chapter are achieved by agents running within the blackboard framework. There are three agents involved in finding the summation calculations. The rest of the section describes the agents involved in finding summation calculations in tables.

6.3.1 Numeric String Recogniser

The task of differentiating numeric and non-numeric data in tables is achieved by the *Numeric String Recogniser* agent. Recognising numeric data is a complex task because of the large number of data types involved: date, time, price, volume, percentage, speed, temperature, pressure, weight, indices, exchange rate, special code, identification numbers (such as flight number, part number, reference number, and telephone numbers), binary numbers (0 and 1), special names (such as 401(K)), address and post code (such as 2108), Internet Protocol address (such as 123.13.237.104), and business names (such as 7-11) are just some of the examples. The *Numeric String Recogniser* agent used in this experiment interprets three specific types of numeric data: numeric strings surrounded by brackets, numeric strings meaning 'void' or '0', and comma-separated numeric strings. The correct recognition of each of these data types requires disambiguation.

Brackets are used for many purposes: in arithmetic expressions, brackets are used to define the order of the operations; in written English-language text, brackets can be used to provide additional information - as in the string 'Development properties (mining entities)' in row 4.11 of Figure 6.2, or they can direct readers to related information; in the financial domain, brackets are often used to denote negative numbers. To interpret brackets surrounding numbers, the *Numeric String Recogniser* agent makes use of domain knowledge and information in related table cells. For example, using the accounting domain knowledge, the *Numeric String Recogniser* agent treats brackets surrounding numbers as negative signs. Using this knowledge, (100) would be interpreted as -100, and (-100) would be interpreted as 100. However, not all numbers enclosed in brackets are negative numbers. For example, the last cell in row 4.31 of Figure 6.2 contains a number in a bracket, and this number should be interpreted as 'accumulated losses is 38'. The reason for this interpretation is because the row header for this cell contains a pair of antonyms - 'retained profits' and 'accumulated losses', with the 'accumulated losses' being surrounded by a pair of brackets. This information gives the following hints: in this column, numbers that are surrounded by brackets in this column should be interpreted as 'accumulated losses', otherwise, they should be interpreted as 'retained profits'. This example shows how table headers could provide information on the interpretation of related data cells.

As can be seen from Figure 6.2, the '-' character is often used. The '-' character has many usages: as a negative sign; as a minus operation in arithmetic expressions; as a dash to provide further explanation; or to denote information not available; or to denote that the value is 0. In this experiment, the agent interprets the '-' character and the strings of 'NA' and 'N/A' as '0' if other cells in the same column/row are numbers.

Similar to the ambiguities associated with the '-' character, ambiguities exist in using the comma symbol. The comma symbol has two main usages: it can be used as a punctuation device in written text, or it can be used as a thousand separator in numbers. To work out the correct meaning of the comma symbol, the *Numeric String Recogniser* agent uses the regular expressions listed in Appendix A to detect numeric data. If the string in a table cell is accepted by any of the regular expressions, then the string type is recognised.

Given a table, the *Numeric String Recogniser* agent goes through every table cell, and for those table cells that contain strings representing numbers, it normalises the strings to a common format by removing the comma separators, turning numbers in brackets to the appropriate values, and replacing the '-' character with the number 0.

6.3.2 Table Index Recogniser

Indices are often used in tables to provide clarity and convenience for referencing purposes. Indices can appear in different forms such as 1, 2, 3 . . . ; (1), (2), (3) . . . ; [1], [2], [3] . . . ; A, B, C . . . ; a, b, c . . . ; 1.1, 1.2, 1.3 . . . ; I, II, III, IV . . . ; 1998, 1999, 2001, 2001 . . . . When indices appear in numeric forms (as in the ones shown in Figure 6.2), they have the potential to complicate calculation detections. Although syntactically they are numbers (the same data type as the ones participating in calculations), indices do not participate in calculations. This creates the need for identifying indices.

The *Table Index Recogniser* agent finds indices, and the information that the agent relies on is as follows. An index is a series of data that meets both the geometric requirement and the arithmetic progression requirement. The geometric requirement specifies that the column/row under examination must be in the beginning part of a table. Although index columns/rows are often placed in the first column/row, exceptions have been seen. The arithmetic progression requirement specifies that the table cells in the same index column/row form a sequence, and the difference between any two successive members in the sequence is a positive constant.

6.3.3 Arithmetic Calculation Recogniser

This agent makes use of linguistic information to find the summation calculations in tables. One way (the inefficient way) of finding summation calculations is to try every possible combination and check if the arithmetic 'sum' relationship holds. This method has an excessive run time, because the number of operands involved in a summation calculation is unknown: it could be as small as 2, or it could be as large as the number of columns/rows in a table minus 1. For example, for Figure 6.2, the trial-and-error process would look for summation calculations in the following way: from the third row to the last row in the table, the algorithm exhaustively checks if the row equals the sum of the previous two, three, four . . . rows. Note that the rows to which the operands belong are not necessarily adjacent to each other.

Suppose that a table has n rows, and the table contains one or more summation calculations. In this case, the number of operands between 2 and $n-1$ inclusive. The reason for the number of operands cannot be greater than $n-1$ is because at least one table row is used for hosting the sum. Let the lines in this n-row table be indexed from 1 to n. The brute-force method test whether the 3^{rd} row is a sum. There are only 2 rows above the sum under test, so the method checks if the first two rows add up to the 3^{rd} row. The method moves on to test whether the 4^{th} row is a sum. This time, there are 3 rows before the sum under test, and the number of operands could be 2 or 3. For the case where the number of operands equals 2, the method has to find which 2 rows add up to the 4^{th} row; it has to check 3 combinations - the 1^{st} and the 2^{nd} rows, the 1^{st} and the 3^{rd} rows, and the 2^{nd} and the 3^{rd} rows. After testing whether the 4^{th} row is a sum or not, the method moves on to test the rest of the rows in the table. In general, let n be the number of rows in a table. Assuming that a sum is listed either before or after all the operands, and there is no error in the input table, then the number of checks, in the worst case, is the following.

$$\sum_{i=2}^{n-1} \sum_{j=i+1}^{n} \binom{j-1}{i}$$

In this formula, i enumerates the number of possible operands, and j enumerates the number of possible locations for the sums. For a summation calculation that has i operands, the sum can appear anywhere between the $(i+1)^{th}$ row to the n^{th} row. In order to check whether the j^{th} row is the sum of previous i operands $(i < j)$, there are $\binom{j-1}{i}$ tests need to be performed.

The above method assumes that there is no error in the input document. When a table contains errors, such as transposition errors, this trial-and-error method cannot detect sum calculations at all. A more efficient way of finding summation calculations is to use linguistic information. The words *total* and *sum of* provide hints as to the location of the summation calculations. There are two categories describing how these language hints can provide information about summation calculations. The first category describes summing pairs of complementary sets. Two sets are complementary if any object in a particular domain belongs to only one of the two sets. For example, ‘students’ and ‘non-students’ are two complementary sets, but ‘students’ and ‘politicians’ are not, because there are individuals (such as ‘teachers’) who do not belong to either of these two sets, and there are individuals who belong to both of these sets.

Figure 6.2 contains two complementary sets: ‘current assets’, and ‘non-current assets’; ‘current liabilities’ and ‘non-current liabilities’. Generally, the linguistic hint for finding labels denoting complementary sets is that the labels differ by the word ‘non’. The linguistic hint for finding the summation calculations of complementary items is the pattern of having the word ‘total’, optionally followed by the name of the domain. For example, the string ‘total liabilities’ hints that the underlying data is the sum of ‘current liabilities’ and ‘non-current liabilities’. Part of the *Arithmetic Calculation Recogniser* makes use of these language hints to find summation calculations. As a side-effect, the agent can detect the incorrect use of terminology if underlying complementary items do not add up to their sums.

The second category describes summing non-complementary sets. The linguistic characteristic of this category is that the arithmetic relationship contains the word ‘total’, but the word ‘non’ is not present. For example, the row ‘total current assets’ in Figure 6.2 is the sum of the previous 5 rows. Finding this type of summation arithmetic expressions can be viewed as an optimisation problem: the goal is to minimise the physical distances between some data cells in a table, subject to the conditions that (1) the total cell is the sum of all data cells, and (2) the data cells are physically contiguous, and they are either above or below to the total cell.

6.4 Evaluation for Sum Identification

Evaluation is conducted in two steps: to report the effectiveness of identifying sums using certain target words; and for each sum that is correctly identified, to report the performance of finding the operands.

6.4.1 Markup Formats

The RDF-based blackboard architecture discussed in Chapter 3 was used by agents to find summation calculations. Similar to the way the agents work in Chapter 5, agents in this chapter share their knowledge on the blackboard by writing RDF statements. When the problem solving process ends, the blackboard contains identified summation calculations in the form of RDF statements. Because RDF statements representing the final solutions are stored as unordered pairs of vertices in a directed graph (see discussion in Section 3.3.2), and comparing undirected graphs is computationally expensive, final solutions are extracted and marked up in XML before evaluation. Appendix G shows the XML schema for the example given below.

```
<rdf:RDF xmlns:testFile="TestData:2002/03/03001/196930.gsml#" >
<topDownSummationCalculatoins>
    <evalResult tableCellUri="testFile:table0cell-7-2"> </evalResult>
    <operand tableCellUri="testFile:table0cel2-1-2"> </operand>
```

```
        <operand tableCellUri="testFile:table0cel2-1-3"> </operand>
        <operand tableCellUri="testFile:table0cel2-1-4"> </operand>
        <operand tableCellUri="testFile:table0cel2-1-5"> </operand>
        <operand tableCellUri="testFile:table0cel2-1-6"> </operand>
    </topDownSummationCalculatoins>
```

At the top level is the tag for the document, which includes zero or more calculations. In this marked up example, the ⟨topDownSummationCalculations⟩ tag marks the calculation; the ⟨evalResult⟩ tag marks the location of the sum; and the operands that make up of the sum are listed inside a pair of ⟨operand⟩ tags.

6.4.2 Test Data

In order to evaluate the table interpretation performance independently (that is, without being affected by the errors that are carried forward from the table structure recognition steps), the input data for the experiments covered in this chapter are tables whose structures are known. The test data are the expected answers for the third set of sampled data (sample 3 in Figure 4.2) discussed in Section 4.5. The test data contain 250 summation calculations.

6.4.3 Performance Analysis

Four experiments were conducted to test the effectiveness of the information that leads to the discovery of sums in tables. The first experiment uses the word 'total' in table headers as the hint to find potential sums. Using this information alone, the recall and precision are 46% and 100% respectively. For the correctly identified sums, 368 out of 470 operands are identified correctly. The second experiment uses both the words 'total' and 'net' to identify sums. The additional word of 'net' brings an additional 10% accuracy to the performance. The third experiment adds the plus symbol (+) to the words which are used in the second experiment, and about 70% of the sums are identified. Finally, the geometric information of finding sums in the last rows of tables, is added to the search process. The geometric information brings no additional gain to the performance. The performance details are given in Tables 6.1 and 6.2.

Target words for finding sums	Num of sums in expected answers	Num of sums that are correctly identified	Num of sums that are missed	Num of sums that are incorrectly introduced
'total'	250	115	135	0
'total' or 'net'	250	139	111	1
'total', 'net' or 'plus'	250	177	73	2

Table 6.1: Experimental results of identifying sums in tables using target words

Target words for finding sums	Num of sums that are correctly identified	Num of operands in the sums that are correctly identified	Num of operands that are correctly identified	Num of operands that is missed	Num of operands that are incorrectly introduced
'total'	115	236	135	101	0
'total' or 'net'	139	603	457	146	0
'total', 'net' or 'plus'	177	715	553	162	0

Table 6.2: Experimental results of identifying operands in summation calculations

As Table 6.1 shows, the precision rates for identifying sums are relatively high compared to the recall rates. The reason for this is that after a sum hypothesis is found, the agents look for the operands. If operands are not found for a particular sum, then the sum is discarded. This ensures that there are no calculations in which the totals are not equal to the sum of two or more operands in the experimental results.

The two summation calculations that are identified in the experimental results but not in the expected answers are due to the following reasons. In one of the summation calculations, the sum differs from the sum of operands by 1, and the program mistakenly thinks that it is a summation calculation with a rounding error. The other summation calculation is actually correctly identified by the program, but purposely left out in the expected answers by the annotator due to errors in the input document. Figure 6.4 displays the section of the input document containing the errors.

In this table, the 'NET ASSETS (row 4.33)' is equal to 'TOTAL ASSETS (row 4.19)' minus 'TOTAL LIABILITIES (row 4.32)'. This relationship is correctly reflected in the last two columns, as $45366 - 44101 = 1265$, and $35,168 - 34,343 = 825$. However, in the middle row, because of the usage of brackets, which denote the number as a negative number, the relationship is reflected differently: 28,878 *plus* (27,789) is equal to 1,089. When marking up the expected answer, the annotator suggests that the inconsistency should be fixed by removing the brackets in rows 4.25 and 4.32, and the summation calculation created by the inconsistency is excluded in the expected answers.

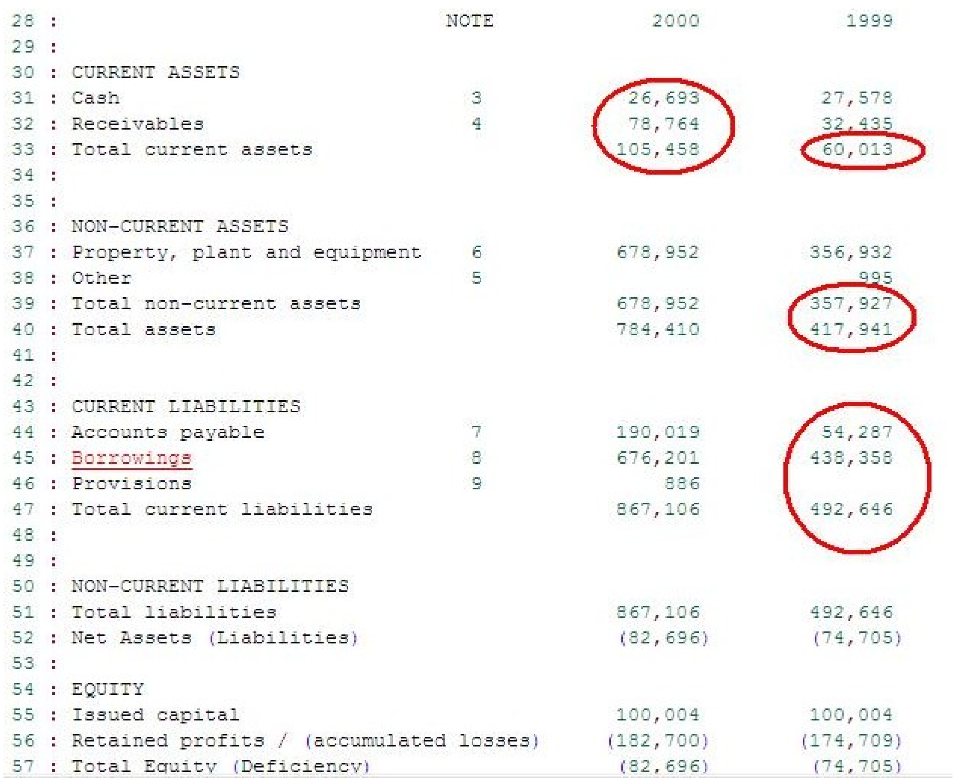
There are 73 summation calculations that could not be identified by the programs. Nine (9) of these calculations are due to the blank headings in the sum rows. That is, there is no information in the row headers indicating that the rows are the sums; eight (8) of them are due to headings which do not contain the words used in finding sum hypothesis. Examples of such headers are 'Cash at end of period (see Reconciliation of cash)' and 'Profit (loss) from ordinary activities after tax attributable to members'. The rest (56)calculations are missed even though their corresponding headers contain the keywords('total', 'net', or the '+' symbol) used in finding the sum hypothesis, and the main reason is due to the '-' or '0' used in the calculation. When looking for summation calculations, the program disallows 0's to be operands, and the program assumes that there are at least two operands to add up to a sum. However, there are many one-term calculations in the input files. In the section of an input document listed above, for example, the program knows that 'total liabilities (row 4.32)' should be equal to 'total current liabilities (row 4.25)' plus 'total non-current liabilities (row 4.31)'. However, because row

4.19	TOTAL ASSETS	28,878	45,366	35,168
	CURRENT LIABILITIES			
4.20	Payables	4,601	17,522	5,975
4.21	Interest bearing liabilities	23,188	26,579	28,368
4.22	Tax liabilities	-	-	-
4.23	Provisions exc tax liabilities	-	-	-
4.24	Other (provide details if material)	-	-	-
4.25	Total current liabilities	(27,789)	44,101	34,343
	NON-CURRENT LIABILITIES			
4.26	Payables	-	-	-
4.27	Interest bearing liabilities	-	-	-
4.28	Tax liabilities	-	-	-
4.29	Provisions exc tax liabilities	-	-	-
4.30	Other (provide details if material)	-	-	-
4.31	Total non-current liabilities	-	-	-
4.32	TOTAL LIABILITIES	(27,789)	44,101	34,343
4.33	NET ASSETS	1,089	1,265	825

Figure 6.4: Calculation errors in input documents. The brackets around '27,789' should have been removed.

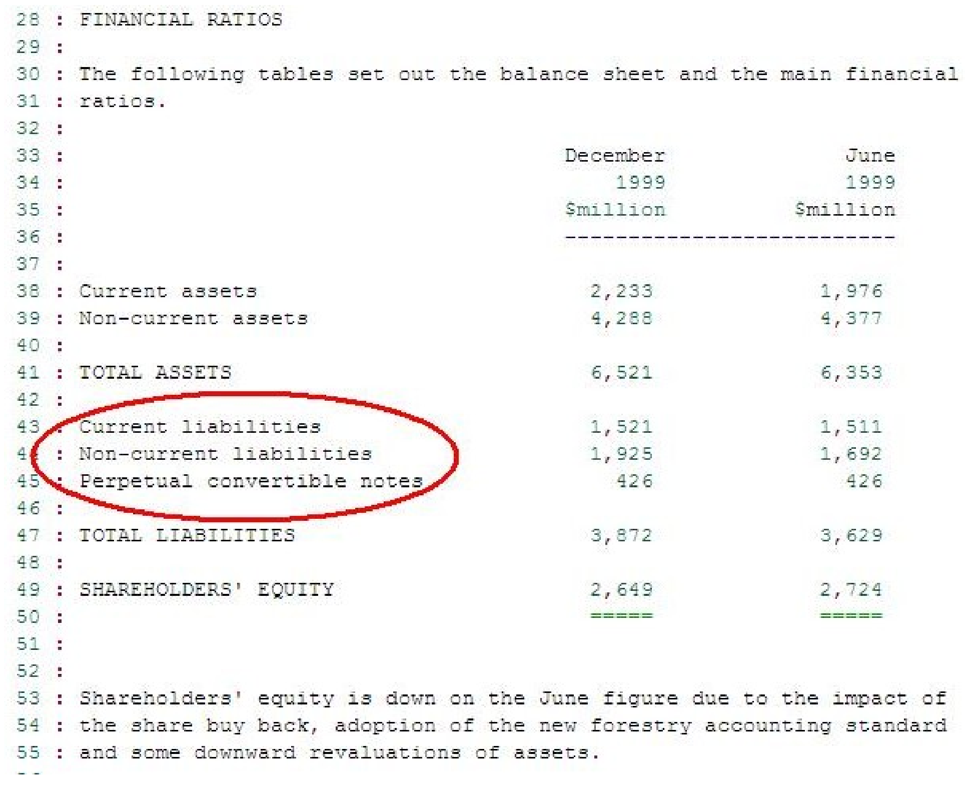
4.31 is '-', the program could not find two or more operands which add up to the sum hypothesis. The same explanation accounts for the majority of the missing operands in the identified summation calculation.

As a side-effect, the agents point out a number of rounding errors and inconsistent use of terminology in the input documents. Figure 6.5 shows three calculation errors in the input document, and Figure 6.6 shows an inconsistency in using terminology: the 'Perpetual convertible notes' (in line 45) should not be listed in parallel to the 'Current liabilities' (in line 43) and 'Non-current liabilities' (in line 44) categories.



```
                                   NOTE           2000          1999

CURRENT ASSETS
Cash                                 3          26,693        27,578
Receivables                          4          78,764        32,435
Total current assets                           105,458        60,013


NON-CURRENT ASSETS
Property, plant and equipment        6         678,952       356,932
Other                                5                           995
Total non-current assets                       678,952       357,927
Total assets                                   784,410       417,941


CURRENT LIABILITIES
Accounts payable                     7         190,019        54,287
Borrowings                           8         676,201       438,358
Provisions                           9             886
Total current liabilities                      867,106       492,646


NON-CURRENT LIABILITIES
Total liabilities                              867,106       492,646
Net Assets (Liabilities)                       (82,696)      (74,705)

EQUITY
Issued capital                                 100,004       100,004
Retained profits / (accumulated losses)       (182,700)     (174,709)
Total Equity (Deficiency)                      (82,696)      (74,705)
```

Figure 6.5: An example of calculation errors in input documents. The errors were found by the program described in this chapter.



```
28 : FINANCIAL RATIOS
29 :
30 : The following tables set out the balance sheet and the main financial
31 : ratios.
32 :
33 :                                                December              June
34 :                                                    1999              1999
35 :                                                $million          $million
36 :                                                --------------------------
37 :
38 : Current assets                                    2,233             1,976
39 : Non-current assets                                4,288             4,377
40 :
41 : TOTAL ASSETS                                      6,521             6,353
42 :
43 : Current liabilities                               1,521             1,511
44 : Non-current liabilities                           1,925             1,692
45 : Perpetual convertible notes                         426               426
46 :
47 : TOTAL LIABILITIES                                 3,872             3,629
48 :
49 : SHAREHOLDERS' EQUITY                              2,649             2,724
50 :                                                   =====             =====
51 :
52 :
53 : Shareholders' equity is down on the June figure due to the impact of
54 : the share buy back, adoption of the new forestry accounting standard
55 : and some downward revaluations of assets.
```

Figure 6.6: An example of inconsistent use of terminology in input documents. The inconsistentnecy (lines 43 to 45) was found by the program described in this chapter.

6.5 Summary

This chapter illustrates the ability of carrying out table interpretation activities using the RDF-based blackboard framework discussed in Chapter 3. Since table interpretation is a complex task, a divide-and-conquer problem-solving strategy is often needed. Agents might employ a different problem-solving strategy to tackle specific sub-problems. For example, the agents in the experiments in this chapter use multiple kinds of algorithms, linguistic information, domain knowledge and logic inferences to achieve their goals. The experiments show that the RDF-based blackboard framework is able to support a wide range of table interpretation tasks, and it is able to combine partial knowledge together when producing an overall solution.

7 Conclusions and Future Work

7.1 Summary

Tables are frequently used in printed documents. Due to their heavy usage and the important nature of data contained therein, tables have been a research subject since the invention of computers. Traditionally, a table analysis problem is solved by a monolithic program. The table processing literature shows that these programs deliver satisfactory results only if they are applied to relatively small domains. This research project set out to improve methods for extracting and interpreting tabular data embedded in printed documents. The result is an RDF-based blackboard architecture that supports both table layout analysis and table interpretation analysis. Experiments in this dissertation cover both layout analysis and interpretation analysis. The experiments confirm not only the feasibility, but also the advantages, of the blackboard architecture.

The advantages of the blackboard architecture are due to three factors: a table analysis problem can be divided into solvable sub-problems; each sub-problem can be solved by diverse knowledge sources, especially knowledge sources representing partial solutions; and the running order of the knowledge sources for each sub-problem is determined by the agents representing the knowledge sources (that is, the running order of the agents do not have to be explicitly determined at design time). Not having to schedule the running order of the agents is a unique characteristic of the blackboard architecture, and this characteristic makes it easier to add or remove agents in a blackboard environment than in a non-blackboard environment. This, together with the multi-level evaluation measures, reports the effectiveness of certain agents and their collaboration. The information disclose the strengths and weaknesses of a system and is useful to the improvement of the algorithms brought by the agents. For example, Appendix K contains results from experiments using various combinations of agents. The data show that the accuracies for recognising cells and rows are relatively low compared to the one for recognising tables in all experiments. This can be explained by the focuses of the agents - they are designed to detect table boundaries rather than recognising the boundaries of table cells and rows. Nevertheless, the multi-level evaluation measures disclose the strengths and weaknesses of the agents adequately.

The blackboard architecture facilitates the co-operation and communication among multiple knowledge sources, so that their combined efforts can lead to the discovery of the overall solutions to a problem they try to solve. In order to achieve this, a number of problems, such as how to coordinate the actions of agents, how to facilitate communications among them, and how to resolve potential conflicts, must be solved. This dissertation provides the solutions to each of these problems. The contributions of this dissertation include finding a suitable knowledge representation for both table structural analysis and table interpretation analysis; deciding on a table annotation scheme that can unambiguously refer to a table, its cells, columns, and rows in an input document; proposing evaluation measures that provide more feedback information than the traditional measures could; and providing details of an RDF-based blackboard framework that can incorporate a wide range of knowledge sources (such as layout and language information) into a table analysis task.

7.2 Future Directions

A number of areas that require future work have been identified in this dissertation. These directions can be divided into five categories, which are discussed in the following sub-sections.

7.2.1 Combining Multiple Sources of Evidence

Throughout this dissertation, the need for combing multiple sources of evidence when making a binary decision has occurred frequently. Combing multiple sources of evidence takes place at the conflict resolving step (see Sections 3.5 and 5.5) as well as at the final solution assembly step (see Section 5.4). In the former case, conflicting statements are produced using different sources of evidence, and a decision must be made to decide which statement can be trusted. The conflict resolving strategies used in this dissertation are based on the assumption that the reliability of each source of evidence is unknown. If the reliability of each source of evidence is available, then more sophisticated conflict resolution strategies can be experimented. In the latter case, various attributes of a table hypothesis are presented, and a decision must be made to decide whether to accept the hypothesis or not. This dissertation makes no assumption about the connection between each attribute and the decision. Rather, it uses an algorithm that learns the connection information from some training data, and makes the decision based on the learnt information. Since assembling the final solutions is an important step, it is worthwhile to explore other strategies and to compare their effectiveness.

7.2.2 Table Analysis on Document Images

Compared to processing plain text documents and markup documents, processing document images has additional processing steps: recognising basic elements, such as letters, digits, spaces, symbols and lines, from some arrangement of pixels, recognising higher levels of elements such as words, numbers, phrases, lines of texts, and ultimately, tables from the basic elements. Given the variability (such as skewness of document images) and the noise (such as dots and creases introduced by scanning processes) in document images, the uncertainty associated with the recognition process increases substantially. Whether the table identification and interpretation problem on document images can still be solved effectively by the agent-based approach, and whether the problem solving process can also be improved from the environment jointly created by the blackboard framework, the markup strategy, and the table structural evaluation method discussed in this dissertation are still open questions.

7.2.3 The Need for a Public Corpus

The lack of a public corpus containing markup data is an obstacle in comparing and improving table analysis systems. As a number of researchers have noted, there has not been much work done to systematically evaluate the performance of table detection algorithms (Y. Wang et al., 2001; H.-H. Chen et al., 2000; Hurst, 1999a). This was partly due to the lack of a large amount of publicly available and accurate table ground truth data to train and test the algorithms. If a markup corpus was publicly made available, then systems could be compared and advanced, perhaps through table analysis competitions, using the same test data. Constructing such a corpus could be resolved by the common efforts of the researchers in the table processing community.

7.2.4 Nested Tables

Although infrequently appearing in plain text documents, nested tables have not been fully explored in this research. Nested tables could potentially complicate the evaluation process discussed in Section 4.4, as there are four possible outcomes that need to be considered: both the surrounding table and the embedded table are identified correctly; only the surrounding table is identified correctly (the embedded table is incorrectly identified); only the embedded table is identified correctly (the surrounding table is incorrectly identified); and neither the surrounding table nor the embedded table is identified correctly.

7.2.5 Recurring Structures as Tables

The literature shows that table identification work to date has focused on finding structures that are physically arranged as columns and rows. By doing this, tables are assumed to have the column-row structures. This assumption rigidly ties the logical characteristics of a table to the physical characteristics of a table. For example, Figure 1.11 contains data that is not physically arranged as columns and rows. However, the data can be rewritten in a table, as shown in Figure 7.1. One possible solution is to expand the definition of tables by including recurring data structures.

Wilkinsons Real Estate Agencies	Flag no.1	241 Windsor St Richmond NSW 2753	ph: (02) 4578 1233		Map
William Inglis & Son Ltd	Flag no.2	42 Argyle St Camden NSW 2570	ph: (02) 4655 3322	advertisement	Map
Williams Allan H	Flag no.3	48 George St Parramatta NSW 2150	ph: (02) 9635 8733		Map
Williams & Hennessy Estate Sales Pty Ltd	Flag no.4	76 Argyle St Camden NSW 2570	ph: (02) 4655 3702		Map
Williams Estate experts	Flag no.5	55 Sydney Rd Manly NSW 2095	ph: (02) 9977 3055		Map
Williamson Real Estate	Flag no.6	Shop 1a 8-10 Somerset Ave Narellan NSW 2567	ph: (02) 4647 1044		Map

Figure 7.1: Tabular format for the data in Figure 1.11

Although this type of recurring structure is often spotted in HTML documents, it could potentially appear in plain text documents and document images as well.

For HTML documents, detecting the recurring structure is more challenging than detecting tables created by ⟨TABLE⟩ tags, because not only do the recurring structures not have fixed length, but also the number of tags and the type of tags used for marking up the data is unknown in advance. These recurring structures have not been given sufficient attention in the table processing literature. However, being able to detect recurring structures is useful, as algorithms that can detect recurring structures can also be used to detect traditional tables that are marked by ⟨TABLE⟩ tags. That is, tables created by ⟨TABLE⟩ tags are one type of recurring structures, with each row being marked by the ⟨TR⟩ tags.

Assuming that each element in recurring structures is marked up by a pair of matched markup tags (as opposed to the element being included as a property within a markup tag), there are three ways in which the element can be marked up. The first is that each element is marked up by a pair of matched tags, and there is a separate markup tag marking the beginning and ending of the element. For example, each repeating element in Figure 1.11 could be embedded in a pair of tags such as ⟨BusinessDetail⟩. In this case, one way to detect repeating patterns is to parse the documents, and for each node in the parsed trees (such as the parent node for the sub-trees marked by the ⟨BusinessDetail⟩ tags), look for immediate isomorphic sub-trees. There are published algorithms for finding isomorphic trees (see, for example, Crepeau (2004)). Given the erroneous nature of documents that exist in WWW, it is worth noting that the isomorphic algorithms are unlikely to be applied without modification. For example, elements with repeating patterns often contain missing fields or empty fields that are excluded from markups, or some unrelated data (such as advertisements) could be inserted between some repeating elements. The working solution will have to apply soft computation to accommodate small errors in input documents, and the results are likely to be probability-based.

In the second case, each element is marked up by a unique tag, and there is no markup tag to indicate the boundaries of repeating elements. If documents are parsed, there will be a flat structure for the multiple repeating elements that share a common parent. Depending on the uniqueness of the markup tags used, the problem could be reduced to the problem of finding the (longest) repeating sub-strings in a string. If the fields in a repeating element are marked up by different tags, the reduction is completed by mapping each type of markup tag to a unique letter. For example, for the HTML document displayed in Figure 1.11, the markup tags for identifying business names can be mapped to the letter ‘a’, and the markup tags for identifying business addresses can be mapped to the letter ‘b’, and so on. Each repeating substring represents a repeating element. Again, a working algorithm will have to provide flexibility to deal with possible errors in input documents.

The worst case occurs when each field in a repeating element is marked up by the same tag. In this case, detecting repeating structures becomes much harder. Because all fields are marked up the same markup tag, the markup tag conveys no information about the data type. Using the markup tag alone is not sufficient for detecting repeating patterns, and other sources of information have to be used. This includes deriving data types from the marked up data.

7.3 Conclusions

Until now, the results produced by various table processing systems have been isolated and restricted to small areas of application. Experiments show that with the contributions made in this dissertation, it is possible to combine the results of different table processing systems together to form a more robust system.

Although the blackboard architecture provides the flexibility of incorporating various knowledge sources in table processing tasks, the decision to use the blackboard architecture alone does not guarantee high levels of accuracy. There are many other factors, such as the way a problem is decomposed and the effectiveness and robustness of the knowledge sources, that determine the accuracy of a table processing system. For example, the experiments covered in Chapter 5 confirm that the strength of the Final Solution Assembler (agent # 14 in Figure 5.2) has a direct influence on the overall performance. Since whether a problem can be solved effectively depends ultimately on the capability of individual agents, finding the right techniques and improving effectiveness of knowledge sources will remain the focus of future research directions in this area.

The Blackboard architecture presented in this dissertation makes it possible to integrate table layout analysis and table interpreation knowledge sources together for improving table analysis tasks. Given the range of problems that are still to be solved, table processing is set to remain a vital and vibrant area of enquiry.

A

A Context-free Grammar Describing Text Documents in Terms of Words

```
Document()  ->  (WordSequence())+ (<EOF>)?

WordSequence()  ->
        POSITIVE_DECIMAL_NUMBER_Dollar()
        | NEGATIVE_DECIMAL_NUMBER_Dollar()
        | POSITIVE_INTEGER_NoThousandSeparator_Dollar()
        | NEGATIVE_INTEGER_NoThousandSeparator_Dollar()
        | POSITIVE_INTEGER_WithThousandSeparator_Dollar()
        | NEGATIVE_INTEGER_WithThousandSeparator_Dollar()
        | POSITIVE_DECIMAL_NUMBER_Percentage()
        | NEGATIVE_DECIMAL_NUMBER_Percentage()
        | POSITIVE_INTEGER_NoThousandSeparator_Percentage()
        | NEGATIVE_INTEGER_NoThousandSeparator_Percentage()
        | POSITIVE_INTEGER_WithThousandSeparator_Percentage()
        | NEGATIVE_INTEGER_WithThousandSeparator_Percentage()
        | POSITIVE_DECIMAL_NUMBER()
        | NEGATIVE_DECIMAL_NUMBER()
        | POSITIVE_INTEGER_NoThousandSeparator()
        | NEGATIVE_INTEGER_NoThousandSeparator()
        | POSITIVE_INTEGER_WithThousandSeparator()
        | NEGATIVE_INTEGER_WithThousandSeparator()
        | POSITIVE_HEX_INTEGER()
        | NEGATIVE_HEX_INTEGER()
        | POSITIVE_OCTAL_INTEGER()
        | NEGATIVE_OCTAL_INTEGER()
        | POSITIVE_SCIENTIFIC_NUMBER()
        | NEGATIVE_SCIENTIFIC_NUMBER()
        | Word_SequenceOfNonBoundaryChar()
        | TwoOrMoreWhiteSpace()
        | OneSingleWhiteSpace()
        | SequenceOfWordSeparatorPunctuationChar()
        | SequenceOfNonPrintableChar()

POSITIVE_DECIMAL_NUMBER_Dollar()    ->
    ("+")? "$" (<DIGIT>)* "." (<DIGIT>)+
    | ("+")? (<DIGIT>)* "." (<DIGIT>)+ "$"

NEGATIVE_DECIMAL_NUMBER_Dollar() ->
    "-" "$" (<DIGIT>)* "." (<DIGIT>)+
    | "-" (" ")? (<DIGIT>)* "." (<DIGIT>)+ "$"

POSITIVE_INTEGER_NoThousandSeparator_Dollar()    ->
    ("+")? "$" ["1"-"9"](<DIGIT>)*
    | ("+")? ["1"-"9"](<DIGIT>)* "$"

NEGATIVE_INTEGER_NoThousandSeparator_Dollar()    ->
    "-" "$" ["1"-"9"] (<DIGIT>)*
    | "-" (" ")? ["1"-"9"] (<DIGIT>)* "$"

POSITIVE_INTEGER_WithThousandSeparator_Dollar()    ->
    ("+")? "$" ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>)
    | ("+")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>) "$"

NEGATIVE_INTEGER_WithThousandSeparator_Dollar()    ->
    "-" "$" ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>)
    | "-" (" ")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>) "$"

POSITIVE_DECIMAL_NUMBER_Percentage()    ->
    ("+")? (<DIGIT>)* "." (<DIGIT>)+ (" ")? "%"

NEGATIVE_DECIMAL_NUMBER_Percentage()    ->
    "-" (" ")? (<DIGIT>)* "." (<DIGIT>)+ (" ")? "%"
```

```
POSITIVE_INTEGER_NoThousandSeparator_Percentage()    ->
    ("+")? ["1"-"9"](<DIGIT>)* (" ")? "%"

NEGATIVE_INTEGER_NoThousandSeparator_Percentage()    ->
    "-" (" ")? ["1"-"9"] (<DIGIT>)* (" ")? "%"

POSITIVE_INTEGER_WithThousandSeparator_Percentage()    ->
    ("+")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)(<DIGIT>)
    (<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>) (" ")? "%"

NEGATIVE_INTEGER_WithThousandSeparator_Percentage()    ->
    "-" (" ")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>) (" ")? "%" >

POSITIVE_DECIMAL_NUMBER()    ->
    ("+")? (<DIGIT>)* "." (<DIGIT>)+

NEGATIVE_DECIMAL_NUMBER()    ->
    "-" (" ")? (<DIGIT>)* "." (<DIGIT>)+

POSITIVE_INTEGER_NoThousandSeparator()    ->
    ("+")? ["1"-"9"](<DIGIT>)*

NEGATIVE_INTEGER_NoThousandSeparator()    ->
    "-" (" ")? ["1"-"9"] (<DIGIT>)*

POSITIVE_INTEGER_WithThousandSeparator()    ->
    ("+")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)(<DIGIT>)
    (<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>)

NEGATIVE_INTEGER_WithThousandSeparator()    ->
    "-" (" ")? ["1"-"9"](<DIGIT>)?(<DIGIT>)? "," ((<DIGIT>)
    (<DIGIT>)(<DIGIT>)",")* (<DIGIT>)(<DIGIT>)(<DIGIT>)

POSITIVE_HEX_INTEGER()    ->
    ("+")? "0" ["x","X"] (["1"-"9","a"-"f","A"-"F"])
    (["0"-"9","a"-"f","A"-"F"])*

NEGATIVE_HEX_INTEGER()    ->
    "-" (" ")? "0" ["x","X"] (["1"-"9","a"-"f","A"-"F"])
    (["0"-"9","a"-"f","A"-"F"])*
```

```
POSITIVE_OCTAL_INTEGER()    ->
    ("+")? "0" (["1"-"7"]) (["0"-"7"])*

NEGATIVE_OCTAL_INTEGER()    ->
    "-" (" ")? "0" (["1"-"7"]) (["0"-"7"])*

POSITIVE_SCIENTIFIC_NUMBER()    ->
    ("+")? (<DIGIT>)+ "." (<DIGIT>)* <EXPONENT> (["f","F","d","D"])?
    | ("+")? "." (<DIGIT>)+ <EXPONENT> (["f","F","d","D"])?
    | ("+")? (["1"-"9"]) (<DIGIT>)* <EXPONENT> (["f","F","d","D"])?

NEGATIVE_SCIENTIFIC_NUMBER() ->
    "-" (" ")? (<DIGIT>)+ "." (<DIGIT>)* <EXPONENT> (["f","F","d","D"])?
    | "-" (" ")? "." (<DIGIT>)+ <EXPONENT> (["f","F","d","D"])?
    | "-" (" ")? (["1"-"9"]) (<DIGIT>)* <EXPONENT> (["f","F","d","D"])?

Word_SequenceOfNonBoundaryChar() -> (~["\u0000"-"\u0008", "\n", "\r",
    "\f", "\u000B", "\u000E"-"\u001F", "\u007F", " ", "\t", "!", "#",
    "&", "’", "*", ":", ";", "=", "?", "@", "^", "‘", "|", "~" ])+

TwoOrMoreWhiteSpace() -> "\t" | (("\t"|" ")("\t"|" ")+)

OneSingleWhiteSpace() -> " "

SequenceOfWordSeparatorPunctuationChar() ->
(PUNCTUATION_CHAR_WordBoundary()) (PUNCTUATION_CHAR_WordBoundary())*

PUNCTUATION_CHAR_WordBoundary() -> "!" | "#" | "&" | "’" | "*"
    | ":" | ";" | "=" | "?" | "@" | "^" | "‘" | "|" | "~"

SequenceOfNonPrintableChar() > NON_PRINTABLE_CHAR()
    (NON_PRINTABLE_CHAR())*

NON_PRINTABLE_CHAR() > ["\u0000"-"\u0008", "\n", "\r", "\f",
    "\u000B", "\u000E"-"\u001F", "\u007F"]

WORD -> (~["\n","\r"," ","\t","|"])+

RULING_LINE1 -> ("--")("-")+ ("\n"|"\r"|"\r\n")
```

```
END_OF_LINE ->
    (" "|"\t")* ("\n"|"\r"|"\r\n")

PHRASE  -> <WORD> (" "<WORD>)*

COLUMN_DELIMITER -> (" ")?  ("  " | "\t" | "|")
    ((" ")?("  " | "\t" | "|"))*  (" ")?
```

B

A Context-free Grammar Describing Plain Text Documents in Terms of Blank Lines, Ruling Lines and Text Lines

```
Document()  ->  (<BLANK_LINE> | <RULING_LINE> | <TEXT_LINE>)+

BLANK_LINE  ->  ((" "|"\t")* ("\n"|"\r"|"\r\n"))
    |  ((" "|"\t")+ ("\n"|"\r"|"\r\n")?)

RULING_LINE2 ->((" "|"\t")* <RULING_LINE_CHAR> <RULING_LINE_CHAR>
    (<RULING_LINE_CHAR>)+ )+  (" "|"\t")*  ("\n"|"\r"|"\r\n")?

RULING_LINE_CHAR ->  "#" | "*" | "_" | "-" | "+" |"=" | ":" |"."

TEXT_LINE   ->  (~["\n","\r"])+ ("\n"|"\r"|"\r\n")?
```

C

A Context-free Grammar Describing Plain Text Documents in Terms of Table and Non-Table Regions

```
Document ->
    (LOOKAHEAD(TableBlock())TableBlock()|NonTableBlock())*

TableBlock  -> LOOKAHEAD(TableBlockWithVerticalSpannedCell())
    TableBlockWithVerticalSpannedCell() | TableBlockNoVerticalSpannedCell()

TableBlockWithVerticalSpannedCell   ->
    (LOOKAHEAD(T2())T2()|(T1()(LOOKAHEAD(B2())B2()|B1()|A1())
    (LOOKAHEAD(T2())T2()|T1()))) (LOOKAHEAD(TailBlock())TailBlock())+

TableBlockNoVerticalSpannedCell()   ->
    (LOOKAHEAD(T2())T2()|(T1()(LOOKAHEAD(B2())B2()|B1()|A1())
    (LOOKAHEAD(T2())T2()|T1())))

TailBlock() ->
    (LOOKAHEAD(B2())B2()|B1()|A1()) (LOOKAHEAD(T2())T2()|T1())

NonTableBlock() ->
    LOOKAHEAD(B3())B3() | LOOKAHEAD(B2())B2() | B1()
    | LOOKAHEAD(A1())A1() | Cell_LINE_IN_NonTableBlock()

B3  ->
    END_OF_LINE() END_OF_LINE() (LOOKAHEAD(2)END_OF_LINE())+
B2  ->
    END_OF_LINE() END_OF_LINE()

B1  ->
    <END_OF_LINE>

T2  ->
    TableRowLine() (LOOKAHEAD(TableRowLine())TableRowLine())+

T1  ->
    TableRowLine()

A1  ->
    <RULING_LINE2>

Cell_LINE_IN_NonTableBlock  ->
     (" ")? (COLUMN_DELIMITER())?
     (PHRASE() (COLUMN_DELIMITER())?)+  END_OF_LINE()

TableRowLine    ->
     (" ")? ( COLUMN_DELIMITER() | Table_CellLine() )+
     TableRowLine_CarriageReturn()

Table_CellLine  ->  PHRASE()

TableRowLine_CarriageReturn ->  END_OF_LINE()

PHRASE -> WORD() (" " WORD())*

WORD -> (~["\n","\r"," ","\t","|"])+

RULING_LINE3 -> ("--")("-")+ ("\n"|"\r"|"\r\n")

END_OF_LINE -> (" "|"\t")* ("\n"|"\r"|"\r\n")

COLUMN_DELIMITER -> (" ")?  ("  " | "\t" | "|")
    ((" ")?("  " | "\t" | "|"))*  (" ")?
```

D
A Context-free Grammar Describing Markup Documents

The following context-free grammar, together with rules that check for matched opening and closing tags, is used to validate whether an XML file meets the annotation specification discussed in section 4.4.3. If an XML file meets the annotation specification, the parsed tree that is generated by this context-free grammar is also used to derive the bounding boxes for evaluation, which is discussed in section 4.4.2.

```
Document -> (metaTaggedItem | xmlTaggedItem)*

metaTaggedItem ->
    MetaTagStart IDENTIFIER (metaTagAssignment)* EndOfTag

xmlTaggedItem ->
    OpenTagStart OpenTagName (PropertyValueAssignment)* EndOfTag
    (LOOKAHEAD(xmlTaggedItem) (xmlTaggedItem | nonTagElement))*
    CloseTagStart CloseTagName EndOfTag

MetaTagStart -> "<?"

IDENTIFIER -> ALPHA (ALPHANUM)*

metaTagAssignment -> IDENTIFIER EqualSign CDATA

PropertyValueAssignment -> IDENTIFIER EqualSign CDATA

OpenTagStart -> "<"
```

```
CloseTagStart -> "</"

nonTagElement-> CDATA

EndOfTag -> ">" | "?>"

EqualSign -> "="

CDATA -> "'" ( ~["'"] )* "'" | "\"" ( ~["\""] )* "\""

OpenTagName -> IDENTIFIER

CloseTagName -> IDENTIFIER

ALPHA -> ["a"-"z","A"-"Z","_","-","."]

ALPHANUM -> ALPHA | NUM

NUM -> ["0"-"9"]
```

E

A Document from the ASX Corpus

The following document shows an ASX document containing long tables. Apart from the inserted line numbers, the document was included without any other editing work.

```
CONSOLIDATED PROFIT AND LOSS ACCOUNT

(Equity Accounted)

                                                CURRENT     PREVIOUS

                                                PERIOD   CORRESPONDING

                                                             PERIOD

                                                AUD000       AUD000




1.1  Sales (or equivalent operating) revenue     4,139        8,027




1.2  Share of associates "net profit(loss)

     attributable to members"

     (equal to item 16.7)                         (370)        (408)


```

```

1.3  Other revenue                                   496        1,649



1.4  Operating  profit (loss) before

     abnormal items and tax                      (1,412)        (772)



1.5  Abnormal items before tax

     (detail in item 2.4)                              -            -



1.6  Operating profit (loss) before tax

     (items 1.4 + 1.5)                           (1,412)        (772)



1.7  Less tax                                          -            -



1.8  Operating profit (loss) after tax

     but before outside equity interests         (1,412)        (772)



1.9  Less outside equity interests                     -            -



1.10 Operating profit (loss) after tax

     attributable to members                     (1,412)        (772)

```

1.11 Extraordinary items after tax (detail in item 2.6)	-	-
1.12 Less outside equity interests	-	-
1.13 Extraordinary items after tax attributable to members	-	-
1.14 Total operating profit (loss) and extraordinary items after tax (items 1.8 + 1.11)	(1,412)	(772)
1.15 Operating profit (loss) and extraordinary items after tax attributable to outside equity interests (items 1.9 + 1.12)	-	-
1.16 Operating profit (loss) and extraordinary items after tax attributable to members		

```
     (items 1.10 + 1.13)                      (1,412)        (772)



1.17 Retained profits (accumulated losses)

     at beginning of financial period        (27,615)     (24,422)



1.18 If change in accounting policy as set

     out in clause 11 of AASB 1018 Profit

     and Loss Accounts, adjustments as

     required by that clause (include brief

     description)                                   -            -



1.19 Aggregate of amounts transferred

     from reserves                                  -            -



1.20 Total available for appropriation       (29,027)     (25,194)



1.21 Dividends provided for or paid                 -            -



1.22 Aggregate of amounts transferred

     to reserves                                    -            -
```

1.23 Retained profits (accumulated losses) at end of financial period	(29,027)	(25,194)

PROFIT RESTATED TO EXCLUDE AMORTISATION OF GOODWILL	Current Period	Previous Corresponding Period
	AUD000	AUD000
1.24 Operating profit(loss) after tax before outside equity interests (items 1.8) and amortisation of goodwill	(1,412)	(772)
1.25 Less (plus) outside equity interests	-	-
1.26 Operating profit(loss) after tax (before amortisation of goodwill) attributable to members	(1,412)	(772)

```





INTANGIBLE, ABNORMAL AND EXTRAORDINARY ITEMS



                                   Consolidated  -  current period



                             Before   Related   Related     Amount

                               tax      tax     outside     (after

                                                 equity       tax)

                                                interests  attributable

                                                             to members



                             AUD000    AUD000    AUD000      AUD000



2.1 Amortisation of

    goodwill                     -         -         -           -



2.2 Amortisation of

    other intangibles            -         -         -           -



```

```
2.3 Total amortisation

    of intangibles              -         -         -            -



2.4 Abnormal items              -         -         -            -



2.5 Total abnormal items        -         -         -            -



2.6 Extraordinary items         -         -         -            -



2.7 Total extraordinary

    items                       0         -         -            -





COMPARISON OF HALF YEAR PROFITS                Current     Previous

(Preliminary final statement only)               year         year

                                                AUD000       AUD000

3.1  Consolidated operating profit

     (loss) after tax attributable

     to members reported for the 1st

     half year (item 1.10 in the

     half yearly report)                           N/A          N/A
```

```
3.2  Consolidated operating profit

     (loss) after tax attributable

     to members for the 2nd half year              N/A           N/A




CONSOLIDATED BALANCE SHEET

(See note 5)

                                   At end of  As in last    As in last

                                    current     annual     half yearly

                                    period      report      report

                                    AUD000       AUD000       AUD000

      CURRENT ASSETS

4.1   Cash                              16            1           80

4.2   Receivables                    4,981        5,420        6,346

4.3   Investments                        -            -            -

4.4   Inventories                    3,732        4,380        4,585

4.5   Other (provide details

      if material)                       -            -            -
```

4.6	Total current assets	8,729	9,801	11,011
	NON-CURRENT ASSETS			
4.7	Receivables	124	123	279
4.8	Investments in associates	852	887	860
4.9	Other investments	6	6	26
4.10	Inventories	-	-	-
4.11	Exploration and evaluation expenditure capitalised	-	-	-
4.12	Development properties (mining entities)	-	-	-
4.13	Other property, plant and equipment (net)	11,034	11,056	10,876
4.14	Intangibles (net)	-	-	-
4.15	Other (provide details if material)	-	-	-
4.16	Total non-current assets	12,016	12,072	12,041
4.17	Total assets	20,745	21,873	23,052

	CURRENT LIABILITIES			
4.18	Accounts payable	269	789	652
4.19	Borrowings	5,156	4,280	3,180
4.20	Provisions	79	111	151
4.21	Other (provide details if material)	-	-	-
4.22	Total current liabilities	5,504	5,180	3,983
	NON-CURRENT LIABILITIES			
4.23	Accounts payable	-	-	-
4.24	Borrowings	63	71	53
4.25	Provisions	104	136	109
4.26	Other (provide details if material)	-	-	-
4.27	Total non-current liabilities	167	207	162

4.28	Total liabilities	5,671	5,387	4,145
4.29	Net assets	15,074	16,486	18,907
	EQUITY			
4.30	Capital - Share Capital and Share Premium	32,879	32,879	32,879
4.31	Reserves	11,222	11,222	11,222
4.32	Retained profits (accumulated losses)	(29,027)	(27,615)	(25,194)
4.33	Equity attributable to members of the parent entity	15,074	16,486	18,907
4.34	Outside equity interests in controlled entities	-	-	-
4.35	Total equity	15,074	16,486	18,907
4.36	Preference capital included as part of 4.33	-	-	-

EXPLORATION AND EVALUATION EXPENDITURE CAPITALISED

To be completed only by entities with mining interests if amounts are material. Include all expenditure incurred regardless of whether written off directly against profit.

		Current period	Previous corresponding period
		AUD000	AUD000
5.1	Opening balance	N/A	N/A
5.2	Expenditure incurred during current period	N/A	N/A
5.3	Expenditure written off during current period	N/A	N/A
5.4	Acquisitions, disposals,		

revaluation increments, etc.	N/A	N/A
5.5 Expenditure transferred to Development Properties	N/A	N/A
5.6 Closing balance as shown in the consolidated balance sheet (item 4.11)	N/A	N/A

DEVELOPMENT PROPERTIES

(To be completed only by entities with mining interests if amounts are material)

	Current period	Previous corresponding period
	AUD000	AUD000
6.1 Opening balance	N/A	N/A
6.2 Expenditure incurred		

```

     during current period                          N/A         N/A



6.3  Expenditure transferred from

     exploration and evaluation                     N/A         N/A



6.4  Expenditure written off

     during current period                          N/A         N/A



6.5  Acquisitions, disposals,

     revaluation increments, etc.                   N/A         N/A



6.6  Expenditure transferred to

     mine properties                                N/A         N/A



6.7  Closing balance as shown in

     the consolidated balance sheet

     (item 4.12)                                    N/A         N/A





CONSOLIDATED STATEMENT OF CASH FLOWS

```

(See note 6)

		Current period AUD000	Previous corresponding period AUD000
CASH FLOWS RELATED TO OPERATING ACTIVITIES			
7.1	Receipts from customers	5,293	9,803
7.2	Payments to suppliers and employees	(5,279)	(11,143)
7.3	Dividends received from associates	-	-
7.4	Other dividends received	-	-
7.5	Interest and other items of similar nature received	52	79
7.6	Interest and other costs of finance paid	(96)	(108)

```
7.7   Income taxes paid                               -          -



7.8   Other (provide details if

      material)                                       -          -



7.9   Net operating cash flows                      (30)    (1,369)



CASH FLOWS RELATED TO INVESTING ACTIVITIES

7.10  Payment for purchases of

      property, plant and equipment                (191)      (511)



7.11  Proceeds from sale of

      property, plant and equipment                   4      1,226



7.12  Payment for purchases of

      equity investments                              -      (252)



7.13  Proceeds from sale of

      equity investments                              -          -
```

7.14	Loans to other entities	(212)	-
7.15	Loans repaid by other entities	-	1,131
7.16	Other (provide details if material)	-	-
7.17	Net investing cash flows	(399)	1,594
	CASH FLOWS RELATED TO FINANCING ACTIVITIES		
7.18	Proceeds from issues of securities (shares, options, etc.)	-	-
7.19	Proceeds from borrowings	444	-
7.20	Repayment of borrowings	-	(422)
7.21	Dividends paid	-	-
7.22	Other (provide details if		

```
      material)                               -         -

7.23  Net financing cash flows              444     (422)

7.24  NET INCREASE (DECREASE) IN CASH HELD   15     (197)

7.25  Cash at beginning of period

      (see Reconciliation of cash)            1       277

7.26  Exchange rate adjustments

      to item 7.25.                           -         -

7.27  Cash at end of period

     (see Reconciliation of cash)            16        80


NON-CASH FINANCING AND INVESTING ACTIVITIES

Details of financing and investing transactions which have had a

material effect on consolidated assets and liabilities but did not
```

involve cash flows are as follows. If an amount is quantified, show comparative amount.

N/A

RECONCILIATION OF CASH

Reconciliation of cash at the end of the period (as shown in the consolidated statement of cash flows) to the related items in the accounts is as follows.	Current period AUD000	Previous corresponding period AUD000
8.1 Cash on hand and at bank	16	80
8.2 Deposits at call	-	-
8.3 Bank overdraft	-	-
8.4 Other - Deposit	-	-

```

8.5  Total cash at end of

     period (item 7.26)                              16           80





RATIOS                                           Current     Previous

                                                  period   corresponding

                                                                period

     PROFIT BEFORE ABNORMALS AND TAX/SALES

9.1  Consolidated operating profit (loss)

     before abnormal items and tax (item

     1.4) as a percentage of sales revenue

     (item 1.1)                                   (34.1) %     (9.62) %



     PROFIT AFTER TAX / EQUITY INTERESTS

9.2  Consolidated operating profit (loss)

     after tax attributable to members

     (item 1.10) as a percentage of equity

     (similarly attributable) at the end of

     the period (item 4.33)                       (9.37) %      (4.08) %



```

EARNINGS PER SECURITY (EPS)

10.1 Calculation of basic, the following

in accordance with

Earnings per Share"

(a)	Basic EPS	(2.4) c	(1.3) c
(b)	Diluted EPS (if materially different from (a))	N/A c	N/A c
(c)	Weighted average number of ordinary shares outstanding during the period used in the calculation of the Basic EPS	59,151,465	59,151,465

NTA BACKING	Current	Previous

```

 (see note 7)                                       period   corresponding

                                                                period

11.1 Net tangible asset backing

     per ordinary security                              25 c         32 c


```

F
XML Schema for Table Structure Annotation

The following XML schema specifies the annotation specification, which is discussed in section 4.4.3.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="Doc">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="TABLE"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="TABLE">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Row"/>
      </xs:sequence>
      <xs:attribute name="tableID" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Row">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="Cell"/>
      </xs:sequence>
      <xs:attribute name="rowID" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Cell">
    <xs:complexType>
      <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" ref="Line"/>
      </xs:sequence>
      <xs:attribute name="cellID" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="Line">
    <xs:complexType>
      <xs:attribute name="beginColumn" use="required" type="xs:integer"/>
      <xs:attribute name="beginLine" use="required" type="xs:integer"/>
      <xs:attribute name="endColumn" use="required" type="xs:integer"/>
      <xs:attribute name="endLine" use="required" type="xs:integer"/>
      <xs:attribute name="lineID" use="required" type="xs:NCName"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

G XML Schema for Annotating Summation Calculations in Tables

The following XML schema specifies the annotation rules for marking up summation calculations in chapter 6.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="topDownSummationCalculatoins">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="evalResult"/>
        <xs:element maxOccurs="unbounded" ref="operand"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="evalResult">
    <xs:complexType>
      <xs:attribute name="tableCellUri" use="required"
  type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="operand">
    <xs:complexType>
      <xs:attribute name="tableCellUri" use="required"
  type="xs:NMTOKEN"/>
    </xs:complexType>
  </xs:element>
```

```
</xs:schema>
```

H

Table Annotation Example

The following file and its corresponding markup code illustrate how expected answers are markup in this dissertation. The markup code use the stand-off annotation methods discussed in section 3.3.3 and the annotation specification discussed in section 4.4.3.

```
 1  outlook for the commercial construction and development market over
 2  the next few years.
 3
 4
 5  CURRENT           PREVIOUS
 6  YEAR              YEAR
 7  $’000             $’000
 8
 9  Total Revenue 178,583           203,496
10  Operating Profit Before
11  Abnormals and Tax 4,326             7,557
12  Less Tax 1,706 2,722
13  Operating Profit After Tax           2,620 4,835
14  Operating Profit After Tax
15  and Extraordinary Items              2,620            4,835
16  Earnings per Share                   3.79c            7.23c
17  Dividends per Share                   3.0c            4.25c
18  Return on Shareholders Funds        13.64%           26.86%
19  NTA per Share                       27.49c 26.45c
20
21  The order book stands at $81.11 million compared to the previous
22  corresponding period of $127 million.
23
```

```
 <?xml version="1.0" encoding="ISO8859-1"
?> <?xml-stylesheet type="text/xsl"
href="C:\\PhD\\MarkupTool\\TableMarkupTool\\styleSheet.xsl"?> <!--
line numbers and column numbers are 0-indexed in this document -->
<Doc>
  <TABLE tableID="table0">
    <Row rowID="table0row0">
        <Cell cellID="table0cell-0-0">
        </Cell>
        <Cell cellID="table0cell-1-0">
            <Line lineID="table0cell-1-0line0"
            beginLine="55"  beginColumn="37"  endLine="55"  endColumn="43"> </Line>
            <Line lineID="table0cell-1-0line1"
            beginLine="56"  beginColumn="37"  endLine="56"  endColumn="40"> </Line>
            <Line lineID="table0cell-1-0line2"
            beginLine="57"  beginColumn="37"  endLine="57"  endColumn="41"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-0">
            <Line lineID="table0cell-2-0line0"
            beginLine="55"  beginColumn="55"  endLine="55"  endColumn="62"> </Line>
            <Line lineID="table0cell-2-0line1"
            beginLine="56"  beginColumn="55"  endLine="56"  endColumn="58"> </Line>
            <Line lineID="table0cell-2-0line2"
            beginLine="57"  beginColumn="55"  endLine="57"  endColumn="59"> </Line>
        </Cell>
    </Row>
    <Row rowID="table0row1">
        <Cell cellID="table0cell-0-1">
            <Line lineID="table0cell-0-1line0"
            beginLine="59"  beginColumn="2"  endLine="59"  endColumn="14"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-1">
            <Line lineID="table0cell-1-1line0"
            beginLine="59"  beginColumn="37"  endLine="59"  endColumn="43"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-1">
            <Line lineID="table0cell-2-1line0"
            beginLine="59"  beginColumn="55"  endLine="59"  endColumn="61"> </Line>
        </Cell>
    </Row>
    <Row rowID="table0row2">
        <Cell cellID="table0cell-0-2">
            <Line lineID="table0cell-0-2line0"
            beginLine="60"  beginColumn="2"  endLine="60"  endColumn="24"> </Line>
            <Line lineID="table0cell-0-2line1"
            beginLine="61"  beginColumn="2"  endLine="61"  endColumn="18"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-2">
            <Line lineID="table0cell-1-2line0"
            beginLine="61"  beginColumn="39"  endLine="61"  endColumn="43"> </Line>
        </Cell>
    <Cell cellID="table0cell-2-2">
        <Line lineID="table0cell-2-2line0"
        beginLine="61"  beginColumn="57"  endLine="61"  endColumn="61"> </Line>
    </Cell>
</Row>
<Row rowID="table0row3">
    <Cell cellID="table0cell-0-3">
        <Line lineID="table0cell-0-3line0"
        beginLine="62"  beginColumn="2"  endLine="62"  endColumn="9"> </Line>
    </Cell>
    <Cell cellID="table0cell-1-3">
        <Line lineID="table0cell-1-3line0"
        beginLine="62"  beginColumn="39"  endLine="62"  endColumn="43"> </Line>
    </Cell>
    <Cell cellID="table0cell-2-3">
        <Line lineID="table0cell-2-3line0"
        beginLine="62"  beginColumn="57"  endLine="62"  endColumn="61"> </Line>
    </Cell>
</Row>
<Row rowID="table0row4">
    <Cell cellID="table0cell-0-4">
        <Line lineID="table0cell-0-4line0"
        beginLine="63"  beginColumn="2"  endLine="63"  endColumn="27"> </Line>
    </Cell>
    <Cell cellID="table0cell-1-4">
        <Line lineID="table0cell-1-4line0"
        beginLine="63"  beginColumn="39"  endLine="63"  endColumn="43"> </Line>
    </Cell>
    <Cell cellID="table0cell-2-4">
        <Line lineID="table0cell-2-4line0"
        beginLine="63"  beginColumn="57"  endLine="63"  endColumn="61"> </Line>
    </Cell>
</Row>
<Row rowID="table0row5">
    <Cell cellID="table0cell-0-5">
        <Line lineID="table0cell-0-5line0"
        beginLine="64"  beginColumn="2"  endLine="64"  endColumn="27"> </Line>
        <Line lineID="table0cell-0-5line1"
        beginLine="65"  beginColumn="2"  endLine="65"  endColumn="24"> </Line>
    </Cell>
    <Cell cellID="table0cell-1-5">
        <Line lineID="table0cell-1-5line0"
        beginLine="65"  beginColumn="39"  endLine="65"  endColumn="43"> </Line>
    </Cell>
    <Cell cellID="table0cell-2-5">
        <Line lineID="table0cell-2-5line0"
        beginLine="65"  beginColumn="57"  endLine="65"  endColumn="61"> </Line>
    </Cell>
</Row>
<Row rowID="table0row6">
    <Cell cellID="table0cell-0-6">
```

```
            <Line lineID="table0cell-0-6line0"
            beginLine="66"  beginColumn="2"  endLine="66"  endColumn="19"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-6">
            <Line lineID="table0cell-1-6line0"
            beginLine="66"  beginColumn="39"  endLine="66"  endColumn="43"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-6">
            <Line lineID="table0cell-2-6line0"
            beginLine="66"  beginColumn="57"  endLine="66"  endColumn="61"> </Line>
        </Cell>
    </Row>
    <Row rowID="table0row7">
        <Cell cellID="table0cell-0-7">
            <Line lineID="table0cell-0-7line0"
            beginLine="67"  beginColumn="2"  endLine="67"  endColumn="20"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-7">
            <Line lineID="table0cell-1-7line0"
            beginLine="67"  beginColumn="40"  endLine="67"  endColumn="43"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-7">
            <Line lineID="table0cell-2-7line0"
            beginLine="67"  beginColumn="57"  endLine="67"  endColumn="61"> </Line>
        </Cell>
    </Row>
    <Row rowID="table0row8">
        <Cell cellID="table0cell-0-8">
            <Line lineID="table0cell-0-8line0"
            beginLine="68"  beginColumn="2"  endLine="68"  endColumn="29"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-8">
            <Line lineID="table0cell-1-8line0"
            beginLine="68"  beginColumn="38"  endLine="68"  endColumn="43"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-8">
            <Line lineID="table0cell-2-8line0"
            beginLine="68"  beginColumn="56"  endLine="68"  endColumn="61"> </Line>
        </Cell>
    </Row>
    <Row rowID="table0row9">
        <Cell cellID="table0cell-0-9">
            <Line lineID="table0cell-0-9line0"
            beginLine="69"  beginColumn="2"  endLine="69"  endColumn="14"> </Line>
        </Cell>
        <Cell cellID="table0cell-1-9">
            <Line lineID="table0cell-1-9line0"
            beginLine="69"  beginColumn="38"  endLine="69"  endColumn="43"> </Line>
        </Cell>
        <Cell cellID="table0cell-2-9">
            <Line lineID="table0cell-2-9line0"
            beginLine="69"  beginColumn="56"  endLine="69"  endColumn="61"> </Line>
        </Cell>
    </Row>
</TABLE> </Doc>
```

RDF Vocabulary in Table Analysis

The following vocabulary was used in the table analysis experiments covered in this dissertation.

I.1 Noun Resources Describing Document Content

URI	Definition
`Doc:SequenceOfNon PrintableChar`	Sequence of non-printable characters.
`Doc:MarkupTag`	Markup tags.
`Doc:TwoOrMoreSpaces`	Two or more consecutive spaces.
`Doc: SequenceOfCharacters`	Sequence of characters.
`Doc:LineArtLine`	Ruling lines, which are typically make up of a sequence of punctuation characters.
`Doc:Table`	A column-row structure in a printed document.
`Doc:TableLine`	A table line.
`Doc:ThreeOrMoreSpaces`	Three or more consecutive spaces.
`Doc:SequenceOfNon SpaceCharacters`	Sequence of non-space characters.
`Doc:SequenceOf PuntuationChar`	Sequence of punctuation characters.
`Doc:TextLine`	A content line (neither a blank line nor a ruling line) in an input document.
`Doc:PlainTextDocument`	Documents that are make up of series of ASCII characters.

I.2 Relation Resources Describing Document Content

URI	Definition
`Doc:totalNumberOf SequenceInLine`	The subject of this relation is a text line, and its number of sequences (consecutive of non-space characters) can be found in the object denoted by this relation.
`Doc:isBeginningOf TableRegion`	The subject of this relation denotes the begin-boundary of a table.
`Doc: minNumOfSeqPerLnInDoc`	The subject of this relation is a document, and its minimum number of sequence per line can be found in the object denoted by this relation.
`Doc:lastTextLineIndex`	The subject of this relation is a document, and its last content line index can be found in the object denoted by this relation.
`Doc: firstTextLineIndex`	The subject of this relation is a document, and its first content line index can be found in the object denoted by this relation.
`Doc:hasTrailing WhiteSpaceSequece`	The subject of this relation is a text line, and it has trailing space characters.

`Doc:totalNumberOfNonSpaceSequenceInLine`	The subject of this relation is a text line, and its number of non-space sequence can be found in the object denoted by this relation.
`Doc:distanceToNextTextLine`	The subject of this relation is a text line, and its distance (in terms of number of lines) to the next text line can be found in the object denoted by this relation.
`Doc:totalNumOfSeqInDoc`	The subject of this relation is a document, and its total number of sequences can be found in the object denoted by this relation.
`Doc:maxNumOfSeqPerLnInDoc`	The subject of this relation is a document, and its maximum number of sequence per line can be found in the object denoted by this relation.
`Doc:isNeitherTableLnNorNonTableLn`	The subject of this relation denotes a line that has not yet been classified as a table line or a non-table line.
`Doc:hasContentString`	The subject of this relation is a table cell, and its content string can be found in the object denoted by this relation.
`Doc:hasDocumentUri`	The subject of this relation is a document, and its URI can be found in the object denoted by this relation.
`Doc:numberOfLineInDocument`	The subject of this relation is a document, and the number of lines it contains can be found in the object denoted by this relation.
`Doc:totalNumberOfTwoOrMoreConsecutiveSpaceInLine`	The subject of this relation is a text line, and the total number of two or more consecutive spaces sequence can be found in the object denoted by this relation.
`Doc:totalNumberOfSingleSpaceInLine`	The subject of this relation is a text line, and the total number of space character can be found in the object denoted by this relation.

`Doc: containTableCellLine`	The subject of this relation is a table cell, and it contents can be found in the object denoted by this relation.
`Doc:containSubstring`	The content string of the subject contains a substring that can be found in the object denoted by this relation.
`Doc:containRow`	The subject of this relation is a table, and it contains a row that can be found in the object denoted by this relation.
`Doc:containColumn`	The subject of this relation is a table, and it contains a column that can be found in the object denoted by this relation.
`Doc:hasLineIndex`	The subject of this relation is an input line in a document, and the line's index can be found in the object denoted by this relation.
`Doc:hasDocumentWidth`	The subject of this relation is a document, and its width can be found in the object denoted by this relation.
`Doc:numberOfSequences`	The object of this relation denotes the number of sequences that a line or a document can have.
`Doc:hasDocumentName`	The subject of this relation is a document, and its name can be found in the object denoted by this relation.
`Doc:hasLineLength`	The subject of this relation is a line, and its length can be found in the object denoted by this relation.
`Doc:averageNumOf SeqPerLnInDocument`	The average number of sequence per line in a document.

I.3 Noun Resources Describing Physical Table Structures

URI	Definition
`TableModel: SpannedCell`	A table cell that spans over multiple rows or columns.
`TableModel: EmbeddedTable`	A table that is nested in another table.
`TableModel: IndexRow`	A table row that provides indexing function. Usually, it is the top most column.
`TableModel: IndexColumn`	A table column that provides indexing function. Usually, it is the right most column.
`TableModel: SingleLineCell`	A table cell, which content is displayed in one text line.
`TableModel: MultiLineCell`	A table cell, which content is displayed in two or more text lines.
`TableModel: TableCell`	A table cell.
`TableModel: TableColumn`	A table column
`TableModel: TableRow`	A table row.
`TableModel: EmptyCell`	An empty table cell.

I.4 Relation Resources Describing Physical Table Structures

URI	Definition
`TableModel:numberOfEmptyCellsInRow`	The subject of this relation denotes the number of empty cells in a table row.
`TableModel:tableIndex`	A document can contain more than one table. The subject of this relation denotes the table index in a document.
`TableModel:numberOfEmptyCellsInColumn`	The subject of this relation denotes the number of empty cells in a table column.
`TableModel:containTableCell`	The subject of this relation is a table, and it contains a table cell that is referenced by the object.
`TableModel:tableCellContentStr`	The subject of this relation is a table cell, and its string can be found in the object denoted by this relation.
`TableModel:numberOfCellLinesInCell`	The subject of this relation is a table cell, and the number of lines in the table cell can be found in the object denoted by this relation.
`TableModel:numberOfNonEmptyCellsInColumn`	The subject of this relation denotes the number of non-empty cells in a column.
`TableModel:numberOfNonEmptyCellsInRow`	The subject of this relation denotes the number of non-empty cells in a row.
`TableModel:numberOfTable`	The subject of this relation is a document, and the number of tables it contains can be found in the object denoted by this relation.
`TableModel:columnIndex`	The subject of this relation is a table cell, and its column index can be found in the object denoted by this relation.

`TableModel: numberOfNonEmpty CellsInTable`	The subject of this relation denotes the number of non-empty cells in a table.
`TableModel: numberOfRowsInTable`	The subject of this relation is a table, and the number of rows it has can be found in the object denoted by this relation.
`TableModel: numberOfColumnsInTable`	The subject of this relation is a table, and the number of columns it has can be found in the object denoted by this relation.
`TableModel:rowIndex`	The subject of this relation is a table cell, and its row index can be found in the object denoted by this relation.
`TableModel:numberOfEmpty CellsInTable`	The subject of this relation denotes the number of empty cells in a table.

I.5 Noun Resources Describing Dates

URI	Definition
`GeneralVocabulary: Date`	A data type denoting a data.
`GeneralVocabulary: Date_YearMonthDay`	A data type denoting a date with the format of listing the year first, then the month, then the day. Example of this data type is '2008-01-15'.
`GeneralVocabulary: Date_ MonthDayWithoutYear`	A data type denoting a date with the format of listing the month followed by the day. The year is not listed. Example of this data type is 'July 4'.
`GeneralVocabulary: Date_MonthOnly`	a data type denoting a month. The year is not listed. Example of this data type is 'July'.
`GeneralVocabulary: Date_ YearMonthWithoutDay`	A data type denoting a date with the format of listing the year followed by the month. The day is not listed. Example of this data type is '2008 January'.
`GeneralVocabulary: Date_YearOnly`	A data type denoting a year. '2008'.
`GeneralVocabulary: Conflicts`	Statements that are semantically contradict to each other. Example of these statement are 'line 10 is a table-begin boundary' and 'line 10 is a table-end boundary'.

I.6 Noun Resources Describing Table Lines

In this category, the first entry is the super class of the subsequent entries. An rdf statement of the form (?s, ?p o), where o is one of the subclasses, can be inferred as $(?s,\ ?p\ Doc : TableLine)$ (see discussion in section 3.3.4).

URI	Definition
`Doc:TableLine`	This noun resource denotes a text line that is also a table line.
`Doc:TableLine_ hasTableColumnDelimiter`	This noun resource denotes a text line that is also a table line, because it contains column delimiters.
`Doc:TableLine_ fallWithinTableRegion`	This noun resource denotes a text line that is also a table line, because it falls within a table region that is detected by the context-free grammar described in appendix C.

I.7 Noun Resources Describing Non-Table Lines

In this category, the first entry is the super class of the subsequent entries. An rdf statement of the form $(?s,\ ?p\ o)$, where o is one of the subclasses, can be inferred as $(?s,\ ?p\ Doc : NonTableLine)$ (see discussion in section 3.3.4).

URI	Definition
`Doc:NonTableLine`	This noun resource denotes a non-table line.
`Doc:NonTableLine_ hasForbiddenChar`	This noun resource denotes a non-table line, because it contains characters (such as xml tags) that are unlikely to be in tables.
`Doc:NonTableLine_ singleLongText`	This noun resource denotes a non-table line, because it is a long text line (see discussion of agent #4 in section 5.3).
`Doc:NonTableLine_ betweenLongTextLines`	This noun resource denotes a non-table line, because although the line is not a long text line, its preceding and following lines are long text lines.
`Doc:NonTableLine_ precedingAndFollowing_ LinesAreNonTableLine`	This noun resource denotes a non-table line, because its previous two lines and its following two lines are known as non-table lines.
`Doc:NonTableLine_ noTableColumnDelimiter_ AndBetweenNonTableLines`	This noun resource denotes a non-table line, because it does not contain any column delimiter, and its preceding line and its following line are known as non-table lines.

I.8 Noun Resources Describing Table-Begin Boundaries

In this category, the first entry is the super class of the subsequent entries. An rdf statement of the form $(?s,\ ?p\ o)$, where o is one of the subclasses, can be inferred as $(?s,\ ?p\ Doc : BeginningOfTableRegion)$ (see discussion in section 3.3.4).

URI	Definition
`Doc: BeginningOfTableRegion`	This noun resource denotes a table-begin boundary.
`Doc:BeginningOfTable_ 1stLineInNumericColumn`	This noun resource denotes a table-begin boundary, because it does not contain numeric data, and it is the first table-line immediately followed by one or more numeric columns.
`Doc:BeginningOfTable_ firstTableBoundary`	This noun resource denotes a table-begin boundary, because it is the first table-line in a document.
`Doc:BeginningOfTable_ changeOfLineType`	This noun resource denotes a table-begin boundary, because it is at the point where text line patterns changing from a block of non-table lines to a block of table-lines.
`Doc:BeginningOfTable_ atOrNearTableLnContain TypicalHeaderStr`	This noun resource denotes a table-begin boundary, because it contains keywords that are typically used in tables.
`Doc:BeginningOfTable_ longestTableBlock BetweenTwoTableEnd Regions`	This noun resource denotes a table-begin boundary, because the boundary line balancer (agent #9 in figure 5.2) is able to find a suitable table-begin boundary between two consecutive table-end boundaries (see discussion in section 5.3.3).
`Doc: BeginningOfTableRegion_ final`	This noun resource denotes a table-begin boundary, which is decided by the final solution assembler (agent #14 in figure 5.2).
`Doc:BeginningOfTable_ grammarRules`	This noun resource denotes a table-begin boundary, because it is matched by the context-free grammar described in appendix C.

I.9 Noun Resources Describing Table-End Boundaries

In this category, the first entry is the super class of the subsequent entries. An rdf statement of the form $(?s,\ ?p\ o)$, where o is one of the subclasses, can be inferred as $(?s,\ ?p\ Doc : EndOfTableRegion)$ (see discussion in section 3.3.4).

URI	Definition
`Doc:EndOfTableRegion`	A table-end boundary.
`Doc:EndOfTableRegion_final`	This noun resource denotes a table-end boundary, which is decided by the final solution assembler (agent #14 in figure 5.2).
`Doc:EndOfTable_lastLineInNumericColumn`	This noun resource denotes a table-end boundary, because it is the last table-line in a numeric column.
`Doc:EndOfTable_changeOfLineType`	This noun resource denotes a table-end boundary, because it is at the point where text line patterns changing from a block of table-lines to a block of non-table lines.
`Doc:EndOfTable_lastTableBoundary`	This noun resource denotes a table-end boundary, because it is the last table-line in a document.
`Doc:EndOfTable_longestTableBlockBetweenTwoTableBeginRegions`	This noun resource denotes a table-end boundary, because the boundary line balancer (agent #9 in figure 5.2) is able to find a suitable table-end boundary between two consecutive table-begin boundaries (see discussion in section 5.3.3).
`Doc:EndOfTable_grammarRules`	This noun resource denotes a table-end boundary, because it is matched by the context-free grammar described in appendix C.

I.10 Noun Resources Describing Numeric Data

URI	Definition
`GeneralVocabulary:DataType`	Data type (e.g. string or integer).
`GeneralVocabulary:Number`	A numeric data type.
`GeneralVocabulary:Positive`	A numeric data type that is greater than zero
`GeneralVocabulary:Negative`	A negative number.
`GeneralVocabulary:Integer`	An integer in the base-ten numeral system. Example of this data type is '33'
`GeneralVocabulary:DecimalNumber`	A data type denoting a decimal number in the base-ten numeral system. Example of this data type is '2.3'
`GeneralVocabulary:Percentage`	A data type denoting a number expressed as a percentage. Example of this data type is '33%'
`GeneralVocabulary:CurrencyAmount`	A numeric data denoting the amount of money.
`GeneralVocabulary:CurrencyUnit`	A data type denoting a currency unit. Example of this data type is '$000'
`GeneralVocabulary:NumberUnit_thousands`	A measurement unit that means 'one thousand'.
`GeneralVocabulary:NumberUnit_million`	A measurement unit that means 'one million'.

I.11 Noun Resources Describing Arithmetic Expressions

URI	Definition
`Arithmetic:MathsTerms`	Mathematic glossary.
`Arithmetic: ArithmeticExpression`	Arithmetic expression.
`Arithmetic: ArthmeticOperator_less`	The subtract arithmetic operator.
`Arithmetic: ArthmeticOperator_ multiply`	The multiply arithmetic operator.
`Arithmetic: ArthmeticOperator_minus`	The subtract arithmetic operator.
`Arithmetic: ArthmeticOperator_ dividedBy`	The divide arithmetic operator.
`Arithmetic: ArthmeticOperator_times`	The multiply arithmetic operator.
`Arithmetic: ArithmeticOperator`	Arithmetic operators.
`Arithmetic: ArthmeticOperator_plus`	The add arithmetic operator.

I.12 Relation Resources Describing Arithmetic Expressions

URI	Definition
`Arithmetic:SubTotal`	The subject of this relation denotes the total of part of a series of numbers.
`Arithmetic: hasEvalResultWith RoundingError`	The subject of this relation denotes an arithmetic result, which contains rounding error.
`Arithmetic: hasCalculationLayout`	The subject of this relation denotes a summation layout, which either lists the sum before a series of numbers, or lists the sume after a series of numbers.
`Arithmetic: useArithmeticOperator`	The subject of this relation uses an arithmetic operator stated in the object denoted by this relation.
`Arithmetic:leftOperand`	The subject of this relation has a left operand, which is stated in the object denoted by this relation..
`Arithmetic:Sum`	The ‘sum of’ or ‘total’ relationship.
`Arithmetic:hasEvalResult`	The subject of this relation has an evaluation result stated in the object denoted by this relation.
`Arithmetic:rightOperand`	The subject of this relation denotes The subject of this relation has a right operand, which is stated in the object denoted by this relation.
`Arithmetic:hasError`	The subject of this relation denotes a calculation error.
`Arithmetic:Total`	The subject of this relation is an arithmetic sum of a series of numbers.
`Arithmetic: hasCalculationError`	The subject of this relation denotes an arithmetic calculation result, which contains error(s).
`Arithmetic:Operand`	The subject of this relation has an operand, which is either a left operand or a right operand.

I.13 Noun Resources Describing Accounting Concepts

The following accounting terms are extracted from `http://www.accountz.com/glossary.html`. They are used for the table interpretation experiments that are covered in this dissertation.

URI	Definition
`Accounting:asset`	Assets represent what a business owns or is due.
`Accounting:Total_Assets`	The sum of current assets and non-current assets.
`Accounting:Net_Assets`	The difference between the business's assets and liabilities. It is also referred as 'net worth' or 'equity'.
`Accounting: Total_Liability`	The sum of current liabilities and non-current liabilities.
`Accounting:total`	The sum of.
`Accounting:Liability`	Money owed to external parties.
`Accounting:Balance_sheet`	A summary of all the accounts of a business.

J Inference Rules in Table Analysis Experiments

The following inference rules were used in the table analysis experiments covered in this dissertation.

J.1 Inference Rules for Detecting Conflicting Statements

```
 [conflict_tableLineStatus_1:
  (?s a Doc:NonTableLine),
  (?s a Doc:TableLine)
  ->
  (?s GeneralVocabulary:contain GeneralVocabulary:ConflictedStatements)
]

[conflict_tableLineStatus_2:
  (?s a Doc:BeginningOfTableRegion),
  (?s a Doc:EndOfTableRegion)
  ->
  (?s GeneralVocabulary:contain GeneralVocabulary:ConflictedStatements)
]
```

J.2 Inference Rules for Deriving Table Boundaries

```
/* A table line marks the beginning of a table boundary
```

```
    if it is the first line in document.
*/
[isBeginningOfTable_1:
  (?tableLineUri a Doc:TableLine)
  (?fileUri Doc:firstTextLineIndex ?firstTextLineIndex),
  (?tableLineUri Doc:hasLineIndex ?firstTextLineIndex),
  ->
  (?tableLineUri a Doc:BeginningOfTable_firstTextLineInDoc)
]

/* A table line marks the end of a table boundary if it is the
last line in document.
*/
[isEndOfTableRegion_1:
  (?tableLineUri a Doc:TableLine)
  (?fileUri Doc:lastTextLineIndex ?lastTextLineIndex),
  (?tableLineUri Doc:hasLineIndex ?lastTextLineIndex),
  ->
  (?tableLineUri a Doc:EndOfTable_lastTextLineInDoc)
]

/* A table line marks the beginning of a table
(isBeginningOfTable) if its preceding line is a non-table line and
its following line has not yet be classified as a non-table line.
*/
[isBeginningOfTable_2:
  (?tableLineUri a Doc:TableLine),
  (?precedingLineUri Doc:nextTextLineUri ?tableLineUri),
  (?tableLineUri Doc:nextTextLineUri ?followingLineUri),
  (?precedingLineUri a Doc:NonTableLine),
  (?followingLineUri a Doc:TableLine)
  ->
  (?tableLineUri a Doc:BeginningOfTable_changeOfLineType)
]

[isBeginningOfTable_3:
  (?tableLineUri a Doc:TableLine),
  (?precedingLineUri Doc:nextTextLineUri ?tableLineUri),
  (?tableLineUri Doc:nextTextLineUri ?followingLineUri),
  (?precedingLineUri a Doc:NonTableLine),
  (?followingLineUri a Doc:NeitherTableLnNorNonTableLn)
  ->
  (?tableLineUri a Doc:BeginningOfTable_changeOfLineType)
]

/* A table line marks the end of a table region
(isEndOfTableRegion) if its preceding line
has not yet been classified as a non-table line, and its following
line is a non-table line.
*/
[isEndOfTableRegion_2:
  (?tableLineUri a Doc:TableLine),
  (?precedingLineUri Doc:nextTextLineUri ?tableLineUri),
  (?tableLineUri Doc:nextTextLineUri ?followingLineUri),
  (?precedingLineUri a Doc:TableLine),
  (?followingLineUri a Doc:NonTableLine)
  ->
  (?tableLineUri a Doc:EndOfTable_changeOfLineType)
]

[isEndOfTableRegion_3:
  (?tableLineUri a Doc:TableLine),
  (?precedingLineUri Doc:nextTextLineUri ?tableLineUri),
  (?tableLineUri Doc:nextTextLineUri ?followingLineUri),
  (?precedingLineUri a Doc:NeitherTableLnNorNonTableLn),
  (?followingLineUri a Doc:NonTableLine)
  ->
  (?tableLineUri a Doc:EndOfTable_changeOfLineType)
]
```

J.3 Inference Rules for Deriving Statements Through Sub-classes or Sub-properties

```
 [isBeginningOfTableRegion_1:
  (?s a Doc:BeginningOfTable_atOrNearTableLnContaingTypicalHeaderStr)
  ->
  (?s a Doc:BeginningOfTableRegion)
]

[isBeginningOfTableRegion_2:
  (?s a Doc:BeginningOfTable_changeOfLineType)
  ->
  (?s a Doc:BeginningOfTableRegion)
]

[isBeginningOfTableRegion_3:
  (?s a Doc:BeginningOfTable_firstTextLineInDoc)
  ->
  (?s a Doc:BeginningOfTableRegion)
]

[isEndOfTableRegion_1:
  (?s a Doc:EndOfTable_changeOfLineType)
  ->
  (?s a Doc:EndOfTableRegion)
]

[isEndOfTableRegion_2:
  (?s a Doc:EndOfTable_lastTextLineInDoc)
  ->
  (?s a Doc:EndOfTableRegion)
]

[isNonTableLine_1:
  (?s a Doc:NonTableLine_betweenLongTextLines)
  ->
  (?s a Doc:NonTableLine)
]

[isNonTableLine_2:
  (?s a Doc:NonTableLine_singleLongText)
  ->
  (?s a Doc:NonTableLine)
]

[isNonTableLine_3:
  (?s a Doc:NonTableLine_consecutiveLongText)
  ->
  (?s a Doc:NonTableLine)
]
```

```
[isNonTableLine_4:
  (?s a Doc:NonTableLine_noTableColumnDelimiterAndBetweenNonTableLines)
  ->
  (?s a Doc:NonTableLine)
]

[isNonTableLine_5:
  (?s a Doc:NonTableLine_hasForbiddenChar)
  ->
  (?s a Doc:NonTableLine)
]

[isNonTableLine_6:
  (?s a Doc:NonTableLine_precedingAndFollowingLinesAreNonTableLine)
  ->
  (?s a Doc:NonTableLine)
]

[isTableLine_1:
  (?s a Doc:TableLine_hasTableColumnDelimiter)
  ->
  (?s a Doc:TableLine)
]

[isTableLine_2:
  (?s a Doc:TableLine_fallWithinTableRegion )
  ->
  (?s a Doc:TableLine)
]

[Number_1:
  (?s ?p GeneralVocabulary:DecimalNumber)
  ->
  (?s ?p GeneralVocabulary:Number)
]

[Number_2:
  (?s ?p GeneralVocabulary:HexDecimalNumber)
  ->
  (?s ?p GeneralVocabulary:Number)
]
```

```
[Number_3:
  (?s ?p GeneralVocabulary:Integer)
  ->
  (?s ?p GeneralVocabulary:Number)
]

[Number_4:
  (?s ?p GeneralVocabulary:OctalNumber)
  ->
  (?s ?p GeneralVocabulary:Number)
]

[Number_5:
  (?s ?p GeneralVocabulary:ScientificNumber)
  ->
  (?s ?p GeneralVocabulary:Number)
]

[CurrencyAmount_1:
  (?s ?p GeneralVocabulary:AusDollarAmount)
  ->
  (?s ?p GeneralVocabulary:CurrencyAmount)
]

[CurrencyAmount_2:
  (?s ?p GeneralVocabulary:USDollarAmount)
  ->
  (?s ?p GeneralVocabulary:CurrencyAmount)
]
```

K
Evaluation Results

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Inser-tion	Dele-tion	Proper subset error	Proper superset error	Interset error	
Table-level	75	170	11	166	40	81	0	38	7.63%
Row-level	149	1988	402	369	1137	155	155	139	18.70%
Cell-level	149	7069	10	191	4374	78	2345	262	7.15%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.1: The benchmark result produced by the deterministic method published by Ng et al. (1999).

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	2	170	36	7	23	12	98	1	64.95%
Row-level	11	1988	809	1050	377	479	175	148	32.94%
Cell-level	11	7069	23	499	1977	199	4546	324	10.67%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.2: The collaborative result produced by all agents, excluding the benchmark agent. The final selection step uses the global optimisation algorithm.

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	1	170	32	14	23	11	97	7	64.38%
Row-level	6	1988	794	1065	448	423	175	148	30.62%
Cell-level	6	7069	16	549	2319	204	4222	308	9.87%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.3: The collaborative result produced by all agents, excluding the benchmark agent. The final selection step uses the local optimisation algorithm.

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	2	170	30	21	13	15	100	12	70.82%
Row-level	13	1988	863	1131	351	451	175	148	33.68%
Cell-level	13	7069	23	603	1977	204	4539	326	10.93%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.4: The collaborative result produced by all agents, including the benchmark agent. The final selection step uses the global optimisation algorithm.

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	1	170	33	15	24	11	94	8	74.18%
Row-level	6	1988	844	1058	382	439	175	148	31.07%
Cell-level	6	7069	22	533	2053	203	4485	306	9.98%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.5: The collaborative result produced by all agents, including the benchmark agent. The final selection step uses the local optimisation algorithm.

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	1	170	32	29	42	24	70	2	65.54%
Row-level	5	1988	669	753	640	356	175	148	27.53%
Cell-level	5	7069	20	329	2841	134	3784	290	9.98%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.6: The collaborative result produced by agents #2 and #14 in Figure 5.2. Agnet #14 uses the global optimisation work mode in this experiment.

	Insertion errors on test cases without tables	Test cases that contain tables							% Area of bounding boxes being detected correctly
		Number of expected answers	Number of correctly detected tables	Non-overlapping error		Overlapping error			
				Insertion	Deletion	Proper subset error	Proper superset error	Interset error	
Table-level	1	170	26	12	40	26	68	10	68.23%
Row-level	6	1988	696	938	552	419	173	148	27.52%
Cell-level	6	7069	24	409	2519	197	4030	299	9.39%

Proper subset error: parts of expected answer are detected without introducing non-table contents
Proper superset error: the whole expected answer tables, together with some non-table contents, are detected
Interset error: parts of expected answer, together with some non-table contents, are detected

Table K.7: The collaborative result produced by agents #3, #4, #6, #7 and #14 in Figure 5.2. Agnet #14 uses the global optimisation work mode in this experiment.

L
Examples of errors made by the benchmark algorithm

As Section 5.3 shows, the benchmark algorithm in this dissertation is a re-implementation (implemented by agent #6) of what was published by Ng et al. (1999). The algorithm was originally used to establish a benchmark performance in identifying table lines from Wall Street Journals, which are plain text documents. The algorithm was able to identify 70% of the lines correctly. However, when this algorithm is applied to the test data in this dissertation, it can identify only about 7% of the text lines. Given that the algorithm was clearly specified in the original publication, the difference in the performance can be caused by the difference in test data, or what considered as tables by the human annotators in both experiments, or the combination of the two factors. Figures L.1, L.2 and L.3 list some errors made by the benchmark algorithm on the test data used in this dissertation. The 'recall err' column shows table lines that are failed to be identified by the algorithm, whereas the 'precision err' column lists non-table lines that are incorrectly identified as table lines by the algorithm. The rules for a text line to be classified as a table line is listed in Section 5.3 when describing agent #6. For clarity reasons, the rules are listed again and are numbered as the follows. When a line is incorrectly classified by the benchmark algorithm on the test data in Figures L.1, L.2 and L.3, the rule number is shown in the appropriate column.

rule 1: a blank line is always classified as a non-table line;

rule 2: a text line is classified as a table line if it is not a blank line, and the ratio of the position of the first non-space character in the line to the length of the line exceeds some pre-determined threshold (0.25);

rule 3: a text line is classified as a table line if it is not a blank line, and the line consists entirely of one special character, where a special character is any character that is neither a space character nor a alphanumeric character;

rule 4: a text line is classified as a table line if it is not a blank line, and the line contains three or more segments, each consisting of two or more contiguous space characters;

rule 5: a text line is classified as a table line if it is not a blank line, and the line contains two or more segments, each consisting of two or more contiguous separator characters, where a separator character is any of the following characters: '.', '*' or '-'.

rule 6: a text line is classified as a non-table line if none of the above conditions is met.

Line	Document	Recall err	Precision err
1	`Open Telecommunications Limited               2004-04-30  ASX-SIGNAL-G`		
2			
3	`HOMEX - Sydney`		
4	`+++++++++++++++++++++++++`		rule 3
5	`Open Telecom. reported positive cash flow of $1,505,000 for the quarter`		
6	`ended 31 March 2004. Operating cash flow for the period was $(289,000).`		
7	`Investing cash flow was $(3,371,000). Financing cash flow was`		
8	`$5,165,000. Cash in hand at the end of the quarter was $3,682,000.`		
9			
10	`Consolidated Statement of Cash Flows for the quarter ended 31 March 2004`		
11			
12	`+----------------------------+--------------------+-----------------+`	rule 6	
13	`|                            |Current Quarter     |Year to Date     |`		
14	`|                            |A$'000 - 31 March   \|$A'000 - 3 months|`		
15	`|                            |2004                |                 |`		
16	`+----------------------------+--------------------+-----------------+`	rule 6	
17	`|Net Operating Cash Flows    |(289)               |(289)            |`		
18	`+----------------------------+--------------------+-----------------+`		
19	`|Net Investing Cash Flows    |(3,371)             |(3,371)          |`		
20	`+----------------------------+--------------------+-----------------+`	rule 6	
21	`|Total Operating and         |(3,660)             |(3,660)          |`		
22	`|Investing Cash Flows        |                    |                 |`		
23	`+----------------------------+--------------------+-----------------+`	rule 6	
24	`|Net Financing Cash Flows    |5,165               |5,165            |`		
25	`+----------------------------+--------------------+-----------------+`	rule 6	
26	`|Net Increase(decrease) in   |1,505               |1,505            |`		
27	`|Cash Held                   |                    |                 |`		
28	`+----------------------------+--------------------+-----------------+`	rule 6	
29	`|Cash at begining of         |2,177               |2,177            |`		
30	`|Quarter/Year to Date        |                    |                 |`		
31	`+----------------------------+--------------------+-----------------+`	rule 6	
32	`|Cash end of Quarter         |3,682               |3,682            |`		
33	`+----------------------------+--------------------+-----------------+`	rule 6	
34			

Figure L.1: First example of errors made by the benchmark algorithm, which performance is listed in Appendix K (Figure K.1)

Line	Document	Recall err	Precision err
1	APPENDIX 3B		rule 2
2	NEW ISSUE ANNOUNCEMENT		rule 2
3			
4	APPLICATION FOR QUOTATION OF ADDITIONAL SECURITIES AND AGREEMENT		
5			
6	Information or documents not available now must be given to ASX as		
7	soon as available. Information and documents given to ASX become		
8	ASX's property and may be made public.		
9			
10	Introduced 1/7/96. Origin Appendix 5. Amended 1/7/98, 1/9/99,		
11	1/7/2000.		
12			
13	Name of Entity		
14	Oroton International Limited		
15			
16	ACN or ARBN		
17	000 038 675		
18			
19	We (the entity) give ASX the following information.		
20			
21			
22	PART 1 - ALL ISSUES		
23	You must complete the relevant sections (attach sheets if		
24	there is not enough space).		
25			
26			
27			
28	1. Class of securities issued Options over ordinary shares	rule 6	
29	or to be issued	rule 6	
30		rule 1	
31	2. Number of securities issued 220,000	rule 6	
32	or to be issued (if known)	rule 6	
33	or maximum number which	rule 6	
34	may be issued	rule 6	
35		rule 1	
36	3. Principal terms of the securities The exercise price is $2.96,	rule 6	
37	(eg, if options, exercise price the options have a term of	rule 6	
38	and expiry date; if partly paid eight years and become	rule 6	
39	securities, the amount exercisable in three equal	rule 6	
40	outstanding and due dates for tranches at two year intervals.	rule 6	
41	payment; if convertible securities,	rule 6	
42	the conversion price and dates	rule 6	
43	for conversion)	rule 6	
44		rule 1	
45	4. Do the securities rank equally No	rule 6	
46	in all respects from the date	rule 6	
47	of allotment with an existing	rule 6	
48	class of quoted securities	rule 6	
49		rule 1	
50	If the additional securities		
51	do not rank equally, please	rule 6	
52	state:		
53	* the date from which they do	rule 6	
54	* the extent to which they	rule 6	
55	participate for the next	rule 6	
56	dividend, (in the case of		
57	a trust, distribution) or	rule 6	
58	interest payment	rule 6	
59	* the extent to which they do		
60	not rank equally, other than	rule 6	
61	in relation to the next	rule 6	
62	dividend, distribution or	rule 6	
63	interest payment		
64		rule 1	
65	5. Issue price or consideration Nil	rule 6	
66		rule 1	
67	6. Purpose of the issue (if An issue to five employees	rule 6	
68	issued as consideration for under the Oroton Executive	rule 6	
69	the acquisition of assets, Share and Option Scheme on the	rule 6	
70	clearly identify those terms contained in that Scheme.		
71	assets)	rule 6	
72		rule 1	
73	7. Dates of entering securities N/A	rule 6	
74	into uncertified holdings	rule 6	
75	or despatch of certificates	rule 6	
76		rule 1	

Figure L.2: Second example of errors made by the benchmark algorithm, which performance is listed in Appendix K (Figure K.1) (to be continued)

Line	Document	Recall err	Precision err
77	NUMBER CLASS		
78	8. Number and class of all 18,228,142 Ordinary	rule 6	
79	securities quoted on	rule 6	
80	ASX (including the	rule 6	
81	securities in clause	rule 6	
82	2 if applicable)	rule 6	
83		rule 1	
84	NUMBER CLASS		
85	9. Number and class of all 1,020,000 Options	rule 6	
86	securities not quoted	rule 6	
87	on ASX (including the	rule 6	
88	securities in clause 2	rule 6	
89	if applicable)	rule 6	
90			
91	10.Dividend policy (in the case N/A	rule 6	
92	of a trust, distribution	rule 6	
93	policy) on the increased	rule 6	
94	capital (interests)	rule 6	
95			
96	PART 2 - BONUS ISSUE OR PRO RATA ISSUE		
97			
98	Items 11 to 33 are Not Applicable		
99			
100	PART 3 - QUOTATION OF SECURITIES		
101	You need only complete this section if you are applying for quotation		
102	of securities		
103			
104	Items 34 to 37 are Not Applicable		
105			
106	Entities that have Ticked Box 34 (b)		
107			
108	Items 38 to 42 are Not Applicable		
109			
110	ALL ENTITIES		
111			
112	Fees		
113			
114	43. Payment method (tick one)		
115			
116	Cheque attached		rule 2
117			
118	Electronic payment made		
119	Note: Payment may be made electronically if Appendix 3B is		
120	given to ASX electronically at the same time.		
121			
122	Periodic payment as agreed with the home branch has been		
123	arranged		rule 2
124	Note: Arrangements can be made for employee incentive		
125	schemes that involve frequent issues of securities.		
126			
127	QUOTATION AGREEMENT		
128			
129	1. Quotation of our additional securities is in ASX's absolute		
130	discretion. ASX may quote the securities on any conditions it		
131	decides.		rule 2
132			
133	2. We warrant to ASX that the issue of the securities to be quoted		
134	complies with the law and is not for an illegal purpose, and that		
135	there is no reason why those securities should not be granted		
136	quotation. We warrant to ASX that an offer of the securities for		
137	sale within 12 months after their issue will not require		
138	disclosure under section 707(3) of the Corporations Law.		
139			
140	3. We will indemnify ASX to the fullest extent permitted by law in		
141	respect of any claim, action or expense arising from or connected		
142	with any breach of the warranties in this agreement.		
143			
144	4. We give ASX the information and documents required by this form.		
145	If any information or document not available now, will give it to		
146	ASX before quotation of the securities begins. We acknowledge that		
147	ASX is relying on the information and documents. We warrant that		
148	they are (will be) true and complete.		
149			
150			
151	P A Stearn		
152	COMPANY SECRETARY		
153	05/07/2001		

Figure L.3: Second example of errors made by the benchmark algorithm, which performance is listed in Appendix K (Figure K.1) (continued)

Glossary

ASX	Australian Stock Excange.
Blank line	A *blank line* is either an empty line or a line that contains only whitespace characters.
Cell line	A *cell line*, marked by one or two column delimiters, is a sequence of text tokens embedded in the same row-line. For example, the first line in Figure 1.6 is a row-line containing two cell lines: ‘Current Period ended 31’ and ‘Previous Period ended 31’.
Column delimiter	A *column delimiter* is a sequence of characters that separate table columns. For example, the ‘|’ characters and their preceding whitespaces in Figure 1.6 are column delimiters.
DAG	Directed Acyclic Graph.
Document length	A *Document length* is the number of lines in a document. It is useful to know the document length as most operations are performed in the iterations within the document length. i++).
Document width	A *document width* is the maximum number of characters in a line, excluding trailing white space. Document width is used to calculate the relative lengths of text lines in a document. over 50% or 90% long. The relative length is one possible measure for judging whether a text line is short or not. and calculate ”longTextLineThreshold”. For example, average + 1.5 standard deviation, half way between min and max that is min + 0.5*(max - min), or set to a specific fixed threshold (for example, 55). Tables usually contain short text lines.
DTD	Document Type Definition.
GUI	Graphical User Interface.
NLP	Natural Language Processing.
OASIS	Organization for the Advancement of Structured Information Standards.
OCR	Optical Character Recognition.
RDF	Resource Description Framework.
Row block	A *row block* is the longest block of adjacent row lines. For example, the first four lines in Figure 1.7 form one row block, because the fifth line is not a row line. Another example: all the lines in Figure 1.8 are in one row block, and the table contains one big row block.

Row delimiter A *row delimiter* is a sequence of lines that separate two rows. In Figure 1.6, each pair of adjacent rows in the table is separated by a ruling line; and in Figure 1.7, all the blank lines are row delimiters.

Row-line A *row-line* is a table-line that contributes to the content of a table. For example, the first line in the table in Figure 1.7 is a row-line. Note that row-lines may also contain punctuation marks that are used to indicate column boundaries. The relationship between a row-line and a table-line is that a row-line is a table-line, but a table-line (such as an empty line or a ruling line in a table) is not necessarily a row-line.

Ruling line A *ruling line* is a line whose purpose is to serve as a row delimiter. Ruling 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 Figure 1.6 is a ruling line.

Table A *table* is an interpretable column-row structure.

Table cell A *table cell* is the most basic, semantically complete unit in a table. A cell sometimes is just a cell line, but for a vertically spanned cell, a cell must contain two or more cell lines from adjacent row-lines. For example, the last line in Figure 1.8 is a row-line containing three cell lines, and each of these cell lines is a cell. However, the first row of the table in Figure 1.6 has two (spanned) cells containing two cell lines each. One of them, with a string value of 'Current Period ended 31 March 2004 (NZ$000)', consists of two cell lines: 'Current Period ended 31' and 'March 2004 (NZ$000)'.

Table row A *table row* is one or more row-lines containing cells that are horizontally aligned. Sometimes a table row can be just a row-line, but very often a table row may consist of multiple row-lines. For example, the last line in Figure 1.8 is a table row, and the first four lines in Figure 1.7 also form one table row.

Wide space sequence A *wide space sequence*, as opposed to single-space sequence, is a sequence of two or more consecutive whitespace characters appearing in the same line. Tables in plain text documents are often created using blank lines and a mixture of whitespace characters and punctuation characters. These sequences of white space characters and punctuation characters are often referred to as row delimiters and column delimiters.

WYSIWYG What You See Is What You Get.

References

Abu-Tarif, A. (1998). *Table processing and understanding.* Unpublished master's thesis, Rensselaer Polytechnic Institute.

Amano, A., & Asada, N. (2002). Complex table form analysis using graph grammar. In *Document Analysis System V: 5th International Workshop, DAS 2002* (p. 283 ff.). Princeton, NJ, USA: Springer.

Amano, A., & Asada, N. (2003, August 3-6). Graph grammar based analysis system of complex table form document. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR)* (p. 916-920). Edinburgh, Scotland: IEEE Computer Society.

Amano, A., Asada, N., & Mukunoki, M. (2004). Modification table form generation system based on the form recognition. In *Proceedings of the 17th International Conference on Pattern Recognition, ICPR 2004* (p. II: 659-662). Cambridge UK: IEEE Computer Society.

Antonacopoulos, A., & Bridson, D. (2007, September). Performance Analysis Framework for Layout Analysis Methods. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR2007)* (p. 1258-1262). Curitiba, Brazil: IEEE.

Antonacopoulos, A., Karatzas, D., & Bridson, D. (2006). Ground Truth for Layout Analysis Performance Evaluation. In H. Bunke & A. S. (Eds.) (Eds.), *Document Analysis Systems VII: 7th International Workshop on Document Analysis Systems (DAS2006)* (p. 302-311). Nelson, New Zealand: Springer Lecture Notes in Computer Science, LNCS 3872.

Baird, H. (1992). *Anatomy of a versatile page reader.*

Beach, R. J. (1985). *Setting tables and illustrations with style.* Unpublished doctoral dissertation, University of Waterloo, Waterloo, Canada.

Belaid, Y., & Belaid, A. (1999). Form analysis by neural classification of cells. In *DAS '98: Selected Papers from the Third IAPR Workshop on Document Analysis Systems* (pp. 58–71). London, UK: Springer-Verlag.

Biggerstaff, T. J., Endres, D. M., & Forman, I. R. (1984). Table: Object oriented editing of complex structures. In *Proceedings of International Conference on Software Engineering.* IEEE Computer Society.

Bordini, R., Dastani, M., Dix, J., & Seghrouchni, A. E. F. (Eds.). (2005). *Multi-agent programming languages, platforms and applications.* New

York, NJ, USA: Springer.

Bradshaw, J. (Ed.). (1997). *Software agents.* AAI Press / The MIT Press.

Brenner, W., Zarnekow, R., & Wittig, H. (1998). *Intelligent software agents, foundations and applications.* New York: Springer.

Bridson, D., & Antonacopoulos, A. (2008, December). A geometric approach for accurate and efficient performance evaluation of layout analysis methods. In *Proceedings of the 19th International Conference on Pattern Recognition (ICPR2008).* Tampa, Florida, USA: Springer Berlin / Heidelberg.

Buteau, B. (1990). A generic framework for distributed, cooperating blackboard systems. In *Proceedings of the 1990 ACM Annual Conference on Cooperation* (p. 358 - 365).

Cameron, J. P. (1989). *A cognitive model for tabular editing.* Computer and Information Science Research Center, Ohio State University, 2036 Neil Avenue Mall, Columbus, Ohio 43210.

Campos, A. M. de, & Macedo, M. M. de. (1992). A blackboard architecture for perception planning in autonomous vehicles. In *Proceedings of the 1992 International Conference on Industrial Electronics, Control, Instrumentation and Automation* (p. 826 - 831). San Diego, California, USA: IEEE.

Carver, N., & Lesser, V. (1992, October). *The evolution of blackboard control architectures* (Tech. Rep. No. UM-CS-1992-071).

Chen, H.-H., Tsai, S.-C., & Tsai, J.-H. (2000, July). Mining tables from large scale html texts. In *Proceedings of the 18th International Conference on Computational Linguistics.* Saarbrucken, Germany: Morgan Kaufmann.

Chen, J. S., & Tseng, D. C. (1996, September). Overlapped-character separation and reconstruction for table-form document. In *ICIP-96* (p. 17A8). Lausanne, Switzerland: IEEE. Signal processing society.

Cherry, L. L., & Lesk, M. E. (January 1979). *Tbl - a program to format tables.*

Clark, N. (1987). *Tables and graphs as a form of exposition.*

Consortium, T. W. W. W. (1999a). *The world wide web consortium (w3c) recommendation 24 december 1999. html 4.01 specification.* Last accessed on 20 September 2009 from http://www.w3.org/TR/html401.

Consortium, T. W. W. W. (1999b). *The world wide web consor-*

tium (w3c) recommendation 24 december 1999. tables. Last accessed on 20 September 2009 from http://www.w3.org/TR/REC-html40/struct/tables.html.

Crepeau, C. (2004). Introduction to computer science. online classroom material hosted by mcgill university.

Crucianu, M., Ayadi, R. E., & Vincent, N. (2001). *On the representation of tables in xml.*

Das, A. K., Saha, S. K., & Chanda, B. (March 2002). *An empirical measure of the performance of a document image segmentation algorithm.* Berlin / Heidelberg: Springer.

Douglas, S., & Hurst, M. (1996). Layout and language: lists and tables in technical documents. In *Proceedings of SIGPARSE Workshop on Punctuation in Computational* (p. 19-24). Santa Cruz, CA, USA: Morgan Kaufmann.

Douglas, S., Hurst, M., & Quinn, D. (1995). Using natural language processing for identifying and interpreting tables in plain text. In *Proceedings of the Fourth Symposium on Document Analysis and Information Retrieval* (p. 535-546). Las Vegas, Nevada. University of Nevada: World Scientific Publishing Co. Pte. Ltd.

Duygulu, P., & Atalay, V. (2000, January). *A hierarchical representation of form documents for identification and retrieval.* San Jose, USA.

Duygulu, P., Atalay, V., & Dincel, E. (1998, August). A heuristic algorithm for hierarchical representation of form documents. In *Proceedings of the 14th International Conference on Pattern Recognition.* Brisbane, Australia: IEEE.

Embley, D., Hurst, M., Lopresti, D., & Nagy, G. (2006, June). Table-processing paradigms: a research survey. In *International Journal on Document Analysis and Recognition* (p. 66-86). Nelson, New Zealand: Springer.

Embley, D., Tao, C., & Liddle, S. (2002, October). Automatically extracting ontologically specified data from html tables with unknown structure. In *Proceedings of the 21st International Conference on Conceptual Modeling (ER 2002)* (p. 322-327). Tampere, Finland: Springer.

Erman, L. D., Hayes-Roth, F., Lesser, V. R., & Reddy, D. R. (1980). The hearsay-ii speech understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, *12*(2), 213-253.

Erman, L. D., & Lesser, V. R. (1990). The hearsay-ii speech understanding system: A tutorial. In A. Waibel & K.-F. Lee (Eds.), *Readings in Speech Recognition* (p. 235-245). Morgan Kaufmann.

Freiburghouse, R. (1974). *Tbl - table building language.* Last accessed on 20 September 2009 from http://www.multicians.org/raf-tbl-definition.html.

Furuta, R. (1986, 14-16 April). An integrated, but not exact-representation. In J. C. van Vliet (Ed.), *Proceedings of the International Conference on Text Processing and Document Manipulation* (p. 246-259). University of Nottingham: Cambridge University Press.

Garris, M., Janet, S., & Klein, W. (1999, January). Federal register document image database. In *Proceedings of Document Recognition and Retrieval VI (IS&T/SPIE Electronic Imaging 1999)* (Vol. 3651, p. 97-108). San Jose, CA.

The gnu troff project. (n.d.). http://groff.ffii.org.

Green, E., & Krishnamoorthy, M. (1995a, August). Model-based analysis of printed tables. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR 1995)* (p. 214-217). Montreal, Canada: IEEE.

Green, E., & Krishnamoorthy, M. (1995b, August). Model-based analysis of printed tables. In *Proceedings of the First International Workshop on Graphics Recognition (GREC 1995)* (p. 234-242). University Park, PA, USA: Springer.

Green, E., & Krishnamoorthy, M. (1995c, April). Recognition of tables using table grammars. In *Proceedings of the 5th Annual Symposium on Document Analysis and Informatin Retrieval (SDAIR 1995)* (p. 261-277). Las Vegas, Nevada: World Scientific Publishing Co. Pte. Ltd.

Green, E. A. (May 1996). *Model-based analysis of printed tables.* Unpublished doctoral dissertation, PhD thesis, Rensselaer Polytechnic Institute, New York.

Guthrie, J. T., Weber, S., & Kimmerly, N. (1993). *Searching documents: Cognitive processes and deficits in understanding graphs, tables and illustrations.*

Ha, J., Haralick, R., & Phillips, I. (1995). Recursive x-y cut using bounding boxes of connected components. In *Proceedings of 3rd Interna-*

tional Conference Document Analysis and Recognition (p. 952-955). Springer.

Haralick, R. M. (1994). Document image understanding: Geometric and logical layout. In *Proceedings of Conference on Computer Vision and Pattern Recognition (CVPR94)* (pp. 385–390). Seattle, WA, USA: IEEE Computer Society.

Hayes-Roth, B. (1984). *Bb1: An architecture for blackboard systems that control, explain, and learn about their own behavior* (Tech. Rep.). Stanford, CA, USA.

Hayes-Roth, B., Pfleger, K., Lalanda, P., Morignot, P., & Balabanovic, M. (1995). A domain-specific software architecture for adaptive intelligent systems. *IEEE Trans. Software Engineering, 21*(4), 288-301.

Hirayama, Y. (1995). A method for table structure analysis using dp matching. In *Proceedings of the Third International Conference Document Analysis and Recognition* (p. 583-586).

Hirschberg, J. (March 2008). Automatic speech recognition: An overview.

Hooker, J. T. (1990). Reading the past. In *British Museum Publications.*

Hori, O., & Doermann, D. S. (1995). Robust table-form structure analysis based on box-driven reasoning. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR)* (p. 218-221). Montreal: IEEE.

Hu, J., Kashi, R., Lopresti, D., Nagy, G., & Wilfong, G. (2001, 10-13 September). Why table ground-truthing is hard. In *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR)* (p. 129-133). Seattle, WA, USA: IEEE.

Hu, J., Kashi, R., Lopresti, D., & Wilfong, G. (2000a). Medium-independent table detection. In *Document Recognition and Retrieval VII* (p. 291-302).

Hu, J., Kashi, R., Lopresti, D., & Wilfong, G. (2000b, December). A system for understanding and reformulating tables. In *Fourth ICPR Workshop on Document Analysis Systems* (p. 361-372). Rio De Janeiro, Brazil.

Hu, J., Kashi, R., Lopresti, D., & Wilfong, G. (2001a). Experiments in table recognition. In *Workshop on Document Layout Interpretation and Applications.* Seattle, Washington.

Hu, J., Kashi, R., Lopresti, D., & Wilfong, G. (2001b). Table structure recognition and Its Evaluation. In *Document Recognition and Retrieval VIII* (p. 44-55).

Hu, J., Kashi, R., Lopresti, D., & Wilfong, G. (2002). Evaluating the performance of table processing algorithms. *International Journal of Document Analysis and Recognition (IJDAR)*, *4*(3), 140-153.

Hurst, M. (1999a). Layout and language: A corpus of documents containing tables. In *AAAI Fall Symposium Using Layout for the Generation, Understanding or retrieval of documents.*

Hurst, M. (1999b). Layout and language: Beyond simple text for information interaction - modelling the table. In *Proceedings of the Second International Conference on Multimodal Interfaces.* Hong Kong.

Hurst, M. (2000). *The interpretation of tables in texts.* Unpublished doctoral dissertation, School of Cognitive Science, University of Edinburgh, Edinburgh, Scotland.

Hurst, M. (2001a). Layout and language: An efficient algorithm for detecting text blocks based on spatial and linguistic evidence. 56-67.

Hurst, M. (2001b, September). Layout and language: Challenges for table understanding on the web. In *Proceedings of the 1st International Workshop on Web Document Analysis* (p. 27-30). Seattle WA, USA.

Hurst, M. (2003, 3-6 August). A constraint-based approach to table structure derivation. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR)* (p. 911-915). Edinburgh, Scotland: IEEE.

Hurst, M., & Douglas, S. (1997a). Layout and language: Preliminary experiments in assigning logical structure to table cell. In *Proceedings of the Fifth Applied Natural Language Processing Conference* (p. 217-220). Washington D.C.

Hurst, M., & Douglas, S. (1997b). Layout and language: Preliminary investigations in recognizing the structure of tables. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* (p. 1043-1047). Ulm, Germany: IEEE.

Hurst, M., & Nasukawa, T. (2000). Layout and language: Integrating spatial and linguistic knowledge for layout understanding tasks. In *Proceedings of the Eighteenth International Conference Computational Linguistics.* Saarbrucken, Germany.

Ide, N., & Romary, L. (2004). International standard for a linguistic annotation framework. *Naturaql Language Engineering*, 10:3-4, 211-225.

Ide, N., & Romary, L. (2006). Representing linguistic corpora and their annotations. In *Proceedings of the Fifth Language Resources and Evaluation Conference (LREC).* Genoa, Italy.

Jena 2 inference support. (n.d.). http://jena.sourceforge.net/inference.

Jennings, N. (2001). *An agent-based approach for building complex software systems.* Communications of the ACM.

Kanai, J., Nartker, T. A., Rice, S. V., & Nagy, G. (1993, October). Performance metrics for document understanding systems. In *IEEE Computer Society Press* (p. 424-427). Tsukuba, Japan.

Kboubi, F., Chabi, A. H., & Ahmed, M. B. (2005, September). Table recognition evaluation and combination method. In *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR)* (p. 1237-1241). Seoul, Korea: IEEE.

Kieninger, T. (1998, January). Table structure recognition based on robust block segmentation. In *Proceedings of IS&T/SPIE's 10th Annual Symposium Electronic* (p. 22-32). San Jose, CA.

Kieninger, T., & Dengel, A. (2005, September). An approach towards benchmarking of table structure recognition results. In *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR)* (p. 1232-1236). Seoul, Korea: IEEE.

Komfeld, W., & Wattecamps, J. (1998). Automatically locating, extracting and analyzing tabular data. In *Proceedings of the Twenty-first International ACM SIGIR Conference Research and Development in Information Retrieval* (p. 347-348). Melbourne, Australia.

Krupl, B., Herzog, M., & Gatterbauer, W. (2005, 10-14 May). Using visual cues for extraction of tabular data from arbitrary html documents. In *Proceedings of the Special Interest Tracks and Posters of the 14th International Conference on WWW* (pp. 1000–1001). ACM Press.

Lamport, L. (1994). *Latex: A document preparation system—user's guide and reference manual* (Second ed.). Addison Wesley.

Laurentini, A., & Viada, P. (1992). Identifying and understanding tabular material in compound document. In *Proceedings of the Eleventh International Conference Pattern Recognition* (p. 405-409). The Hague.

Lin, C., & Hsiao, C. (1998, November). Structural recognition for table-form documents using relaxation techniques. *PRAI*, *12*(7), 985.

Lin, S. H., & Ho, J. M. (2002). Discovering informative content blocks from web documents. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (SIGKDD'02).*

Liu, Y., Bai, K., Mitra, P., & Giles, C. L. (2009, July 26-29 2009). Improving the table boundary detection in pdfs by fixing the sequence error of the sparse lines. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR).* Barcelona, Spain: IEEE.

Long, V. (2009, July 26-29 2009). An rdf-based blackboard architecture for improving table analysis. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR).* Barcelona, Spain: IEEE.

Long, V., Cassidy, S., & Dale, R. (2006, February). A multi-level table evaluation method for plain text documents. In *Extended Abstracts of the 7th International Association for Pattern Recognition Workshop on Document Analysis Systems (DAS 2006)* (p. 21-24). Nelson, New Zealand.

Long, V., Dale, R., & Cassidy, S. (2005, September). A model for detecting and merging vertically spanned table cells in plain text documents. In *Proceedings of the 8th International Conference on Document Analysis and Recognition (ICDAR)* (p. 1242-1246). Seoul, Korea: IEEE.

Lopresti, D., & Nagy, G. (1999a). Automated table processing: An (opinionated) survey. In *The Third International Workshop on Graphics Recognition* (p. 109-134). Jaipur, India.

Lopresti, D., & Nagy, G. (1999b, September 26-27). A tabular survey of automated table processing. In *Graphics Recognition. Recent Advances: Third International Workshop, GREC'99, Jaipur, India.* (p. 93-120). Springer Verlag: Springer Berlin / Heidelberg.

Lopresti, D., & Wilfong, G. (2001). Evaluating document analysis results via graph probing. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition* (p. 116-120). Seattle, Washington, USA: IEEE.

Losee, R. M. (2003, April). Adaptive organization of tabular data for display. In *Journal of Digital Information.*

Manning, C. D., & Schutze, H. (1999). *Foundations of statistical natural* *language processing.* The MIT Press.

Maoa, S., Rosenfelda, A., & Kanungob, T. (2003). Document structure analysis algorithms: A literature survey. In *In Proceedings of SPIE* (pp. 197–207).

Marchionini, G., Hert, C., Liddy, L., & Shneiderman, B. (2000). Extending understanding of federal statistics in tables. In *Proceedings of the ACM Conference on Universal Usability* (p. 132-138).

McClain, J. (2004). *A behavior-based blackboard architecture for multi-robot control.* Unpublished master's thesis, University of Georgia.

McKenzie, G., Preece, A., & Gray, P. (2006a, May). Implementing a semantic web blackboard system using jena. In *Jena User Conference.* Bristol, United Kingdom.

McKenzie, G., Preece, A., & Gray, P. (2006b, December). A semantic web blackboard system. In *Proceedings of the International Conference on Innovative Techniques and Applications of Artificial Intelligence (AI 2006)* (p. 275-288). Cambridge, UK: Springer.

Michalski, R. S., Bratko, I., & Kubat, M. (Eds.). (1998). *Machine learning and data miniing methods and applications.* John Wiley and Sons Ltd.

Mitchell, T. M. (1997). *Machine learning.* WCB McGraw-Hill.

Nagy, G. (2000, January). Twenty years of document image analysis in pami. *PAMI*, *22*(1), 38-62.

Nagy, G., & Seth, S. (1984). Hierarchical representation of optically scanned documents. In *Proceedings of the International Conference on Pattern Recognition (ICPR)* (p. 347-349).

Ng, H. T., Lim, C. Y., & Koo, J. L. T. (1999). Learning to recognize tables in free text. In *Proceedings of the 37th Annual Meeting of the Association for Computational Linguistics* (p. 443-450). College Park, Maryland, USA.

Nielson, H. E., & Barrett, W. A. (2003, 3-6 August). Consensus-based table form recognition. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR)* (p. 906-910). Edinburgh, Scotland: IEEE.

OASIS. (1995). *Organization for the advancement of structured information standards (oasis) (technical research paper 9501:1995). table interoperability: Issues for the cals table model.* Last accessed on 20 September 2009 from http://www.oasis-open.org/specs/a501.htm.

OASIS. (1996). *Organization for the advancement of structured information standards (oasis)(technical resolution no. 9503:1995). exchange table model document type definition.* Last accessed on 20 September 2009 from http://www.oasis-open.org/specs/a503.htm.

OASIS. (1999). *Organization for the advancement of structured information standards (oasis) (technical memorandum tr 9901:1999). xml exchange table model document type definition.* Last accessed on 20 September 2009 from http://www.oasis-open.org/specs/tm9901.htm.

Oro, E., & Ruffolo, M. (2009, July 26-29). Pdf-trex: An approach for recognizing and extracting tables from pdf documents. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR).* Barcelona, Spain: IEEE.

Padgham, L., & Winikoff, M. (2004). *Developing intelligent agent system.* John Wiley & Sons Ltd.

Penn, G., Hu, J., Luo, H., & McDonald, R. (2001, September). Flexible web document analysis for delivery to narrow-bandwidth. In *Proceedings of International Conference Document Analysis and Recognition (ICDAR).* Seattle, WA, USA: IEEE.

Peterman, C., Chang, C. H., & Alam, H. (1997). A system for table understanding. In *Document Image Understanding Technology* (p. 55-62). Annapolis, MD.

Pinto, D., McCallum, A., Lee, X., & Croft, W. B. (2003). Table extraction using conditional random fields. In *Proceedings of the 26th ACM SIGIR* (p. 235-242). Toronto, Canada.

Pyreddy, P., & Croft, W. B. (1997). Tintin: A system for retrieval in text tables. In *Proceedings of the Second International Conference on Digital Libraries* (p. 193-200).

Quinlan, J. R. (1993). *C4.5: Programs for machine learning.* San Mateo, CA: Morgan Kaufmann Publishers.

Quint, V., & Vatton, I. (1986). Grif:an interactive system for structured document manipulation. In *Proceedings of the International Conference on Text Processing and Document Manipulation* (p. 200-213). Cambridge University Press.

Quint, V., & Vatton, I. (1996). The languages of thot. In *Technical report, INRIA.*

Rahgozar, M., & Cooperman, R. (1996). A graph-based table recognition system. In *SPIE Proc. 2660* (p. 192-203).

Ramel, J. Y., Crucianu, M., Vincent, N., & Faure, C. (2003). Detection, extraction and representation of tables. In *Proceedings of the 7th International Conference on Document Analysis and Recognition* (p. 374-378). Edinburgh, Scotland: IEEE.

Reid, B. K. (1980). *A document specification language and its compiler.* Unpublished doctoral dissertation, Carnegie-Mellon University, Pittsburgh, PA.

Reklaitis, G., Sunol, A., Rippin, D., & Hortacsu, O. (Eds.). (1996). *Batch processing system engineering fundamentals and applications for chemical engineering.* Springer.

Sentz, K., & Ferson, S. (2002). Combination of evidence in dempster-shafer theory.

Shafer, G. (1992). The dempster-shafer theory. *Encyclopedia of Artificial Intelligence*, 330-331.

Shamailian, J. H., Baird, H. S., & Wood, T. L. (1997). A retargetable table reader. In *Proceedings of the Fourth International Conference on Document Analysis and Recognition* (p. 158-163). IEEE.

Silva, A. C. e. (2007, September 23-26 2007). New metrics for evaluating performance in document analysis tasks. In *Proceedings of the 9th International Conference on Document Analysis and Recognition (ICDAR).* Curitiba, Brizil: IEEE.

Silva, A. C. e. (2009, July 26-29 2009). Learning rich hidden markov models in document analysis: Table location. In *Proceedings of the 10th International Conference on Document Analysis and Recognition (ICDAR).* Barcelona, Spain: IEEE.

Sylwester, D., & Seth, S. (2001). Adaptive segmentation of document images. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition* (p. 827 - 831). Seattle, Washington.

Tengli, A., Yang, Y., & Ma, N. L. (2004, August). Learning table extraction from examples. In *Coling 2004.* Geneva, Switzerland.

Thompson, M. (1996, May). A tables manifesto. In P. Gennusa (Ed.), *Graphic Communications Association: SGML Europe '96* (p. 151-153). Munich, Germany.

Thulke, M., Margner, V., & Dengel, A. (1999). A general approach to quality evaluation of document segmentation results. In *DAS '98:*

Selected Papers from the Third IAPR Workshop on Document Analysis Systems (p. 43-57). London, UK: Springer-Verlag.

Tijerino, Y. A., Embley, D. W., Lonsdale, D. W., & Nagy, G. (2003). Ontology generation from tables. In *Proceedings of the 4th International Conference on Web Information Systems Engineering (WISE 2003)* (p. 242-249). Rome, Italy: IEEE Computer Society.

Tubbs, K. M., & Embley, D. W. (2002). Recognizing records from the extracted cells of microfilm tables. In *ACM Symposium on Document Engineering* (p. 149-156). McLean, Virginia.

Turolla, E., & Belaid, A. (1996). Form item extraction based on line searching. In *LNCS, volume 1072* (p. 69-79). Springer-Verlag.

Wang, X. (1996). *Tabular abstraction, editing and formatting.* Unpublished doctoral dissertation, University of Waterloo, Waterloo, Canada.

Wang, X., & Wood, D. (1993a). An abstract model for tables. In (p. 231-237).

Wang, X., & Wood, D. (1993b). Tabular abstraction for tabular editing and formatting. In *Proceedings of the 3rd International Conference for Young Computer Scientists.*

Wang, Y. (2000). *Document analysis: Table structure understanding and zone content.* Unpublished doctoral dissertation, University of Washington, Seattle, Washington, USA.

Wang, Y., & Hu, J. (2002a, August). Detecting tables in html documents. In *The Fifth IAPR International Workshop on Document Analysis Systems* (p. 249 ff.). Princeton, New Jersey, USA.

Wang, Y., & Hu, J. (2002b, May). A machine learning based approach for table detection on the web. In *Proceedings of the Eleventh International World Web Conference, WWW2002* (p. 242-250). Hawaii, USA.

Wang, Y., Phillips, I., & Haralick, R. (2001, September). Automatic table ground truth generation and a background-analysis-based table structure extraction method. In *Proceedings of the Sixth International Conference on Document Analysis and Recognition* (p. 528-532). Seattle, Washington, U.S.A.

Wang, Y., Phillips, I., & Haralick, R. (2002). Table detection via probability optimization. In *DAS02* (p. 272 ff.).

Watanabe, T., Luo., Q., & Sugie, N. (1993). Structure recognition methods for various types of documents. *MVA*, *6*(2-3), 163-176.

Watanabe, T., Luo, Q., & Sugie, N. (1993). Toward a practical document understanding of table-form document. In *Proceedings of the Second International Conference on Document Analysis and Recognition (ICDAR 1993).* Tsukuba City, Japan: IEEE.

Watanabe, T., Luo, Q., & Sugie, N. (1994). Knowledge for understanding table-form documents. In *IEICE Transactions on Information and Systems* (p. E77-D(7)).

Watanabe, T., Luo, Q., & Sugie, N. (1995, April). Layout recognition of multi-kinds of table-form documents. *PAMI*, *17*(4), 432-445.

Wei, X., Croft, B., & McCallum, A. (2006, September). Table extraction for answer retrieval. In *Information Retrieval* (p. 589-611). Springer Netherlands.

Whittaker, H. (2005). Social and symbolic aspects of minoan writing. In *European Journal of Archaeology* (pp. Vol. 8, No. 1, 29-41 (2005)). European Association of Archaeologists, SAGE Publications.

Wong, W., Martinez, D., & Cavedon, L. (2009). Extraction of named entities from tables in gene mutation literature. In *BioNLP 2009 Workshop* (pp. 46–54).

Wu, C.-H., & Lee, S.-J. (1993, May). An object-oriented expert system for local area network design. In *Proceedings of the Fifth International Conference on Computing and Information* (p. 321 - 326).

Xi, D., & Lee, S. (1998, November). Table structure extraction from form documents based on gradient-wavelet scheme. In *Document Analysis Systems:Theory and Practice, Third IAPR Workshop* (p. 240-254).

Yoshida, M., Torisawa, K., & Tsujii, J. (2001). A method to integrate tables of the world wide web. In *International Workshop on Web Document Analysis* (p. 31-34).

Zanibbi, R., Blostein, D., & Cordy, J. R. (2004, March). A survey of table recognition: Models, observations, transformations, and inferences. *International Journal of Document Analysis and Recognition (IJDAR)*, *7*(1), 1-16.

Zou, J., Le, D., & Thoma, G. R. (2007). *Online medical journal article layout analysis.*