Automatic Table Layout and Formatting

A thesis submitted for the degree of

Doctor of Philosophy

by

Mihai Valeriu Bilaucă

Department of Computer Science and Information Systems

University of Limerick

Supervisor: Dr. Patrick Healy

Submitted to the University of Limerick, July 2012.
Abstract

In this thesis we address the problem of tabular layout with a focus on automatic table layout optimization. In a world where documents are produced in both physical format and also delivered electronically to an increasing range of smart devices, we draw attention to the fact that authoring tables is still difficult, frustrating, prone to errors and automated optimal table layout is basic or non-existent. We show through examples that while tables are well supported by document authoring tools, table layout suffers from many usability bottlenecks and inconsistencies. The current tools provide powerful features for controlling the presentation parameters but they provide limited or no features for controlling the logical table structure. In this context we survey table creation in several popular document authoring programs and review previous work on logical table models. We then present a combinatorial optimization modelling method for the table layout optimization problem, the problem of minimizing a table’s height subject to it fitting on a given page (width). We present three models of the problem for tables with spanning cells and inner tables and evaluate their performance using mixed integer programming, constraint programming and local search. When a table is too large to be displayed in one page, a multiple-objective optimization problem arises for automated table layout, finding suitable ways of splitting the table across multiple pages without affecting the meaning of the data. We present two mixed integer programming optimization models for finding arrangements of columns in pages such that the number of pages is minimized and, at the same time, the changes to the column or row ordering or grouping is also minimized. To demonstrate our results, we developed a Table Drawing Tool prototype which implements an automated solution for the table layout optimization problem using a mathematical modelling method.
Declaration

I hereby declare that this thesis is entirely my own work, and that it has not been submitted as an exercise for a degree at any other University.
Dedic aceasta teza parintilor mei
Acknowledgements

I would like to express my unreserved gratitude to my supervisor Dr. Patrick Healy for his expert guidance, support, kind advice and infinite patience throughout my studies at University of Limerick. It was only through his expert insight in computer science, optimization techniques and skilful guiding in my research that I was able to complete this work. I wish that everybody is as fortunate as I was in having a supervisor that can take you from the darkest corners and lead you in the right direction.

I would like to thank my examiners Professor Barry O’Sullivan from University College Cork, Dr. J.J. Collins from University of Limerick for their patience, support and expert advice on how to improve this thesis. Also I wish to thank Dr. Norah Power from University of Limerick for her patience and encouragement as chair of the examination board.

I would also like to thank Prof. David L. Parnas for motivating, supporting and guiding me in the initial phase of my research at the Software Quality Research Laboratory. It was an unique opportunity to learn at SQRL how software engineering relates to traditional engineering and how to identify valuable scientific work. In addition, I wish to thank Dr. Tom Arbuckle, Dr. Markus Clermont and Mr. Simon Marr for the constant review of my work, insightful comments and aid during my time at SQRL.

I would like to thank Thierry Benoist of Bouygues e-lab for his support during our collaboration on local search techniques.
The Department of Computer Science and Information Systems, University of Limerick has been an excellent environment for pursuing my research with unlimited opportunities and open doors in the scientific world. I would like to thank Ms. Annette McElligott, Head of Department for her constant support during my research and for all the help I received from the C.S.I.S. staff Ms. Ita Page, Ms. Gemma Swift and Ms. Nuala Kitson.

To Ruth Guiry and Nik Nikolov thank you for your advice, support and encouragement for completing this work. To my friends and colleagues Patricia O’Dwyer, James Sweeney, Aedin Foley, Patricia Wright, Breda Tierney, Jason Murphy and Gay Kilgarriff many thanks for your friendship, support and for many years of patiently listening to me and my academic work.

I am most grateful for the unlimited patience, constant support and encouragement that I received throughout the years from Grace Hourigan who knows maybe more than anybody else all the hard work and long hours required to finish this thesis.

I would finally like to thank my family for the unconditional love and support they always surround me with. I take this opportunity to thank my parents: my mother who always believed and supported all my ideas and to my father who always encouraged me to take on any challenge and to pursue the harder but most rewarding way.

I wish to acknowledge the partial financial support received from Science Foundation Ireland under SFI grant 01/P1.2/C009.
## Contents

1 **Introduction** ................................. 1
   1.1 The role of WYSIWYG tools .................. 4
   1.2 Automatic table layout optimization ......... 5
   1.3 The problem of paginated tables ............. 7
   1.4 Contributions .................................. 9
   1.5 Publications and other related work ........ 11
   1.6 Thesis outline .................................. 12
   1.7 Table Layout Performance of Document Authoring Tools ........ 12

2 **Related Work** ............................... 25
   2.1 Logical Table Models ......................... 26
      2.1.1 Tables as Grids ............................ 28
      2.1.2 Structure, Content and Visual Editing .. 29
      2.1.3 Collections of Logically Connected Data Items .... 29
      2.1.4 Tables as Sets ............................. 30
      2.1.5 Tables as Indexed Sets - Tabular Expressions .... 30
      2.1.6 Summary .................................... 31
   2.2 Combinatorial Solutions for Table Layout .... 31
      2.2.1 A Linear Inequality Solver ............... 31
      2.2.2 An Accelerated Branch-and-Bound Algorithm .... 32
      2.2.3 Convex Hull and Weighted-sum Optimization .... 33
List of Tables

1.1 Summary of table formatting features. ........................................... 23
2.1 Solutions for automatic table layout ............................................ 38
3.1 MIP model run time ................................................................. 67
3.2 MIP model run time for fixed number of cell configurations .............. 70
4.1 Run time for MIP and TSC models ............................................. 87
4.2 Summary of TSC running time (s) for 10x10x6 tables ......................... 96
4.3 Summary of TSC running time (s) for 20x20x6 tables ....................... 97
4.4 Summary of TSC running time (s) for 30x30x6 tables ....................... 98
4.5 Summary of TSC running time (s) for 40x40x6 tables ....................... 100
4.6 Overall TSC average running time (s) ....................................... 100
4.7 Optimal height ratio for 10x10x6 tables ................................... 102
4.8 Optimal height ratio for 20x20x6 tables ................................... 103
4.9 Optimal height ratio for 30x30x6 tables ................................... 103
4.10 Optimal height ratio for 40x40x6 tables ................................... 103
4.11 Optimal vs maximum table height ratios ................................... 104
4.12 Run time for TSC, CP and LSP models - Part 1 ........................... 106
4.13 Run time for TSC, CP and LSP models - Part 2 ........................... 107
4.14 Run time for the Table Split model ......................................... 120
5.1 Table caption format .................................................................. 150
5.2 Exceptions in TDT
# List of Figures

1.1 The distance between various cities and towns in Ireland [89] . . . . . . . . . . 2
1.2 A circular table . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 3
1.3 Cell configurations . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
1.4 A7 Documentation Table 3.4-b . . . . . . . . . . . . . . . . . . . . . 15
1.5 A7 Documentation Table 4.3.1-c . . . . . . . . . . . . . . . . . . . . . 15
1.6 TFM A stack of limited range and depth . . . . . . . . . . . . . . . . 15
1.7 Rotated content in Word is misplaced . . . . . . . . . . . . . . . . . . 16
1.8 Rotated text in LibreOffice is incorrectly sized . . . . . . . . . . . . . . 17
1.9 FrameMaker 7.0 misplaces rotated equations . . . . . . . . . . . . . . 17
1.10 Row height in Word requires adjustment . . . . . . . . . . . . . . . . 18
1.11 Borders in Word . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 19
1.12 Borders in LibreOffice . . . . . . . . . . . . . . . . . . . . . . . . . . 19
2.1 A sample table from the Chicago Manual of Style . . . . . . . . . . . . . . 27
3.1 Example of cell configurations . . . . . . . . . . . . . . . . . . . . . . . 52
3.2 Running time for a 3x3 table and up to 200 words per cell . . . . . . . . 65
3.3 Running time for a 10x10 table and up to 6 words per cell . . . . . . . . 66
3.4 MIP model running time for a 3x3 table . . . . . . . . . . . . . . . . . . 69
3.5 MIP model running time for 10x10 tables . . . . . . . . . . . . . . . . . 69
3.6 MIP model running time, tables with cells with fixed number of words 71
4.1 Table layout with spanning cells and minimum height . . . . . . . . . . . . 75
5.18 A tabular expression with rectangular layout ................. 148
5.19 A rectangular layout consists of 5 grids and 4 corners ........ 151
5.20 A Documentation Table 3.4-b ........................................ 158
5.21 A Documentation Table 3.4-b rendered with TDT .............. 158
5.22 A Documentation Table 4.3.1-c ........................................ 160
5.23 A Documentation Table 4.3.1-c rendered with TDT .............. 160
5.24 A 18x6 table with minimum height computed by TDT .......... 162
Chapter 1

Introduction

Presenting information using tables is a very efficient yet simple way to express complex relationships between data items in a limited physical space. Tables allow the reader to find precise answers to complex queries that would otherwise be difficult to answer if the data would be described in plain text. The most common function of a table is to identify the precise value of a data item at the intersection of a row and column. But tables are very efficient in answering more difficult questions: how two or more data items compare to each-other, what is the trend of a data range, what is the minimum or maximum value, what is the value of a domain function applied to a data range, etc. It is easy to see why tables are used in all disciplines and domains, from simple letters to scientific papers. Thus, it is no surprise that they are well supported in all modern document authoring tools.

While most of the time tables are represented as rectangular arrangements of data items in rows and columns they can come in many forms and shapes. For example the table in Figure 1.1 published by Patricia Wright [89] has an unusual aspect with headers displayed on the bottom row while the data items are rotated by almost 30 degrees. This type of table which shows the distance between cities are usually displayed using a rectangular shape with cities displayed in both horizontal and vertical headers and the data items mirrored diagonally.
Tables can take other shapes and in Figure 1.2 we show how a circular table can be represented. The information in this table could also be represented by a graph. Wang also highlights that tabular information can be displayed using other graphical methods such as bar graphics, line graphics, pie charts, etc. The shape of the table is decided in the design process when the author has to select the shape that will best reflect the logical relationship between the data items which will allow the reader to easily locate the data and understand its meaning. Readers are not only interested to find some value in a table, e.g. the distance between Dublin and Cork in Figure 1.1, they also want to evaluate and compare data to find answers to a range of questions. In some of cases, tables are not the most effective way of allowing the reader to compare data and therefore other graphical elements could be utilised: graphs, flowcharts, bar-charts, etc.

The current document authoring tools allow designers to create tables that describe complex relationships which otherwise are very difficult or even impossible to describe in words. In this thesis we show that creating such tables is still tedious, maintaining the documents can be hard, data reuse with other tools almost impossible, and automatically finding optimal layouts is non-existent. The statement that
Allan Heydon and Greg Nelson [43] made in 1994 about authoring documents with figures can also be applied to tables in 2012: if tables are “worth a thousand words to the reader, why is its cost to the writer more like ten thousand?”

With the aim to improve the current situation, our focus is to answer the question: can optimal table layout be automatically computed? Therefore we present how optimal solutions for a number of table layout problems can automatically be determined. We focus mainly on the typographical aspects of tables; human factors, styling or any aesthetic criteria are not within the scope of this research. To the best of our knowledge, this is the first work that shows how the table layout problem – the problem of finding the minimum table height for a given page width – can be modelled to find complete solutions that are proven to be optimal. Because of the lack of benchmark data for table layout we developed a suite of table instances that can be used by the research community in the evaluation of future work. We review the current document authoring tools to highlight their inconsistencies and especially the lack of optimal table layout automation features; we suggest a number of combinatorial optimisation methods for finding optimal table layouts and present an evaluation of their performance together with a statistical analysis of the range of optimal values; finally we present a table formatting tool prototype in order to demonstrate how these techniques can be applied in practical applications.
1.1 The role of WYSIWYG tools

Authoring tables is hard and the need to improve the table layout tools has been highlighted for many years. As WYSIWYG tools have become increasingly popular they provide powerful features for controlling the presentation parameters: Microsoft Word, LibreOffice, Pages are but three of the most popular document authoring tools that provide support for tabular format.

Using these tools high quality tables can be created. Nonetheless the author still has to deal with a number of inconsistencies: controlling borders and border styles can be unpredictable, serious layout issues occur when rotating cell contents or even when basic operations are required such as resizing columns; inner tables and mathematical expressions are not always supported and features for splitting wide tables are not available in popular WYSIWYG tools. This is one of the main causes for the delays when authoring tables as we show in Section 1.7.

Another weakness of these tools is the poor support for exchanging data and formatting styles. If tables from existing documents formatted with one tool need to be edited in a different document authoring tool, the author almost always has to start again with plain data and reformat the tables. This is also true even when the same document format is used such as ODT, DOC or DOCX. For example, if a document is produced in Microsoft Word, saved in DOCX format and then opened in LibreOffice, in many instances, table formatting is altered and needs to be reviewed. This indicates that even if there are agreed standards for exchanging documents poor implementation and the use of non-standard elements in the document format causes formatting issues and thus disruptive delays when authoring documents and tables.

A common problem with these tools is that the authors face difficult tasks when operations at the logical structure of a table are required. WYSIWYG tools provide limited or no features to swap, for example, the vertical and horizontal headers or to easily change how data is grouped in categories and sub-categories. While some
of these features are available in spreadsheet editors or advanced desktop publishing software the author still has to deal with integrating the output of these tools in the original document. This creates a new range of problems such as software incompatibilities, proprietary formats or loss of formatting and presentation work.

1.2 Automatic table layout optimization

Tables can often be larger than the available page space and the user must adjust the table layout until it can fit in the page. This process is time consuming as any change in the layout of a cell, row or column impacts the overall table layout. The process is also prone to errors because the table designer, in an effort to find a suitable layout, may unintentionally and unconsciously change the meaning of the data. Automating this process has been the subject of research in the last two decades and especially in recent years.

One of the problems that has gained attention is finding the table with the minimal height for a given page width. This table layout problem is not easy to solve, especially when the table contains text. A cell that contains text can take on any of the discrete number of bounding boxes given by the way the text can be arranged in lines. For example, the cell presented in Figure 3.1 has three configurations. If a cell contains an image, the cell has only one configuration given by the dimensions of the image; if the cell contains an inner table, its set of configurations is given by the set of all possible configurations for the inner table. The table layout problem is to determine for each cell the configuration that should be selected in order to determine an overall table layout with a minimum height for a given page width. As the table can change when the shape of any cell changes the table has an exponential number of possible shapes. For example, selecting a narrower cell configuration can increases the cell’s height and thus the row height and as a result the overall table height. It is thus unsurprising that this problem belongs to the NP-hard class
The complexity of the problem becomes even more evident when the table contains spanning cells or inner tables.

It can be argued that finding a solution to this problem is not always required. Instead, it is perhaps sufficient to find a layout that satisfies positioning and aesthetic constraints which is acceptable for the author. However, there are many applications where finding the table layout with the minimum height for a given width is desirable. For example, if a large table must be included in a book, knowing the minimum height layout that the table can be printed in the selected page space can reduce costs. Thus, the running time required to find such a layout can be offset against the cost savings in the design or in the printing process. In dynamic rendering of online content and in Variable Data Publishing (VDP) the same document is presented in multiple channels and in multiple page formats. Therefore a method for automated table layout that can optimize the table height for different page sizes is needed since it reduces the time required for formatting tables.

A number of solutions have been proposed for solving the table layout problem using a variety of techniques: systems of constraints described through linear or non-linear equality and inequality expressions, heuristics aiming to solve the problem in polynomial time or mixed approaches. These techniques focused only on the theoretical side of the problem or solving the problem as a constraint satisfaction problem instead of finding optimal solutions. Performance evaluation of these methods has been reported only for small size instances, if any evaluation was provided at all. For many years it has been suggested that constraint programming methods
can be used to find optimal solutions to layout problems in general and to user interface and table layout problems in particular. So far, constraint programming has not found its place in the commercially available document authoring tools. One of the reasons is that for large layout instances these methods proved to be unacceptably slow. However, in recent years the combinatorial optimization techniques have evolved considerably and their performance has improved. This is the motivation behind this thesis where we show, for the first time, how the table layout problem can be modelled using these modelling techniques in order to find complete solutions which are guaranteed to be optimal. Finding optimal solutions is also important for evaluating the results of other heuristics and for establishing a set of benchmark tables for future evaluations.

While recent research focused on either algorithms for optimal solutions or heuristics suitable for applications requiring fast running times, there is also the argument that priority should be given to user constraints imposed by space limitations or other aesthetic criteria, e.g. equal column widths. An advantage of using combinatorial modelling for the table layout problem is that user-imposed constraints can be easily added to the model. Thus, the techniques presented in this thesis are not only interesting because they show how to solve the table layout problem but also because they allow for other user-imposed constraints to be embedded in the same model in a simple and efficient manner.

1.3 The problem of paginated tables

For traditional publishing houses the Internet has dramatically changed the publishing process. Documents are stored as XML data files and layout information is applied using XSLT transformations in order to obtain electronic versions of the documents. The ever growing number of devices that consumers use to access printed material in electronic format raises difficult problems in terms of dynamically format-
ting the material in a suitable format. This is caused by the wide variety of display capabilities and processing power, different screen resolutions and fonts available for these devices. Two formats are predominantly used for electronic publishing over the Internet: HTML and eBooks. However, using these standards, the level of control over the layout of the document is poor and publishers prefer to use the PDF format to deliver publications for both printed and electronic format. As opposed to HTML where a continuous layout model is implemented, PDF layout is focused around paginated content. Designers carefully select where various document elements are placed and they can precisely control how the final document is presented.

When creating documents that support pagination authors face the problem of formatting wide tables that span multiple pages. At present, very few document authoring tools provide features that deal with this problem. These features are mostly available in specialized desktop publishing tools but they are non-existent in the current popular WYSIWYG tools. A difficult problem that occurs in this context is how to optimize the layout of a wide table that must be split over multiple pages.

Authors often face the problem of finding how to group columns in pages in a way that will not affect the meaning of the data. This problem can easily be underestimated as a simple bin-packing problem where the columns are re-arranged in such a way that it minimizes the number of pages used. In practice, it is highly improbable that the order of columns can arbitrarily be changed throughout the table. Most likely, in order to maintain the meaning of the data, the columns must follow some arrangement rules, i.e. groups of columns must be arranged in a particular order or some columns must be kept together. The difficulty of the problem increases in practice when page count minimization is desired and therefore changing the order of columns is accepted but with some limitations so that the effect on the meaning of the data is also minimized. In a wide table, the columns can be grouped for aesthetic and readability reasons and the groups highlighted using for
example a different background colour. Keeping the groups together in one page can require as many pages as the number of groups. This will also lead to a lot of white space left in each page. Besides being uneconomical in the printing process it also slows down the reader by requiring browsing through multiple pages in order to understand the data. It also makes harder a comparative evaluation of the data. Therefore, it is desirable to find an arrangement of columns in pages that not only minimizes the page count but also minimizes the number of groups that are split across pages.

The problem of optimally paginating tables has been highlighted throughout the years \cite{12, 86} but, to the best of our knowledge, without an in-depth analysis or solutions being presented. We suggest in this thesis how to solve some of the problems raised in formatting wide tables using combinatorial optimization solutions. The advantage of using constraint based optimization methods for solving these problems relies in the ability to easily define and integrate custom arrangement requirements. Being able to add and remove constraints to the optimization process allows the user to explore the effect on table layout of various arrangement requirements and then to select the most suitable layout.

1.4 Contributions

The contributions of this thesis can be summarized from five perspectives. First, we review the current popular tools used to create tables and highlight a number of inconsistencies which are the main cause of delays, frustration and errors when authoring tables. We show the need to improve these tools by implementing a table model that allows operations at the table’s logical level and also automatic table layout optimization.

Second, we show how the table layout problem can be modelled using combinatorial optimization techniques that are guaranteed to find optimal solutions. We
discuss the problem for simple tables, then for tables with spanning cells and, finally, tables with inner tables. We first show how to model the table layout problem as a mixed integer programming (MIP) problem; then we show a constraint programming (CP) model and a Local Search model for this same problem. We present experimental results and provide performance evaluations of various methods for tables that contain large amounts of text and also for larger tables with up to 3,600 cells and over 12,000 cell configurations. To the best of our knowledge this is the first attempt to report on run time for large table sizes and specific (rather than heuristic) cell configurations. We claim, further, that our data has a greater degree of realism than those considered heretofore.

Third, we discuss the problem of optimally splitting a wide table across multiple pages. We suggest solutions for finding a grouping of columns in pages in such a way that minimizes the impact on the reader while minimizing the number of pages used. We show how this multi-objective optimization problem can be modelled using mixed integer programming techniques.

Fourth, we present a table drawing prototype – Table Drawing Tool or TDT – in order to show how automatic table layout optimization can be implemented using constraint optimization engines. We describe the typographical issues in developing a table formatting tool. In this prototype we use the IBM ILOG engine to implement all the models presented throughout this thesis. The prototype is used for the visual inspection of the solutions found in the optimization process without the limitations of existing table engines, for example the HTML table layout algorithm. By developing this prototype we are able to identify and outline the engineering problems that need to be addressed in order to bridge the gap between the theoretical aspect of our thesis and the development of a software solution that implements these theoretical results.

Finally, we developed a set of tables that can be used as a benchmark set and which are made available for further research in table layout optimization and table
formatting problems. As far as we are aware a set of benchmark tables does not exist at present. Our aim is to start, in cooperation with other researchers [39], the development of a benchmark set of tables that can be used in comparative evaluations of future work in this area. Our set of tables and additional information about table layout is available at

http://www.tabularlayout.org/

1.5 Publications and other related work

While some of the techniques, methods and results are presented for the first time in this thesis some of our work has already been published:

- Bilauca & Healy, “A New Model for Automated Table Layout” [19] at The ACM Symposium on Document Engineering 2010, Manchester, UK;


- Bilauca & Healy, “Building Table Formatting Tools” [21] at The ACM Symposium on Document Engineering 2011, Mountain View, California, USA;

We are collaborating with the Dept of CSSE in University of Melbourne and the Clayton School of IT, Monash University, Australia on an article for the Operations Research journal on table layout optimization. Also, for the past year I am involved in reviewing an article for the ACM Transactions on the Web journal.

Following the publication of our work, Innovation 24, a subsidiary of Bouygues group in charge of the distribution and the support of the commercial package LocalSolver, are using our model as a sample problem to demonstrate how LocalSolver can be used to solve the table layout optimization using local search techniques. The research and development on LocalSolver was conducted in partnership between Bouygues and Marseille Computer Science Laboratory (LIF), a joint research unit.
of Aix-Marseille University. Bouygues is a diversified industrial group with 130,000 employees in 80 countries and a revenue in 2011 of 32.7 billion. More information can be found at:


1.6 Thesis outline

In the remaining part of this chapter we review the capabilities of the current popular document authoring tools with a focus on table layout features. In Chapter 2 we discuss previous work on table layout and review the logical table models and automatic table layout using constraint optimization. We suggest in Chapter 3 combinatorial optimization solutions for the table layout problem followed in Chapter 4 by an insight into the more difficult table optimization problems such as tables with spanning cells, tables with inner tables and the problem of optimally splitting wide tables on multiple pages. In Chapter 5 we introduce TDT - Table Drawing Tool, the prototype used to evaluate the results of our optimization solutions. We draw our conclusions in Chapter 6 and show areas where more research is necessary. Sample table layouts generated with the Table Drawing Tool prototype – TDT – are presented in appendices together with the TDT input file format and TDT engine specifications. Full code of all the constraint programming models is also presented in appendices.

1.7 Table Layout Performance of Document Authoring Tools

In this section we discuss the user experience when drawing tables. We draw attention to the fact that authoring tables is still difficult and can be a frustrating and error prone exercise and that the development of high-quality table tools should be further pursued. This has also been highlighted by Hurst et al. [44]. They sug-
gest that due to the widespread use of a diversity of devices (PDAs, mobile phones, printers, monitors) the development of high-quality automatic layout tools should be the central concern of the document formatting community. We pursue this suggestion by precisely exemplifying where the current tools are failing. We survey table creation in several popular document authoring programs and identify usability bottlenecks, inconsistencies and we show through examples why authoring tables is still hard and time consuming using these tools.

In this survey we do not include specialized Desktop Publishing software tools (i.e. Corel Draw or QuarkXPress) because their availability is relatively limited compared with document processing tools and they also require specialized training and skills. We limit our survey to some of the more popular document editors that offer table layout features: Microsoft Word 2003/2007/2010, LibreOffice Writer 3, Adobe FrameMaker 7, Apple iWorks Pages 3, HTML including Cascading Style Sheets - CSS 2.1, \TeX and \LaTeX.\

One of the difficulties with tables is that they can be larger than the space available in the page. Thus, the user must reorganize the table’s content in such a way that it can fit in the page without losing the meaning of the data. This process is time consuming as any change in the layout of a cell will impact on other cells and thus on the entire table layout. It is also prone to errors because the table designer, in an effort to find a suitable layout, may unintentionally and unconsciously change the meaning of the data. It is ease to observe that the table layout process can present challenging problems to the designer who relies entirely on the ability of software tools to deliver the expected results. Unfortunately, as we show in this section, this is not always the case making table layout a labour intensive process.

In table layout there are difficult problems not only from a computational standpoint but also there are numerous engineering or technical problems. While research has focused on solving some of the computational problems, e.g. minimize the height of a table for a given page width, little work has been done in identifying and solving
the pure engineering problems encountered in table layout. It is easy to underesti-
mate the root cause of these problems and consider them to be software bugs. We 
believe that the cause of these problems is the lack of a common, well established, 
table layout model that specifies how tables should be drawn. In particular, there 
is no clear set of rules on how to deal with conflicting cases, for example, when 
borders with different attributes (thickness, colours, pattern) overlap, which border 
attribute should be given priority. This is the cause of many inconsistencies: the ap-
pearance of a table that has been designed with one tool is different when exported 
or viewed with different software tools.

Our aim is to discuss the user experience when drawing tables with the purpose 
of demonstrating that authoring tables is still difficult and can be a frustrating 
and error prone exercise. We do not intend to describe all the features that each 
individual tool may offer.

To conduct the study we needed a set of test tables. Unlike other domains where 
benchmarking data is available, there is no formally recognized set of tables that 
could be used for evaluating table editors. Therefore, we selected a set of tables 
from the documentation of the A-7E aircraft released by the US Naval Research 
Laboratory [5]. One of the selected tables is presented in Figure 1.4. These tables 
are interesting from a layout point of view because they are wide and need to be 
split across pages horizontally and vertically; they contain merged cells, rotated 
content and inner tables as shown in Figure 1.5. The table presented in Figure 1.6 
published by Parnas and Dragomiroiu [70] is interesting from a layout perspective 
because while the information has been presented using merged cells it can also be 
drawn, without changing its meaning, by using inner tables or a graph. We consider 
these tables especially relevant because they have been used in documenting real 
software.

For the purpose of this study we replicated in each of the selected tools the 
tables presented in the documentation of the A-7E aircraft. Presented below are
Figure 1.4: An illustration of Table 3.4-b from the A-7E Documentation.

Figure 1.5: An illustration of Table 4.3.1.c from the A-7E Documentation.

Figure 1.6: A stack of limited range and depth as published by Parnas and Dragomiroiu
Figure 1.7: Rotated content in Word is misplaced. To correct this effect each column must be resized. Resizing only the second column will not fix this issue.

the principal findings of our survey. Table 1.1 is a summary of the table formatting features:

**Rotate content.** In a table with wide header cells and narrow content cells rotating the content in the header reduces substantially the table’s width. In Word when the text is rotated by setting the *Text direction* to either *bottom-to-top* or *top-to-bottom* the text is misplaced, as illustrated in Figure 1.7. In Pages, cell content cannot be rotated and therefore it was impossible to draw a table with 24 columns as presented in Figure 1.4. In some cases LibreOffice can resize the text incorrectly when the text is rotated as we show in Figure 1.8. In FrameMaker rotated equations are placed outside cells boundary as shown in Figure 1.9. When rotating text in a cell, the cell’s height is increased correctly to the length of the text because the cell’s height is set to expand to a default value of 14 inches. However, the effect can be unpleasant because it impacts on the table’s layout height.

**Resizing.** In spite of how tables are presented to the user, changing the width of a column is not always easy using the WYSIWYG editors. In Word, LibreOffice and Pages, when the author increases the width of a column, the extra space
is created by narrowing the column to the right while maintaining table width. This happens even when there is still space available on the page where the table can expand. This causes a chain effect: the height of the narrowed column increases to fit its contents which increases table height and this may add another page to the document. Thus, by simply performing a basic operation the author is facing a new set of layout problems which is time consuming and distracting. While there are solutions to correct the situations described above (i.e. pressing the shift key while resizing, or additional resizing operations) the author has to find a number of procedures to correct an inconsistent layout. Only Adobe FrameMaker expands the table width when a column width is increased. FrameMaker correctly shifts all the columns to the right in order to accommodate the wider column and therefore the integrity of the layout is maintained.

**Padding & Margins.** Cell padding and cell margins cannot be controlled in neither LibreOffice, FrameMaker nor Pages. To control padding in LaTeX the author must develop certain techniques. One of these techniques is to adjust the cell’s height in such a way that will provide enough padding. The cell’s height can be adjusted by setting a vertical rule’s height and depth. In \LaTeX
Figure 1.10: The row height required manual adjustment to fit in the text. Also, each column needs to be resized. Chain effect: columns are narrowed, tables height is increased, pages are added.

to control cells’ padding \@-expressions can be used. However this is defined in the column definition section of the tabular environment and not at the cell level.

Borders. Drawing cell borders needs careful consideration. When a thick border is used for a cell, some of the tools will add the space required by the border to the cell’s dimensions. Other tools will add only half the border space to the cell’s dimensions with the other half taken from the cell’s content space. If these cells have no margin, then the adjacent borders are merged, or they collapse. These effects need to be identified and clearly stated. All the tools provide support for controlling cell borders. Only in \TeX controlling borders is not easy; “People who know how to make ruled tables are generally known as \TeX Masters” [51]. It is interesting to observe that Microsoft Word deals with borders in a different way even in the same cell. In Figure 1.11, the entire top border space and only half the left and right border space are included in the cell’s content space while the entire bottom border space is placed outside.
Borders in Word are aligned differently even for the same cell. This in turn increases the table’s width with half the border’s width. This can cause a series of layout problems (i.e. misalignment, misplacement, etc.) especially if the table’s width must be of a precise fixed size. LibreOffice does not deal with borders consistently, either. In Figure 1.12 the top border of cell marked A1 is placed inside the cell space while the left border is centered on the table border; the text in cell B2 is covered by the border; when the horizontal and vertical borders are joined the vertical borders seem to have priority which causes an un-aesthetic joining point.

Auto-layout. All the tools implement some auto-layout algorithm. However, none of the tools provide any layout optimization features. A basic auto-layout algorithm is described in the CSS specifications [81]. It is interesting to note that even in the World Wide Web Consortium specification document it is mentioned that user agents ”can use any other algorithm even if it results in different behavior” and that ”the remainder of this section [auto-layout] is non-normative”. This raises the question: should such specification documents simply describe the behaviour of existing tools? We believe that instead, the community should explore problems, find solutions and develop standards that the tools must follow.
**Inner tables.** While inner tables can be inserted in Word, if the parent cell’s text direction is changed then the layout becomes inconsistent and resizing the column or selecting the text inside the cell becomes hard or impossible. If the cell that contains both text and an inner table is large enough, Word will rotate the text but not the table even if the text direction has been changed for the cell. LibreOffice allows a combination of text and tables to be inserted in a cell. When the cell’s direction is changed, LibreOffice correctly rotates both the text and the table. FrameMaker and Pages does not allow inner tables. As a work-around, these could be avoided by splitting or merging cells but this affects the layout of the outer table. Alternatively, the user can generate the inner table in a PDF file and then place this file in the cell. \LaTeX{} also does not allow inner tables or inner images but \LaTeX{} solves these issues. Using \LaTeX{} images can be included in table cells using `\includegraphics`. When the image is rotated, scaled, etc., the table layout is displayed correctly.

**Equations.** With the exception of Pages and most HTML browsers all the tools allow equations to be inserted in tables. FireFox can natively render equations defined in MathML while other browsers (Internet Explorer, Google Chrome, etc.) require plug-ins that render equations as images. Support for equations in MathML format is available starting with Internet Explorer 9.0 as part of its support for HTML 5 but rendering equations is problematic and a number of bugs have already been reported.

**Splitting tables.** If the layout does not fit vertically on one page then the table is split on subsequent pages. With the exception of \LaTeX{} all the surveyed tools support long tables splitting but none cater for splitting wide tables. In \LaTeX{} tables cannot be split either horizontally or vertically\footnote{\cite{54}} but this problem has been solved in \LaTeX{} by using packages such as `supertab`, `longtable` or `tabularx`; Currently none of the surveyed tools support splitting of wide
tables.

**Grouping rows and columns.** Groups of rows (and similarly, columns) are used as a visual guide for finding information in a table. It has been observed that in a wide table with many columns it is hard to visually follow a row to read a value that is placed far from the header. Readers can improvise by using, for example, a ruler to help them follow the values displayed in a column although this object will likely obstruct parts of the table. In order to visually guide the user the information could be grouped in a logical manner. Thus, groups of rows or columns can be formatted differently. Word, LibreOffice and Pages allow the user to set the foreground and background colours without allowing the definition of row and column groups. This feature is only supported by FrameMaker and HTML.

**Templates.** Templates are sets of formatting parameters that can be applied to tables in order to speed up the design process. The user selects a template instead of specifying all the formatting parameters. Templates can be used to achieve a consistent table look throughout the document. Changing how tables are formatted in a document should be as easy as changing the template.

**Headers and Footers.** When a table is split on multiple pages, headers and footers are repeated on all the subsequent pages. All tools allow for header and footer rows with the exception of Word and LibreOffice which only allow for a header row. Pages allows the author to set a header row and a footer row and it is the only tool that allows the author to set a header column;

**Table Caption.** In Microsoft Word and LibreOffice, tables can have captions but there is no direct link between the table layout and the caption. That means that the caption still exists in the document even when the table is deleted. This occurs even when the caption is specifically entered for the table. In
Pages, there is no provision for table captions. Only \TeX{} and \LaTeX{} allow full control of the caption’s position. In Word, LibreOffice and FrameMaker the caption can be placed to the top or at the bottom of the table, but not to the left or to the right. In HTML and CSS specifications 2.1 the \texttt{caption-side} element can have the \texttt{top} and \texttt{bottom} values but this is expected to be changed in CSS 3, the next release of the CSS specifications. At present, the CSS Tables Level 3 module has been marked as “inactive” on http://www.w3.org/Style/CSS/current-work/.

**Table Summary.** This is an accessibility feature that allows the table content to be described in plain language in order for a text-to-speech engine to provide voice rendering of the table description. Only HTML specifications allow for a table summary to be defined.

As a result of this study the following conclusions can be drawn:

- Table editing is hard, time consuming, and error prone. Basic operations such as resizing cells are difficult using even popular packages;

- Unlike other fields, where there are agreed collections of test data, for table layout there is no library of tables that could be used to evaluate table editors or other algorithms;

- There are no agreed standards on how to solve technical problems such as border conflict resolution;

- The information cannot be easily transferred between table editing tools. All the effort deployed in authoring tables using a tool is lost if another tool is used;

- Wide tables cannot be split across pages. None of the surveyed tools cater for this option;
Table 1.1: Summary of table formatting features.

<table>
<thead>
<tr>
<th>Feature</th>
<th>MsWord 2003/2007/2010</th>
<th>Libre Office 3.4</th>
<th>FrameMaker 7.0</th>
<th>iWorks Pages 3.02</th>
<th>HTML/CSS</th>
<th>LATEX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resizing</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Rotate content</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Padding &amp; Margins</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Borders</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Auto-layout</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Inner tables</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Equations</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Split wide tables</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Split long tables</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Grouping rows/col.</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Templates</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Table Header</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Table Footer</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Column Header</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Caption</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Caption Position</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Table Summary</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

*a“Tricks” are required
*bPlugin required
*cPrint only
*dCSS only
*eMacros required
• Visual indicators and grouping rows and columns are important in large tables. They eliminate the need of using physical guidance tools to follow data in large rows/columns;

• Tools are essential when authoring tables and the development of high-quality table tools should be further pursued.

• While not all the features evaluated in this section are supported in \( \LaTeX \) its strength relies in the ability to include packages that could resolve these problems.
Chapter 2

Related Work

In the introductory section we analysed the features of the current table authoring tools. We now review previous work on tabular layout from two perspectives. Firstly, we present in Section 2.1 a review of work related to the logical table models because of its importance in table layout and because it is poorly supported by the current document authoring tools. We highlight the fact that the development of better software tools requires better logical table models. Secondly, in Section 2.2 we focus our review on automated table layout as a constraint optimization problem because of its relevance to our work.

In Section 2.3 we review recent work in combinatorial optimization with the purpose of providing background information about the optimization methods suggested in this thesis: mixed integer programming, constraint programming and local search.

A comprehensive review of the recent work in the automatic document formatting has been presented by Hurst et al. [44]. The review presents a summary of micro-typography problems such as kerning, paragraphing (line breaking) and line justification and macro-typography problems - document and page layout models, choosing and retargeting page layout. The focus of the review is on techniques that formulate the document layout problems as constraint optimization problems (dis-
crete, continuous and mixed problems). They also discuss the problems encountered in the table formatting process and a summary of the techniques proposed in recent years to solve these problems.

2.1 Logical Table Models

In recent years, as WYSIWYG tools have become increasingly popular they provide powerful features for controlling the presentation parameters of documents and tables. However, WYSIWYG tools provide limited, or no, features for controlling the logical table structure.

Research has shown [30, 85, 86, 87] that table tools should clearly separate the logical table structure from its presentation. That is, separating table topology rules (arrangement and ordering of data items) and table typography (visual parameters).

When editing a document, the author is primarily concerned with the document’s logical structure and only at a later stage with its presentation. Similarly, the author needs to define the logical table structure independent of its presentation. The advantages of separating the logical structure of a table from its physical layout have already been identified: it is easier to define and maintain how a table is represented, there are reduced risks of introducing errors during the authoring process, and it is easier to change how the data items are organized or what filters can be applied in a data set. Similar to document editing where the author is initially focused on defining the logical structure of a document, in tabular editing the author has to make initial decisions that will impact how well the data is represented; what headers will be used, how the rows and columns are grouped in a logical manner or what ordering can be applied to the data items are all initial concerns of table authors. If any of these decisions must be changed, having the ability to modify the logical model of a table will make the changes much easier to implement. For example, in the table presented in Figure 2.1 if values for a new year 2007 need to
be displayed two rows need to be added, one for each group, but there is only one logical entry added to the data domain. If this table had multiple language entries a row would have had to be added for each language.

Using current WYSIWYG tools the author faces difficult tasks when the logical structure of a table must be changed.

For example, when a short table is too wide to fit in the available space, the recommended solution is to swap the table headers and present the information in a vertical manner. With WYSIWYG tools the author has to manually perform this task by copying, moving or re-entering content. This can be time consuming and at a high risk of introducing errors. Using a tool that allows the author to alter the logical structure of the table this task is simply a matter of selecting the horizontal and vertical headers. If computer programs are used to author tables then it should be a lot easier to author tables than it is with the current tools.

Another example that shows how important is the ability of changing a table at
a logical level is the need to modify the criteria by which data items are grouped in categories. Wright [89] mentions a study by Barnard, Morton, Long & Ottley which showed that people were able to find items in a table of products in a considerable shorter time when the data was grouped in familiar categories than when the products were simply arranged in alphabetical order. Later they showed that the improvement was not only caused by the simple presence of categories. In fact, the choice of categories that users can easily understand, such as fish, meat, vegetables, was critical. Selecting other criteria for grouping the information on some arbitrary basis (such as the length of words, or the size of the product) would have not been likely to improve the performance to the same extent. Current WYSIWYG tools provide almost no functionality to assist the author, when needed, to change the criteria used in categorizing data in an existing table. The entire table must be recreated taking into account the new data structure and any previous work on authoring the table will be lost. WYSIWYG tools only allow the author to use functions such as copy, cut and paste of multiple cells, rows and columns but such operations are inefficient and can introduce errors.

A number of authors have proposed various logical table models which are reviewed in this section. It has been accepted by all that it is hard or almost impossible to define a general logical model that covers all types of tables without clearly defining what a table is. This is not easy, as Vanoirbeek explains [85], because it is not easy to distinguish between tables and other components of a document that follow a tabular pattern.

2.1.1 Tables as Grids

Beach [12] defines the logical table model as a document object in the Tioga document manipulation system. Each document object has a set of dimensions and it must know how to format itself. Thus, it must provide a function for determining its layout and a function for rendering the object. The table model proposed by
Beach separates the *table topology* from the *table geometry*. Beach points out that a hierarchical model is not ideal for representing tables because tables are not always hierarchical. Therefore he defines the table topology - the arrangement or ordering of data items - based on a grid. The table geometry is computed based on the dimensions of the table entries and the table topology.

### 2.1.2 Structure, Content and Visual Editing

Cameron [30] studied the mental processes involved in authoring tables. He identifies three processes: *structure editing*, *content editing* and *visual editing*. During structure editing, the authors define the logical dimensions of the data, its classifications and other complex operations such as data transposition or re-arrangement of classification data entries. During content editing authors simply alter the individual data entries which can be numeric, textual, graphics or mathematical expressions. Formatting parameters such as text font, colours, rules and alignment are set during the visual formatting process.

### 2.1.3 Collections of Logically Connected Data Items

In her 1989 study, Christine Vanoirbeek [85] shows through examples the importance of the separation between the logical and the presentation layers. She suggests a logical model where tables are a collection of data items logically connected in a multidimensional space. Each dimension is divided in *rubrics* which classify data items based on some established criteria. The physical layout of a table is a projection from the multidimensional logical space into a two-dimensional visual space. This model is then further expanded by the introduction of *sub-rubrics* and *merged items*. A sub-rubric is utilized as a way to sub-divide the logical dimension. Merged items are used for data items which are connected to multiple rubrics. The table model is then demonstrated in a prototype for formatting an expandable document model.
The document’s structure is defined using a language based on attribute grammars. This allows formatting parameters to be associated to the logical components of the document.

2.1.4 Tables as Sets

Wang [86] proposed a table model that separates the layout structure from the logical structure. The layout structure defines the tabular topology and the typographic style. By using well understood mathematical expressions the logical model proposed by Wang [86] separates its representation and implementation from its mathematical formulation. An abstract table is defined as a tuple \(< C, \delta >\) where \(C\) is a set of categories (a set of labelled domains) and \(\delta\) is a mapping of data entries in the table. Wang also divides the logical operations on abstract tables in \textit{table operations, category operations} and \textit{label and entry} operations. Table operations modify the number of categories in a table thus affecting the number of dimensions and possibly its size. Category operations alter sub-categories (sub-headings) of a table, and label and entry operations deal with the content of the table. As the logical model is defined using unordered sets the ordering of categories and entries is seen as a topology concern and it is not included in the abstract model.

2.1.5 Tables as Indexed Sets - Tabular Expressions

Parnas expands the use of tables from simply organizing data to providing meaningful expressions which can be evaluated and validated. He shows how \textit{tabular expressions} - mathematical expressions in tabular format - can be used to precisely document software systems. In the logical model proposed by Jin and Parnas [48] indexation is used to separate the table semantics from the table appearance. Tables are defined using mathematical expressions, as an indexed set of grids and each grid is an indexed set of expressions. Thus a table is a triple \(< GS, I, x >\), with \(GS\) an
indexed set of grids, an index set $I$ and a mapping $x$ between $GS$ and $I$. A grid is a triple $<SetExp, J, y>$ that is a set of expressions $SetExp$, an indexed set $J$ and a mapping between $SetExp$ and $J$. To precisely define the meaning of the tabular expression a restriction schema and one or more evaluation schemas are provided.

### 2.1.6 Summary

Following this review, it can be observed that while the initial table programs were only dealing with basic presentation of tables research has highlighted the need for an abstract logical table model in order to deal with more complex structural operations.

The logical table model has evolved during the years from a simple grid to multidimensional logical structures such as trees suggested by Vanoirbeek [85], sets proposed by Wang [86] and indexed sets of mathematical expressions which can be evaluated and validated as proposed by Parnas [48].

### 2.2 Combinatorial Solutions for Table Layout

Various algorithms and strategies have been proposed during the years to automatically compute table layout with an increasing number of researchers focusing on combinatorial solutions. We will now survey these solutions and provide an overall summary presented in Table [2.1](#).

#### 2.2.1 A Linear Inequality Solver

Starting in 1985, Beach [12] was first to suggest a table layout algorithm that uses linear inequality constraints to describe the positioning relationships between table entries in a grid based table model. This method also has the advantage of allowing additional explicit constraints supplied by the table designer such as forcing columns of equal width. A linear inequality solver, *Juno*, developed by Greg
Nelson [67] is used in a table formatting prototype to solve the constraints system. Juno integrates constraint optimization in a WYSIWYG editor and an undo feature which is particularly useful in an interactive table formatting tool. Another useful feature is a slack variable used to indicate if the constraints system needs to be recomputed. The slack variable representing the extra space left between a table entry and the grid lines is introduced when linear inequalities are changed in linear equalities. For large tables when the number of constraints can grow very large, Beach suggests a number of strategies to reduce the size of the problem: removing redundant variables or, when possible, solving the constraints for rows and columns independently. Another strategy uses the observation that the position constraints for all the entries of a column can be replaced by only one constraint that uses the maximum left and right offsets. Sparse matrix techniques may reduce the space requirements of the solver because most of the constraint inequalities have only two variable terms. Beach concludes that large tables present a number of problems for which automatic layout computation is desirable: carry-forward table entries, automatic insertion of continuation and summary headings or specific styles that help to improve readability.

2.2.2 An Accelerated Branch-and-Bound Algorithm

The problem of automatic formatting of table layout is formulated by Wang [86] and by Wang and Wood [87] as a system of size constraints expressed as linear equalities or inequalities without objective functions. Only constraints with variables for column widths or variables for row heights are considered, but not both. They argue that finding a solution that satisfies user requirements such as size constraints or other aesthetic criteria, should be prioritized over finding the solution for an objective function. The size constraints are seen to be more relevant for the user because: a solution to an objective function does not always provide a suitable layout, users prioritize size constraints and an optimal solution is not always required
- A close to optimal solution is good enough. However, these arguments are personal observations rather than proved through an indepth study. They prove that the table formatting problem is NP-complete using a reduction to the *Subset Sum* problem and an accelerated branch-and-bound algorithm is suggested. They develop an integrated prototype that allows the user to edit the logical structure of a table, its topology and to easily present the table with various layout structures. In order to improve the running time of the table formatting algorithm, the prototype uses an inequality solver to find solutions for a limited number of linear inequalities for size constraints combined with a branch-and-bound strategy. They also point out in their conclusions that the problems with large tables need to be addressed. For example, when the table is paginated, determining where should a large table be broken in order to minimize the difficulty of reading.

### 2.2.3 Convex Hull and Weighted-sum Optimization

Anderson and Sobti [6] proved that the table layout optimization problem is NP-complete even for very simple instances, that is for tables with height and width configurations restricted to being 1 or 2. They use a reduction from the clique problem where the table is the incidence matrix of the graph and the bounds on the height and width of the table are used to test for a clique. They suggest a heuristic for finding the minimum height table for a given width $W$ by first computing the convex hull of the set of all feasible table sizes, then selecting the shortest table on the convex hull which has a width at most $W$. This heuristic is then compared with a second solution to the *weighted-sum-optimization* problem where the optimal table layout is found by computing the minimum vertex cover with a minimum cut in a flow graph. A linear programming method is also introduced. The linear program is analysed as a generalized network flow problem by introducing dual variables for the constraints on cells and table width. They provide an evaluation of these solutions for small table instances (8x8) that contain randomly generated cell configurations.
or randomly selected segments of real text. For tables that contain text data, the LP algorithm (based on the linear program solver *lp_solve* written by Michel Berkelaar) is reported to be the slowest method compared with all other proposed solutions. Anderson and Sobti also indicate that linear programming presents the advantage of allowing designer constraints to be added in a more general constraint optimization based layout engine.

### 2.2.4 An Iterative Method for Constraint Tabular Layout

Lutteroth and Weber [57] consider using a constraint tabular layout as the basis for the more generic case of two dimensional user interface layout. They start with the observation that the basic elements of user interfaces or structured documents are rectangular areas which must be aligned on common edges in order to represent the required structure and to give a clear overall appearance. As these edges can be either vertical or horizontal they define a flexible grid like model that preserves designer constraints in a dynamic layout space. Two type of constraints are used, ordinal constraints to specify the layout structure and linear constraints (absolute and relative constraints) that specify for example the exact location or size of the rectangular areas. An iterative method (Gauss-Seidel algorithm) is used to solve the resulting linear system of equations. While claiming that the proposed model, through its flexibility, solves some of the shortcomings of other layout engines (HTML or Java GridBag are mentioned), no evaluation of the performance of this algorithm is presented or a comparison with other similar methods or layout engines.

### 2.2.5 Area Approximation and Conic Programming

Hurst, Marriott and Moulder [46] presented two techniques for table layout optimization. The first proposed *area approximation* technique is based on the observation
that for cells that contain more than a few words of text the cell configurations can be approximated by a continuous non-linear constraint that the cell must be large enough to contain its text area when the text is laid out on a single line. They show that this continuous table layout problem is a convex programming problem. Thus, it can be solved in polynomial time as a continuous constraint optimization problem using conic programming which is based on interior point methods. They used the Mosek conic programming optimization kit developed by Mosek ApS [66].

The second method proposed by Hurst et al. consists in setting all the table columns to their narrowest widths and then reducing the table height by widening the table columns. This method relies on an $O(n)$ dynamic programming algorithm for finding the minimum cell width required for a given cell height, where $n$ is the number of words. They combine the two algorithms in a hybrid algorithm where a layout is found using the first method and then followed by the iterative column widening method. As an alternative hybrid method, the HTML layout algorithm is used to determine a layout followed by column widening. They evaluate the performance of these methods and find that the hybrid approach using the HTML layout algorithm is the most appropriate because iterative column widening can be slow for tables with large number of rows, while due to computational overheads conic programming is slow in the case of small tables. The running times reported are for tables with 3 to 6 columns. The largest table has 200 rows and 5 columns for which solutions were determined in a time range between 0.5s and just over 7s.

Following on the previous work, Hurst, Marriott and Albrecht [45] showed that solving the continuous non-linear constrained optimization problem as a conic programming problem requires interior point solvers which are complex to implement, relatively slow and not incremental. Therefore they presented a more efficient variable elimination method for solving the continuous approximation problem for a particular case of tables without spanning cells. To find an optimal solution which minimizes table height for a given page width they focus on finding a solution that
minimizes table’s perimeter which is then iteratively scaled until the height is minimized and the width is no larger than required. There is no report on the performance of this method besides the mention that it “takes only a few milliseconds on tables with thousands of elements”.

2.2.6 Exact solutions: Integer and Constraint Programming

Bilauca and Healy presented in 2010 [19] a new method for optimally solving the automatic table layout optimization problem. They suggested a mixed integer programming model and also a constraint programming model using OPL Optimization Programming Language for solving the problem for simple tables (tables without spanning cells). They provided for the first time a comparative evaluation of the performance of these methods on large instances for tables up to 1,600 cells. Solving this problem from a constraint optimization viewpoint has been suggested in the past but none of the previous methods could guarantee optimal solutions because they were based on heuristics.

In a follow up paper, Bilauca and Healy suggested in 2011 [21] an improved mixed integer programming model for tables with spanning cells. This model outperforms the previously proposed model and therefore they report on its performance for even larger instances (tables with 3,600 cells). They also consider for the first time the problem for tables with inner tables. They suggest a simplified method were the inner tables are recursively merged into the outer table.

2.2.7 A* and Improved Constraint Programming

In 2011 Gange et al. [40] proposed three methods for solving the table layout optimization problem. An A*-based approach uses a heuristic where a width for each column is selected in turn and the algorithm searches for the optimal table layout by selecting partial layouts with the lowest penalty - a lower bound on the table
height that extends the current partial layout. They also suggested a constraint programming model using the Zinc modelling language and a lazy clause generation approach which is a hybrid method that applies boolean satisfiability methods for search reduction. The advantage of the lazy clause generation is that it records the reasons for which a failure occurred during search (a nogood) and it will prevent future searches on similar sets of choices that can lead to the same failure. They evaluated the three methods in a very large test case that included over 2,000 tables and they show that the hybrid approach outperforms the A*-based method, the constraint programming and in some instances the mixed integer programming model proposed by Bilauca and Healy in 2010.

2.2.8 Summary of Previous Work and Conclusions

To summarise the previous work on table layout optimization an overall picture of the methods proposed so far is presented in Table 2.1. The table lists the proposed methods by each author, the engine or algorithm used by each method, the largest table size evaluated and whether the method considered tables with spanning cells and inner tables. Where we could not clearly determine the answer we used a dash in the respective cells. Following the literature review presented in this section we can draw the following conclusions

- Most of the solutions proposed so far are concerned with positioning constraints expressed as systems of linear equalities or inequalities. Unless specified (i.e. Lutteroth) we can only assume that the author is able to express positioning constraints for spanning cells or inner tables. We are unable to find any explicit study on the complexities that constraints for inner tables may introduce;

- Most of the proposed methods are evaluated only for relatively small size tables. If the methods are applied to larger instances additional constraint
Table 2.1: Solutions for automatic table layout

<table>
<thead>
<tr>
<th>Year</th>
<th>Authors</th>
<th>Methods</th>
<th>Engine</th>
<th>Table size</th>
<th>Spanning Cells</th>
<th>Inner Tables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1985</td>
<td>Beach</td>
<td>linear inequalities</td>
<td>Juno</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>1996</td>
<td>Wang</td>
<td>linear inequalities</td>
<td>inequality solver</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>with branch &amp; bound</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1999</td>
<td>Anderson and</td>
<td>linear constraints</td>
<td>ip_solve</td>
<td>8x8</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Sobi</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2005</td>
<td>Hurst et al.</td>
<td>continuous non-linear constraints</td>
<td>Mosek conic programming</td>
<td>200x5</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>2006</td>
<td>Lutteroth and</td>
<td>linear constraints</td>
<td>Gauss-Seidel algorithm</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
</tr>
<tr>
<td>Weber</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2010,</td>
<td>Bilauc and</td>
<td>integer &amp; constraint programming</td>
<td>OPL, CPLEX, CP</td>
<td>60x60</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>2011</td>
<td>Healy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2011</td>
<td>Gange et al.</td>
<td>A*-based, CP &amp; CP/SAT</td>
<td>Mercury, Zinc, Chuffed</td>
<td>40x40</td>
<td>Yes</td>
<td>No</td>
</tr>
</tbody>
</table>

reduction techniques are needed, as highlighted by Beach;

- Only Bilauc and Healy and Gange et al. report on the comparative evaluation of the running time for the suggested solutions.
2.3 Overview of Combinatorial Optimization Methods

In this section we present a review of the combinatorial optimization methods with a particular focus on the methods utilised in this thesis to solve table layout problems: mixed integer programming, constraint programming and local search. The aim of this section is to provide a background for the fundamental principles and advantages given by the techniques implemented in Chapter 3 and Chapter 4 with pointers to recent developments relevant to our work. Overviews of optimization problems, methods, languages, optimization engines, etc can be found in [82, 83, 39, 47, 88].

There is a multitude of problems that occur in practice in various domains where optimal solutions become very important in order to achieve cost savings, efficiencies or even for running a sustainable business. Practitioners in application areas such as scheduling, planning, resource allocation, hardware design, placement, layout, data mining, selection, decision support, etc. have studied these problems for many years. The computational complexity of these problems is directly related to the practical nature of how processes are performed: there are many ways of allocating resources to a project, many combinations of assigning workers to machines, many combinations on how items can be loaded in a truck, etc. Choosing an optimal solution from all these combinations is particularly difficult because very often a complex set of restrictions or constraints need to be respected. These type of problems are known as combinatorial optimization problems. The number of possible solutions can be extremely high even when the number of variables is relatively small. These combinatorial problems are known to be computationally difficult to solve, i.e. there no known algorithms that can solve these problems in a general efficient way. In the last decades, for some of these problems proofs have been documented that they are NP-hard and therefore difficult to solve even for moderate-size instances and no matter how much computational power and time is provided.
The need to find optimal solutions to these combinatorial problems has lead practitioners to develop specialized tools and systems which implement heuristics that are well known in practice to give satisfactory results. It is also known that tools developed in traditional programming languages require a considerable amount of effort and substantial costs for updating when the process changes: for example, when new, larger trucks, are purchased by a delivery company or smaller packages need to be delivered to more destinations or when a publishing house that expands overseas or into a new range of publications is required to implement layout rules for new page formats. Process changes make systems harder to maintain and, at the same time, their performance can be affected when new rules are added to the existing system. This lead to the development of new programming paradigms that separate the definition of rules and relationships (modelling) from the implementation of algorithms that solve these problems.

The definition of a model is usually expressed mathematically thorough algebraic notations that define mathematical relationships which correspond to real life geometrical, physical or other type of relationships [88]. Separating the definition of these relationships in a model independent of its implementation, allows the practitioner to focus on modelling the process instead of focusing on the technical development of specialized optimization algorithms. Also, expressing the process thorough well known and easy to understand algebraic notations allows engineers, economists, planners, designers, etc. to validate the relationships without the need to understand a specialised programming language. As models are independent of their data it allows for the same model to be applied to multiple data sets and thus reducing the development effort for many instances. However, it has been observed that different models perform better depending on the complexity of the data. An analysis of the performance of models in various conditions has always been valuable to practitioners as it allows them to select the best model for the problems that they face. This is one of the justifications for the performance evaluation results.
The purpose of most combinatorial problems is to optimize a function such as duration or cost, by finding an assignment of values for a set of variables subject to a set of limitations or constraints. The function to be optimized (minimized or maximized) is called the objective function. Mathematical programming is a common term utilised to describe in a highly declarative language the optimization of an objective function on a set of variables subject to constraints. Depending on the nature of the function and its constraints mathematical programming problems can be classified in linear programming, integer programming and non-linear programming problems. In linear programming a linear objective function needs to be minimized or maximized subject to a set of linear constraints. A particular case of linear programming problems are integer programming problems where a linear function must be optimized subject to linear constraints over integer variables. Another case is non-linear programming problems where the objective is to optimize a non-linear function subject to non-linear constraints. There are also hybrid problems such as mixed integer linear programming problems where only some variables are restricted to be integers. We discuss in more detail these type of problems in Section 2.3.1.

A programming paradigm for solving combinatorial optimization problems that emerged from logical programming is constraint programming. The main advantage of constraint programming is the support for high-level abstraction of constraints which makes the optimization models much easier to develop and maintain. Having its origins in constraint logic programming, variables have finite domains of possible values which are explored in order to find optimal solutions using predefined built-in search methods. As the size of these domains can be extremely large the search space is reduced by using constraints systems which eliminate in advance directions (combinations of values) that are deemed unsuitable, i.e. they violate requirements. The sets of constraints combined in a so-called constraint store connect the set
of possible combinations of values that variables can take. Constraint programming is discussed in more details in Section 2.3.2.

Layout problems (placement and routing in hardware design automation, factory floor planning, graphical interfaces) have been considered as ideal candidates for constraint optimization languages due to the extensive set of complex relationships that are defined between various elements and also due to the large number of solutions to choose from. In this area, a problem that has drawn attention from a variety of computer science application areas is the problem of optimal rectangle packing [41, 15, 13]. In its basic form the problem is to pack all rectangles with size $1 \times 1$ to $n \times n$ into an enclosing rectangle with minimum area.

Similarities exist between the table layout problem discussed in Chapter 3 and the optimal rectangle packing problem. In the table layout problem, rectangular cells need to be packed into a rectangle with a specified width while minimizing the height of the enclosing rectangle. Compared with the rectangular packing problem, in the table layout problem there are additional ordering and geometrical constraints that must be respected by the optimal solution: ordering constraints ensure that cells are grouped in rows and columns and their order cannot be changed in an optimal placement. The complexity of the table layout problem is also increased by the geometrical constraints which define the relationships between the cells of the same column and row: a wider cell affects the width of all the cells in a column and at the same time it affects their height which in turn can affect the height of all the rows in the table. Also the size of the cells can be determined only by selecting a width/height pair from a predefined set.

The rectangle packing problems have also been motivated by scheduling problems where the duration and the amount of resources required to complete a task represent the two dimensions of a rectangle. Table layout could also be considered in a scheduling context where, for example, the width of each cell represents the duration of a task and its height could be the cost required to perform this task.
This correlation respects even the anti-monotonicity criteria between the width and the height of a cell: increasing the width of a cell will reduce its height. In scheduling problems as the time provided to perform a task is shorter the costs increases if additional labour is necessary. To model the geometrical constraints for columns using scheduling concepts, constraints can be introduced to ensure that all tasks which correspond to cells in a column, start and end at the same time. Also, a definite sequence of tasks is provided to ensure a correct ordering that represents table columns. A possible issue occurs when modelling constraints for row heights which are given by the maximum cell height for all cells in a row. In a scheduling model the overall budget is the sum of all costs for the tasks executed over a period of time. Instead of using a daily budget another analogy should be used such as a daily penalty that depends on the overall maximum daily amount withdrawn to pay for labour. Nevertheless, it can be easily observed that the table layout problem can be considered as a scheduling problem: selecting combinations of task durations and resources that do not exceed a given duration while minimizing the overall maximum daily amounts.

The complexity of the rectangle packing problems is increased further when a temporal dimension is added. For example, for a cargo dock the time when containers are to be lifted in trucks or vessels is critical for the optimal ordering of containers. Also, for a delivery company the order in which parcels must be organized in a truck is important to maximize the amount of parcels transported in one route and also to minimize the unload time. A recent development that aims to solve these type of problems is the introduction by Beldiceanu et al. [14] of geost, a spatio-temporal global constraint kernel. The kernel uses a generic k-dimensional lexicographic sweep-point algorithm for filtering the attributes of an object. geost could potentially be applied for table layout problems because it facilitates modelling by using new geometrical constraints such as inclusion and non-overlapping. Because geost can decompose an object into boxes it becomes relevant for tables.
that have an uncommon shape (see Figure 1.1). Also, because geost considers the shapes of an object it facilitates modelling for objects that can rotate. In table layout there are many such instances where cells can be rotated. It could be easier for example to model the table layout problem using geost when tables have header cells over narrow columns as in Figure 1.4 where the header cells are rotated by 90 degrees in order to minimize the table’s height. geost has been integrated within the CHOCO library [49], as well as in SICStus Prolog [31].

As mentioned earlier, selecting the correct strategy when searching the solution space for optimal solutions can lead to significant performance improvements. In a recent paper Simonis and O’Sullivan [78] showed that, following an analysis of nine search strategies for the packing problem, using “off-the-shelf” constraint solvers and selecting an appropriate strategy, solutions were found in a shorter time than with the previous approaches [52] [53] [65] [64]. They showed that the implementation of branching strategies in SICStus Prolog [31] outperformed the methods where specialised constraints were developed for rectangle packing. This highlights again the importance of analysing multiple strategies for modelling combinatorial optimization problems.

### 2.3.1 Mixed Integer Programming

Linear programming (LP) and its generalisations, integer programming (IP), mixed integer programming (MIP), mixed-integer non-linear programming (MINLP), etc. are widely adopted as methods of modelling real life problems using an intuitive syntax close to their formal algebraic notation. This allows users to define problems and complex relationships that can be easily translated and solved by constraint solvers without the need of custom software development using a traditional programming language. There are many languages that support modelling for this classes of problems: AMPL [39], indexAMPL, OPL [83], GAMS [72], Mozart [84], etc.
The objective of a linear program is to determine for a set of positive real variables those values that optimize (minimize or maximize) an objective function subject to a set of linear constraints. A linear program is of the form:

\[
\text{minimize } \sum_{j=1}^{n} c_j \cdot x_j \\
\text{subject to } \\
\sum_{j=1}^{n} a_{i,j} \cdot x_j \geq b_i (1 \leq i \leq m) \\
x_j \geq 0 (1 \leq j \leq n)
\]

A particular case of linear programming is integer linear programming where variables are required to take integer values. These problems occur in real life when variables represent entities that cannot be subdivided, i.e. for example when deciding how many warehouses a company requires, how many car units need to be manufactured, how many people are assigned to machines, etc. When only some of the variables are required to be integers the problems are called mixed integer programming (MIP) problems.

While linear programs can be solved in polynomial time, in the case of integer programs, the constraint that variables can only take integer values makes integer programming problems hard to solve (they are NP-hard). They normally require more processing power and memory and therefore only smaller instances can be solved compared with linear programs.

MIPs have been the subject of extensive research and various methods have been developed for solving different classes of integer programs: logical conditions programs to extend LP programs, combinatorial programs to solve sequencing and allocation problems, etc. There are well known combinatorial problems that have been modelled using MIP. The job-shop scheduling problem, the travelling sales-
man problem, problems that arise in project selection and investment decisions, the aircrew scheduling problem, the depot location problem are just a few examples.

The most common methods for solving MIP problems are Branch-And-Bound or Branch-And-Cut algorithms. These implicit enumeration techniques rely on identifying and removing solutions that are known not to be optimal (they lead to an objective value worse than the best value determined so far) or solutions that are known to invalidate the constraints. Other algorithms are also used for solving integer programs: cutting-plane techniques, group theoretic techniques and also hybrid approaches.

Many commercial LP and MIP solvers are currently available: IBM-CPLEX [34] which before 2008 was developed by ILOG [73], Mosek [66], SCIP [2] and also open source solvers such as CBC maintained by COIN-OR [38], lp_solve [18] a linear (integer) programming solver based on the revised simplex method and the branch-and-bound method, GNU Linear Programming Kit – glpk [58], etc.

A detailed discussion about MIP fundamentals can be found in the work of Nemhauser and Wolsey [68] while a review of recent work has been presented by Lodi [55] in 2010. A review of practical solutions for non-linear integer programs is presented by D’Ambrosio and Lodi [37].

2.3.2 Constraint Programming

Constraint programming evolved from constraint logic programming and it has been influenced by techniques such as logic programming, mathematical programming, constraint solving, consistency techniques and mathematics (boolean algebra, graph theory and finite mathematics). Constraint systems were initially developed in existing programming languages such as PROLOG II [33], PROLOG III [32], CHIP (Constraint Handling in Prolog) [36] just to name a few but since the mid 1990’s constraint programming is available in traditional programming languages such as C, C++ or Java.
Layout systems based on constraint optimization have been developed at the same time and as a direct application for constraint languages. One of the earliest constraint based layout systems, Sketchpad, has been introduced in 1964 by Ivan Sutherland [80]. Sketchpad introduced a new, direct way of interacting with the display using a pen for editing parts of the drawing. In Sketchpad each drawing object can be positioned based on a set of constraints and thus complex relationships with other objects can easily be defined. Sketchpad is considered to be the first graphical user interface and also the first computer animation tool.

Following Sutherland’s work, constraint systems for graphical layout have been implemented by Alan Borning in Thinglab [22, 23] and also by Guy Steele and Gerald Sussman [79]. They introduced the concept of a constraint language which is optimized to describe structural relationships between entities as opposed to traditional computer languages which are designed to express algorithms. They defined a “language of hierarchical constraint networks” and the concept of constraint propagation which is a method of deriving useful consequences from a set of constraints. Propagation plays an essential role in the modern constraint programming languages because it allows one to simplify and further develop the constraint system by addition of new constraints and the deletion of old ones.

The development of concurrent constraint programming by Saraswat [74] in the 1980’s allows the constraints solvers to concurrently explore alternate solutions by implementing processes (sub-programs) that communicate through adding constraints to or querying the constraint store in order to determine which portions of the search space should be explored. This represents a major step towards the modern constraint programming languages used today. Initially constraint logic programming languages offered a limited set of basic primitive constraints or control structures that could be used to express the constraints of the original problems. This drew the attention of the research community and in the 1990’s new global constraints [4, 15] were developed to express relationships such as among, diffn and
Detailed reviews of the developments in constraint languages up to mid 1990’s is presented in Jaffar and Maher [47], Simonis [76] [77] and Marriott and Stuckey [61].

Constraint optimization systems for graphical layout continuously evolved [29] with the development of improved constraint solving algorithms by Borning [24, 26] and Badros [9]. In the late 1990’s, with the development and wide adoption of the Internet, constraint systems for optimal display of web content have been suggested [8, 27, 28]. During the 1990’s constraint programming languages such as OPL [83], Oz and Mozart [84] have been developed and constraint solvers have emerged such as OTI [25] and ILOG Solver [73] just to name a few.

2.3.3 Local Search

An alternative and complementary method to constraint programming is local search. In constraint programming the solution space is explored globally by dividing the problem in smaller sub-problems until they can be solved directly. Local search starts with an initial solution and iteratively moves to a neighbour (local) configuration based on some predefined relationship until an acceptable solution is found or a timelimit is reached. Local search is a widely-used optimization method in practical applications because it often provides solutions with a very good ratio between quality and running time. Local search is especially relevant and it provides qualitative solutions even for large instances where other methods fail.

While linear programming and constraint programming are well supported, only recently tools that provide support for local search have started to emerge. In 1997 Andreata et al. [7] suggested a framework for evaluating local search heuristics. In 1999 Schaerf et al. [75] proposed a C++ based framework called Local++ for the development and evaluation of local search algorithms. Michel and Van Hentenryck suggested in 2000 a modelling language, Localizer [63], which is in line with traditional mathematical modelling languages, for implementing local search algorithms.
using a notation similar to the natural description of these algorithms. They followed up with a framework [62, 42] where local search and constraint programming are combined, using constraints to describe and control the search process. They also proposed a programming language, COMET, that supports both modelling and high level search abstractions.

Recently, \textit{LocalSolver 1.0} was introduced by Benoist et al. [17, 16] as a solver for a new \textit{Local Search Programming} paradigm. Although in its initial version LocalSolver had a basic declarative language, in its latest version, LocalSolver 2.0 released in March 2012, the modelling language is comparative with any other more established modelling languages such as AMPL, GAMS, OPL, etc. LocalSolver 2.0 is used in this thesis to evaluate the performance of local search techniques for table layout optimization.

An overview of the fundamentals in local search methods, the algorithms used and their complexity, a study of local search application areas, etc. are covered in a book by Aarts and Lenstra [1] which was initially published in 1997.
Chapter 3

Automated Table Layout Optimization

In this chapter we present a solution to the table layout optimization problem using an exact combinatorial optimization modelling method. We restrict our attention to simple tables, that is tables that do not contain spanning cells or inner tables (these tables are discussed in Chapter 4). We give a formal definition of the problem in Section 3.1 and its integer programming definition in Section 3.2. To evaluate this method we first show in Section 3.3 and Section 3.4 how to model the problem as a mixed integer programming (MIP) problem and then we show a constraint programming (CP) model in Section 3.5. We present in Section 3.7 experimental results for tables that contain large amounts of text and also for tables with up to 1,600 cells and 9,000 cell configurations.

3.1 The Table Layout Optimization Problem

The table layout optimization problem is to find for a table $T$ the layout $L$ with minimum $\text{height}(L)$ such that $\text{width}(L) \leq W$ where $W$ is a given page width.\footnote{A similar definition is given by Anderson and Sobti \cite{Anderson1987}}
Figure 3.1: The text in this cell can be arranged in 3 configurations

Table $\mathcal{T}$ is a matrix with $m$ rows and $n$ columns that contains $m \times n$ cells. The content of a cell $i,j$ can have many configurations, i.e. if the cell contains text then it can be split on consecutive lines and therefore it generates various bounding boxes or configurations. For example, the cell presented in Figure 3.1 has three configurations.

Each cell $i,j$ with $1 \leq i \leq m$ and $1 \leq j \leq n$ has a configuration set $C_{i,j}$ of the form $\{(w_{i,j}^k, h_{i,j}^k) \mid 1 \leq k \leq |C_{i,j}|\}$. Finding a layout $\mathcal{L}$ for table $\mathcal{T}$ is selecting for each cell $i,j$ the configuration $c_l = (w_{i,j}^l, h_{i,j}^l)$ with $c_l \in C_{i,j}$ and $1 \leq l \leq |C_{i,j}|$ to be used when displaying the table. Therefore, the cell width is $w_{i,j} = w_{i,j}^l$ and the cell height is $h_{i,j} = h_{i,j}^l$. The column width $w_j$ is the maximum width for that column’s cells and the row height $h_i$ is the maximum height for that row’s cells.

$$w_j = \max_i w_{i,j}$$

$$h_i = \max_j h_{i,j}$$

The layout width $\text{width}(\mathcal{L})$ and the layout height $\text{height}(\mathcal{L})$ are defined as:

$$\text{width}(\mathcal{L}) = \sum_{j=1}^{n} w_j$$

$$\text{height}(\mathcal{L}) = \sum_{i=1}^{m} h_i$$

Our solution for the table layout optimization problem uses a combinatorial optimization modelling language. We give examples for OPL, the Optimization
Programming Language [83]. OPL is a modelling language designed for solving combinatorial optimization problems. It supports constraint programming including search specification, logical and higher order constraints and support for scheduling and resource allocation applications. Compared with other programming languages that support mathematical programming and constraint programming OPL has a major strength given by its syntax. As OPL shares many syntax features of traditional mathematical programming languages such as AMPL [39] or GAMS [72], problems can be formulated in a language similar to their algebraic notation. For our solution we use IBM’s industrial implementation of OPL as part of the IBM ILOG CPLEX Optimization Studio [34]. The listings below show only the relevant parts of the model. The full code of the optimization models is available in Appendix A. 

As we have stated before, in this section we assume that there are no cells that span more than one column or row. We first show how to model the table layout problem for OPL’s mixed integer programming (CPLEX) engine, then a constraint programming (CP) model.

### 3.2 Integer Programming Definition

We define the table layout problem for simple tables as a basic mixed-integer optimization problem. We introduce binary variables $x_{i,j}^k$ associated with each cell configuration and we determine their value such that:

1. the table height is minimized,

2. the table width is less than the given page width $W$, and

3. for each cell $i,j$ only one configuration is selected.
The basic integer programming definition for the table layout problem is:

\[
\begin{align*}
\text{minimize} & \quad \sum_{i=1}^{m} \max_{j,k} h_{i,j}^k \cdot x_{i,j}^k \\
\text{subject to} & \quad \sum_{j=1}^{n} \max_{i,k} w_{i,j}^k \cdot x_{i,j}^k \leq W \\
& \quad \sum_{k} x_{i,j}^k = 1, \quad \forall i, j
\end{align*}
\]

where \( x_{i,j}^k \in \{0, 1\} \), \( \forall i, j, k \)

In the following section we present how this problem is modelled using OPL. Then, in Section 3.4 we introduce an improved model based on the definition presented above.

### 3.3 Basic MIP Model for Table Layout - BMIP

The purpose of an OPL optimization model is to find values for the decision variables declared with the keyword \texttt{dvar} such that all constraints are satisfied or, in optimization problems, to find values for the variables that satisfy all constraints and optimize a specific objective function [73].

To implement the integer programming definition for the automatic table layout problem as defined in the previous section we start with a basic model presented in Listing 3.2. In order to select from the set of cell configurations \texttt{configs} those configurations that minimize table height for a given page width \texttt{pageW}, an array of binary decision variables \texttt{cellSel} is used. The data types defined for this model are presented in Listing 3.1. Each cell configuration \texttt{CellConf} contains the cell indices \( i \) and \( j \) and a configuration of type \texttt{Conf} which contains the width \( w \) and the height
Listing 3.1: BMIP Data types and data initialization

```
1 tuple Conf{
2   int w;
3   int h;
4 }
5 // cell configuration type
6 tuple CellConf{
7   int i;
8   int j;
9   Conf c;
10 }
11 // cell type
12 tuple Cell{
13   int i;
14   int j;
15 }
16 // initialize data
17 int pageW = . . . ;
18 {CellConf} configs = . . . ;
19 // determine the set of cells
20 {Cell} cells = {<i,j> | <i,j,k> in configs};
21 // determine the set of rows
22 {int} rows = {i | <i,j> in cells};
23 // the set of columns
24 {int} cols = {j | <i,j> in cells};
```

The set of cells, rows and columns cols are determined from the set of cell configurations configs.

This model uses a particular feature of modelling languages: the ability to define an array cellSel[configs] indexed by a set of cell tuples configs. The role of the binary variables can be seen in line 5 where for each cell i,j the width is computed as the sum of products between the binary variables cellSel[<i,j,k>] and the cell configuration width k.w. Basically, in order to find the solution to the objective function minimize tableH, for each cell configuration <i,j,k> a value of 1 is stored in cellSel[<i,j,k>] if the configuration is selected, or 0 otherwise. We also use decision expressions declared with dexpr in order to express decision variables in a more compact way.

In this model we introduced two constraints: first, constraint ct1 ensures that the table width tableW must be less than pageW, the available page width. In a second constraint ct2, we make sure that for each cell only one configuration is
Listing 3.2: BMIP - A basic MIP model

// decision variable
dvar int cellSel[configs] in 0..1;

// compute table width by first computing cell width
dexpr int cellW[i,j] in cells = sum(<i,j,k> in configs) cellSel[i,j,k] * k.w;

// then compute column width
dexpr int colW[j in columns] = max(i in rows) cellW[i,j];

// and finally table width
dexpr int tableW = sum(j in columns) colW[j];

// compute table height by first computing cell height
dexpr int cellH[i,j] in cells = sum(<i,j,k> in configs) cellSel[i,j,k] * k.h;

// then compute row height
dexpr int rowH[i in rows] = max(j in columns) cellH[i,j];

// and finally compute table height
dexpr int tableH = sum(i in rows) rowH[i];

// The objective function is to minimize table height
minimize tableH;

// subject to two constraints
constraints
{
ct1:  // table width must be less than page width
      tableW <= pageW;

ct2:  // only one cell configuration can be selected
      forall(i in rows, j in columns)
            sum(<i,j,k> in configs) cellSel[i,j,k] == 1;
}

In our evaluation we found that while providing fast results for small tables with less than 10 rows/columns and a limited number of cell configurations, this model is inefficient for larger tables with many cell configurations. However, compared with other models presented in this chapter, it provides fastest results for tables with a small number of cells and a large number of cell configurations, i.e. tables with large text content. The models presented in the next sections will build upon this foundation.
3.4 A MIP Model - MIP

We now present an improved model that uses the observation that for each column, its width is one of the values selected from the union of cell configuration widths for that column. Another observation is that each cell has a minimum width which, in turn, imposes a minimum width on the column. For each column we can compute the minimum column width $\text{minW}[j]$ which is the maximum of minimum cell widths for a column $j$. We then remove from the set of width values any values less than this maximum thus reducing the solution space. This reduces the number of binary variables and the number of constraints in the model which directly affects the performance of the optimization engine. We modify the initial BMIP model with the code presented in Listing 3.3 where in $\text{minW}[j]$ we store the minimum width for each column.

The important difference of this model compared with the BMIP model is how the width of each column is computed. We build the set of column widths $\text{colWset}$, from all the cell configurations whose widths are greater than minimum column width values $\text{minW}$ (line 5). Each item in this set contains a tuple $<j,w>$ which stores the column $j$ and the width $w$. A new array of binary decision variables $\text{colSel}[\text{colWset}]$ is defined on line 7 which will indicate if a configuration from $\text{colWset}$ is selected or not. Therefore, the column width $\text{colW}[j]$ is computed as the product between the binary variable $\text{colSel}[<j,w>]$ and $w$ (line 14).

Two additional constraints are inserted after ct1 in Listing 3.3. The first constraint ct2 ensures that only one width is selected for each column and ct3 ensures that the selected cell configuration has a width equal to or smaller than the selected column width. As in the previous model, constraint ct4 ensures that only one cell configuration is selected.

As we show in Section 3.7, this model returns the fastest results for simple tables with up to 1,600 cells (40x40).
Listing 3.3: MIP model for the table layout optimization problem

1 int minW[j in columns] = max(i in rows)
2 min(<i,j,k> in configs) k.w;
3 {ColW} colWset={<j, k.w> | <i,j,k> in configs : k.w >= minW[j]};
4 dvar int colSel[colWset] in 0..1;
5 dvar int cellSel[configs] in 0..1;
6 // cell width is computed similarly to the BMIP model
7 dexpr int cellW[i,j] in cells = sum(<i,j,k> in configs) cellSel[i,j,k] * k.w;
8 // column width is selected from the set of column widths
9 dexpr int colW[j in columns] = sum(<j,w> in colWset) colSel[j,w] * w;
10 // table width
11 dexpr int tableW = sum(j in columns) colW[j];
12 dexpr int tableH = // ... same as the BMIP model
13 constraints
14 { ct1: // table width must be less than page width
15 tableW <= pageW;
16 ct2: // each column can have only 1 selected configuration
17 forall(j in columns)
18 sum(<j,k> in colWset) colSel[j,k] == 1;
19 ct3: // the cell width must be less or equal to its column width
20 forall(j in columns, i in rows)
21 cellW[i,j] <= colW[j];
22 ct4: // only one cell configuration can be selected
23 forall(i in rows, j in columns)
24 sum(<i,j,k> in configs) cellSel[i,j,k] == 1;
25 }
3.5 Constraint Programming Model - CP

The main advantage of constraint programming comes from the support for two main activities of combinatorial optimization algorithms: constraint reasoning and search strategy specification. In OPL constraint reasoning is dealt with at a high level by simply specifying a list of constraints for the decision variables. These constraints are then used in bounding techniques for domain reduction, e.g., the set of possible values for the decision variables. In constraint programming languages the user can also specify search strategies. This could lead to improved performance as the search strategy can embed in the model a heuristic specially selected for the problem.

As the basis for constraint programming is domain reduction, models that use binary variables can be inefficient. The MIP models presented in the previous sections proved to be inefficient when the constraint programming engine was selected even when search strategies were included in the model. Therefore we developed a different model where binary decision variables, \( \text{colSel} \) and \( \text{cellSel} \) in the MIP model, are replaced by integer decision variables \( \text{colW} \) and \( \text{rowH} \) as shown in Listing 3.4. The values of these variables are to be found in the domains given by the set of cell configurations.

Using the same strategy as in the MIP model where we first find the column widths that give a table layout that fits in the page, in this model we also define an array of decision variables \( \text{colW}[j] \) for column widths but we also define an array of decision variables \( \text{rowH}[i] \) for row heights. The idea behind this model is to find those column widths and row heights from the range of possible width and height values such that the height of the table layout is minimized, the table layout width is less than the page width and for each cell \( i,j \) there is at least one configuration \( k \) with a width less than or equal to the selected column width and the height less than or equal to the selected row height: \( w_{ij}^k \leq colW[j] \) and \( h_{ij}^k \leq rowH[i] \).
Listing 3.4: CP - Constraint Programming model

```cpp
using CP;  // use Constraint Programming engine

int minW[j in columns] = max(i in rows) min(<i,j,k> in configs) k.w;
int minH[i in rows] = max(j in columns) min(<i,j,k> in configs) k.h;

{Dim} colWset = {<j, k.w> | <i,j,k> in configs: k.w >= minW[j]};
{Dim} rowHset = {<i, k.h> | <i,j,k> in configs: k.h >= minH[i]};

dvar int colW[j in columns] in min(<j,w> in colWset) w .. max(<j,w> in colWset) w;
dvar int rowH[i in rows] in min(<i,h> in rowHset) h .. max(<i,h> in rowHset) h;

dexpr int tableW = sum(j in columns) colW[j];
dexpr int tableH = sum(i in rows) rowH[i];

minimize tableH;

{ constraints
  ct1: // table width must be less than page width
    tableW <= pageW;
  ct2: // only one column width can be selected
    forall(j in columns)
    sum(<j,w> in colWset) (colW[j]==w) == 1;
  ct3: // only one row height can be selected
    forall(i in rows)
    sum(<i,h> in rowHset) (rowH[i]==h) == 1;
  ct4: // at least one cell configuration will be selected
    forall(i in rows, j in columns)
    sum(<i,j,k> in configs)
    (k.w <= colW[j] && k.h <= rowH[i]) == 1;
}
```
To ensure that for each column only one width value is selected we use higher order constraints. In constraint \(\text{ct2} \) the expression

\[
\text{sum}(<j,w> \text{ in } \text{colWset}) (\text{colW}[j]==w) == 1;
\]

will ensure that for a column \(j\) the selected column width \(\text{colW}[j]\) is found only once in \(\text{colWset}\), the set of column widths. In this statement, the summation is over the expressions \(\text{colW}[j] == w\). The key point here is that each expression evaluates to 1 if the constraint is satisfied and 0 otherwise. Therefore the sum in the above expression counts the number of appearances of \(\text{colW}[j]\) in \(\text{colWset}\). In a similar manner, in constraint \(\text{ct4}\) we ensure that at least one cell configuration can fit in the selected column width and row height. The expression

\[
\text{sum}(<i,j,k> \text{ in } \text{configs}) (k.w <= \text{colW}[j] && k.h <= \text{rowH}[i])
\]

counts for a cell \(i,j\), how many configurations \(<i,j,k>\) respect the constraint.

Finally, to improve the performance of our model, we can specify a search strategy. In OPL 5.5 there is no search language. In fact, ILOG stopped providing a constraint programming engine in its OPL Development Studio 4.0 which was later re-introduced in OPL 5.2. This is indeed a strange move of the software manufacturer when the support for “sophisticated search specifications” in the constraint programming engine has been highlighted as one of the key features of OPL. Pascal van Hentenryck, the developer of OPL, calls the ability to specify search procedures “fundamental” for hard combinatorial optimization problems \[83\]. However, in the current version of OPL and after the acquisition of ILOG by IBM in 2008, the default search strategy can only be tuned by specifying search phases \[73\].

Our search strategy presented in Listing 3.5 is to first select values for column widths, then for row heights, using the maximal regret heuristic. During search, when selecting the index of the next variable to be fixed, we give priority to the variable with the largest regret. The regret of a column is the difference between its first and second choice of possible width value, starting with the largest width. The same maximal regret heuristic is also used when deciding the row height con-
figuration with the only difference that the assignment will start with the minimum height values.

3.6 Adding Designer Constraints

The potential of constraint optimization for graphical user interface or for document production has been highlighted for many years. Since 1963 when Ivan Sutherland [80] developed *Sketchpad* numerous systems and prototypes have been developed which use constraint solvers to optimize both layout topology and appearance.

Developing an optimization language model for the table layout problem presents the advantage that designer (user) imposed constraints can be easily added to the model. This has been suggested in the past by Beach [12], Wang [86], Anderson and Sobti [6] just to name a few.

For example, adding the designer constraint that all columns must be of equal width can be implemented by simply inserting a new constraint *ct6* in any of the models above, as presented in Listing 3.6. On line 2, the expression

\[
\text{forall(ordered } j_1, j_2 \text{ in columns) is equivalent to}
\]

\[
\text{forall(} j_1, j_2 \text{ in } 1..n : \ j_1 < j_2)
\]

In a similar manner, other designer constraints can be added to the model and there are numerous such examples.

Listing 3.5: Search strategy in the CP model.

```java
execute {
    var f = cp.factory
    var phase1 = f.searchPhase(colW, f.selectLargest(f.regretOnMax()), f.selectLargest(f.value()));
    var phase2 = f.searchPhase(rowH, f.selectLargest(f.regretOnMin()), f.selectSmallest(f.value()));
    cp.setSearchPhases(phase1, phase2);
}
```
Listing 3.6: An “equal-width column” constraint.

```
ct6:
  forall (ordered j1, j2 in columns)
  colW[j1] == colW[j2];
```

It is important to note that when such constraints are added to a model the complexity of the problem is increased. Therefore finding optimal solutions to the table layout problem can become harder with a substantial increase in the running time.

### 3.7 Experimental Results

In this section we compare the results obtained for both the MIP and CP models for tables up to 40 rows and 40 columns. For our experiments we used IBM ILOG OPL IDE 6.3 with CPLEX engine 12.1 and CP Optimizer 2.3 running on a Windows Vista PC with a 2GHz dual core processor and 2Gb of RAM. Most of our models were developed in 2008 and 2009 on OPL 5.5 and then migrated to OPL 6.3, the latest version available since 2010. By using IBM’s ILOG Concert Technology we also integrated these OPL models in a Table Drawing Tool prototype presented in Chapter 5. This allowed us to perform visual inspection on the computed table layouts. The results were promising from both computational time stand point and also aesthetically.

As with any other experiments, the choice of test data is critical. In our experiments we used real-life tables from the Naval Research Laboratory Report “Software Requirements for the A-7E Aircraft” [5] and also tables with automatically generated text content. This was required in order to avoid data entry delays and also to have a wider domain of test data that is challenging for optimization engines. Gange et al. [40] also confirmed that due to the random nature of the text selected in each cell, these tests have proven to be the most challenging for the optimization
engines.

Compared with other research areas (i.e graph drawing) where there is an established set of benchmark data used for evaluating various methods and algorithms, there is no established set of benchmark tables that can be used to evaluate the output of optimization methods. Gange et al. [40] gathered over 50,000 real life tables from the Internet. They found that only 4% (2,063 tables) were challenging enough because even if the tables were large (the largest table had 26 columns and 589 rows), most of the cells had only one configuration and therefore finding the optimal solution required a short computational time.

It is noteworthy that only the cell’s text has been generated and not simply cell configurations. The text for each cell consists of a sequence of a random number of words taken from a real document which is partially presented in Appendix E. In order to achieve a real data domain for cell configurations we used graphical functions to determine cell dimensions. Since our visualisation tool uses PDF for output it was possible to compute exact text dimensions taking into account parameters such as font size, font style, font family, spacing between characters, words and lines, etc. Thus the cells in these tables contain formatted text with different font size, style, spacing, etc.

We tested the proposed models on two types of tables: tables with a small number of cells but a large number of cell configurations, i.e. tables that contain large amounts of text (and thus many configurations), and also increasingly large tables with cells that contain text with up to 6 words.

The reported running time does not include the time required to compute cell configurations. Paragraphing, determining cell configurations or how to split the text on consecutive lines, is an essential aspect of the table layout optimization problem as it impacts on the quality and computational time of these solutions. It has been studied as a problem by itself starting with Knuth [50][51] and we consider it to be beyond the scope this thesis.
In all the results, BMIP refers to the response time for the basic MIP model, MIP is the response time for the enhanced MIP model and CP represents the response time for the constraint programming model. \(minW\) and \(maxW\) represent the minimum and maximum table width, computed from the cell-width configurations. That is, the table layout where each word in the text of a cell is displayed on a new line is used to compute \(minW\) while \(maxW\) is computed based on a table layout where the text in all cells takes its maximum width without any additional line breaks.

In Figure 3.2 we compare the running time for the BMIP, MIP and CP models on a typical table that contains text. The table has only 3 rows and 3 columns but each cell contains up to 200 words. On this instance we ran the three optimization models with a variety of table width limitations. It can be observed that the best running time was given by the basic MIP model. The running time of the CP model did not change significantly when the tests were repeated multiple times.

Conducting the same procedure on a table with 10 rows and 10 columns with up to 6 words per cell, the running time increases as the page width is decreased,
as shown in Figure 3.3. It can also be observed that when the page width is equal to the minimum table width $minW$ (790 points), the response time was around 1s for all models.

During experiments, we found cases when most computational time was required only to determine a slightly smaller but optimal solution (minimum table height) than a previously discovered feasible solution. There were also cases where the MIP model found a feasible solution in a relatively short period of time compared with the time required to decide that it was the optimal solution. For example, our tests unearthed a case where the best feasible solution of 680 points height for a table with 1600 cells (40x40) and a page width of 7000 points was determined in 43s, and deciding that this was the optimal solution required a further 1 hour and 25 minutes. The same effect has been observed in the case of a 30x30 table with up to 6 words and a page width of 3500 where the solution of 770 points was determined in 30s and the decision that it was the optimal solution was made after 1 hour and

\footnote{We ran our experiments to exhaustion insofar as it was possible. While some of the running times reported here may be unacceptably large they provide an important context, we believe.}
Table 3.1: MIP model run time.

<table>
<thead>
<tr>
<th>Table</th>
<th>minW</th>
<th>+25%</th>
<th>+50%</th>
<th>+75%</th>
<th>maxW</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td>620</td>
<td>900</td>
<td>1,200</td>
<td>1,500</td>
<td>1,814</td>
</tr>
<tr>
<td>height</td>
<td>460</td>
<td>290</td>
<td>180</td>
<td>130</td>
<td>100</td>
</tr>
<tr>
<td>time</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
</tr>
<tr>
<td>summary</td>
<td>m=620x560; M=1814x100; cnf=343; wSet=888</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| width  | 1,320| 2,000| 2,500| 3,000| 4,284|
| height | 1,040| 600  | 440  | 380  | 200  |
| time   | 00:01| 00:21| 00:06| 00:27| 00:01|
| summary| m=1320x1170; M=4284x200; cnf=1382; wSet=592 |

| width  | 2,094| 3,500| 4,500| 5,500| 6,709|
| height | 1,560| 770  | 570  | 440  | 300  |
| time   | 00:07| 1:39:49| 00:19| 04:15| 00:01|
| summary| m=2094x1780; M=6709x300; cnf=3082; wSet=1145 |

| width  | 2,879| 5,000| 6,000| 7,000| 9,189|
| height | 2,140| 1,280| 780  | 680  | 400  |
| time   | 00:17| 01:48| 01:57| 1:26:10| 00:01|
| summary| m=2879x2400; M=9189x400; cnf=5546; wSet=1908 |

39 minutes.

Generally, the time required to find the optimal solution increased as the page width decreased, which was not an unexpected result. However, exceptions to this rule prevailed: in a table with 900 cells (30x30), it took 19s to find the minimum-height layout that could fit in a page width of 4500 points. As we increased the page width to 5500 points, we found that to determine the minimum solution the running time increased to 4 minutes and 15s as seen in the highlighted entries in Table 3.1. Also, in a 20x20 table, if the page width was set to 3000 points it took more than four times longer (27s) to find a solution than when the page width was smaller at 2500 points (6s).

In Table 3.1 we present the running time for the MIP model for all the table configurations that we experimented with. We do not show the running times for BMIP or CP models as these proved not to be practical for larger tables. The runtimes are in the format h:mm:ss. Where the running time only consists of minutes, the hours are not represented. For each table the page width of an instance is computed in increments of squeeze factor of approximately 25% that is the difference
between the minimum to the maximum table width, as computed by the minimum and maximum cell-width sums, $minW$ and $maxW$. This is displayed in the first line. The instance’s solution (table height) is shown on the second line and the running time is given on the third line. In the summary line of each instance we show,

- $m$ minimum table width configuration ($\text{width} \times \text{height}$ in points);
- $M$ maximum table width configuration ($\text{width} \times \text{height}$ in points)
- $cnf$ total number of cell configurations;
- $wSet$ the number of possible column widths;

Another observation is that the performance of the models presented in this section appear to be unaffected by the number of cell configurations in each cell but rather by the number of rows and columns and also by the number of possible columns widths $wSet$. For example, in Figure 3.4 we can see that the running time to optimize a 3x3 table was still under 1.5 seconds even when the number of words in each cell increased starting at 200 and up to 600. Also, in Figure 3.5 we can see that the computational time increased as the increased number of cell configurations resulted in an increased number of possible columns widths $wSet$.

This hypothesis is confirmed by a set of experimental results with tables that have a fixed number of cell configurations and an increasing number of rows and columns.

To maintain the number of cell configurations in the entire table to 1,200 we run tests for tables with 2x3x200, 3x4x100, 10x10x12 and 20x20x3 (the numbers are, respectively, rows, columns and words). It can be observed from Table 3.2 that for a 10x10x12 table, the running time increases to more than 1 minute even if the total number of configurations is equal with the number of configurations in tables with less rows and columns (2x3x200 and 4x3x10) where the running time was under 2 seconds.
Figure 3.4: MIP model running time for a 3x3 table; word-count per cell varies from 200 to 600.

Figure 3.5: MIP model running time for 10x10 tables; word count per cell varies from 6 to 100.
Table 3.2: MIP model run time for fixed number of cell configurations.

<table>
<thead>
<tr>
<th>Table</th>
<th>minW</th>
<th>+25%</th>
<th>+50%</th>
<th>+75%</th>
<th>maxW</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>width</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2x3x200</td>
<td>245</td>
<td>1,000</td>
<td>2,000</td>
<td>3,000</td>
<td>3,980</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>2,350</td>
<td>400</td>
<td>190</td>
<td>130</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
<td>00:01</td>
</tr>
<tr>
<td></td>
<td>summary</td>
<td>m=245x3980; M=17742x20; cnf=1,200; wSet=1,160</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4x3x100</td>
<td>224</td>
<td>2,300</td>
<td>4,600</td>
<td>6,900</td>
<td>9,166</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>3,030</td>
<td>170</td>
<td>90</td>
<td>80</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>00:01</td>
<td>00:02</td>
<td>00:02</td>
<td>00:01</td>
</tr>
<tr>
<td></td>
<td>summary</td>
<td>m=224x4000; M=9166x40; cnf=1,200; wSet=1,113</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10x10x12</td>
<td>701</td>
<td>1,000</td>
<td>2,000</td>
<td>3,000</td>
<td>4,272</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>1,110</td>
<td>730</td>
<td>280</td>
<td>200</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>00:01</td>
<td>01:20</td>
<td>00:12</td>
<td>00:44</td>
</tr>
<tr>
<td></td>
<td>summary</td>
<td>m=701x1200; M=4272x100; cnf=1,200; wSet=844</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20x20x3</td>
<td>1,298</td>
<td>1,500</td>
<td>1,900</td>
<td>2,300</td>
<td>2,754</td>
</tr>
<tr>
<td></td>
<td>height</td>
<td>590</td>
<td>530</td>
<td>420</td>
<td>310</td>
</tr>
<tr>
<td></td>
<td>time</td>
<td>00:01</td>
<td>00:11</td>
<td>00:25</td>
<td>00:28</td>
</tr>
<tr>
<td></td>
<td>summary</td>
<td>m=701x1200; M=4272x100; cnf=1,200; wSet=356</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In Figure 3.6 it can also be observed that for a 20x20x3 table the running time was still higher than the 2x3x200 and 3x4x100 but lower than the 10x10x12 because the input data resulted in a lower number of possible column widths wSet of 356 compared with the 10x10x12 model where wSet is 844 which is more than double.

3.8 Conclusions

In this chapter we described several models of the table layout optimization problem using OPL. Based on the experimental results we can draw the following conclusions:

1. OPL provides viable solutions for applications where the table layout with the minimum height for a given page width must be found. The MIP model which uses the CPLEX optimization engine generally provides faster results than the constraint programming model. However, this does not mean that the CP should be ignored because, depending on the cell configurations data, there are cases where it outperforms the MIP model;

2. In the constraint programming model, providing a search strategy can have a
significant impact on performance. The CP model can find a feasible solution faster than the MIP model, but it continues to explore the solution space until it finds whether the solution found is the final objective;

3. Performance depends highly on the input data. Generally, using the MIP models, the table layout problem is more difficult to solve (it takes longer to find the solution) when the page width is closer to the minimum table width. Nonetheless, the experimental results showed exceptions to this;

4. Using the models presented in this paper it is more difficult to solve the table layout problem when there is a large number of rows, columns and column configurations than when the table has a large number of cell configurations;

5. The constraint programming model may be slower than the MIP model due to the version of the CP engine used in the experiments presented in this paper. It is important to note that ILOG stopped supporting the CP engine in 2005 with version 4.0 of OPL Development Studio. While ILOG re-introduced
constraint programming back into OPL Development Studio in 2007 not all original features are available, especially search features;

6. Using a modelling language allows user imposed constraints to be easily added to the model;

7. As both the MIP and CP models use the set of cell configurations as input, paragraphing – determining cell configurations or how to split the text on consecutive lines – is an essential aspect as it impacts on the quality and computational time of these solutions.

The problems we have considered here contain only non-spanning cells. The obvious next step is to define an enhanced model for tables that contain spanning cells or other constraints met in real life such as inner tables, equal width columns, constraints on groups of columns, etc.

Some of these problems and also the problem of splitting wide tables across pages are discussed and some solutions are suggested in the following chapter.
Chapter 4

Advanced Table Layout Optimization

In the previous chapter we introduced the table layout optimization problem for simple tables that do not contain spanning cells. It has been proved in the past [0, 86] that if the table contains text the problem is NP-hard and therefore it requires long computational time. We proposed to solve the problem as a constraint optimization problem and we provided an integer programming definition. We evaluated the running time for various models including Mixed Integer Programming and Constraint Programming models.

However, in real life most of the tables contain spanning cells or any other type of content, figures, equations and even other nested tables which we discuss in this chapter. In Section 4.1 we present the table layout optimization problem for tables with spanning cells and we suggest a mixed integer programming model that uses a new strategy. We report on the performance improvements of this model compared with the MIP model presented in the Chapter 3. To understand the performance other combinatorial optimization methods for the table layout optimization problem we introduce in Section 4.2 a model based on a local search method. In Section 4.4 we present a comparative evaluation of the performance of all the models and strategies.
that have been proposed in this chapter including two new constraint programming models.

In subsection 4.5 we discuss the problem of optimizing table layout for tables with inner tables. We suggest a simple but effective solution for this case and discuss the implications of using a possible recursive constraint optimization model.

Often, the author has to deal with wide or long tables that require multiple pages in either horizontal or vertical directions. Even if the layout with minimum height for such a table is determined when the table is presented on multiple pages the author is still required to find an arrangement of columns that will minimize the number of pages used and minimize the impact on the reader’s ability to understand the meaning of the data. This raises further optimization problems such as finding suitable arrangements of columns or rows in pages which minimizes the number of column changes or, when splitting groups is acceptable, minimizing the number of logical groups that are split across pages. While current document authoring tools provide features for splitting long tables, splitting wide tables is not supported as we showed in our survey presented in Section 1.7.

In Section 4.6 we discuss how tables can be split across multiple pages in an optimal way. We propose in Section 4.6.2 a MIP model that finds an arrangement of columns in pages which minimizes the number of pages used to display the table while minimizing changes in the columns’ relative positions. When the columns of a wide table are grouped for aesthetic or logical reasons the author has to find an arrangement of rows/columns that not only minimizes the number of pages used and, at the same time, minimizes the number of groups that are divided between pages. We suggest in Section 4.6.3 an optimization model that solves this problem and we report on the performance of the table splitting solutions for various tables sizes.
4.1 Table Layout Optimization for Tables with Spanning Cells

In this section we present an optimization model for the table layout problem for tables that contain spanning cells. For example, in Figure 4.1 we show a table with 5 columns and 10 rows, 2 spanning cells and a variable number of text configurations. Using our optimization model we determined the layout with the minimum height when the page width was constrained to 190 points. It can be observed that any change in the configuration of any cell will cause a chain of changes which will result in a layout with at least the same height or larger.

We showed that the running time for the basic optimization model BMIP becomes unsatisfactory as tables increase in size. We now present an improved strategy where we use additional binary variables for selecting row heights and column widths from a predetermined set. Then, we select only those cell configurations that can fit in the selected column widths and row heights. We show in Section 4.1.3 that the significant performance improvement of this strategy allows us to experiment with even larger tables. As the objective function of the model is to minimize table
height this will ensure that the shortest and therefore the widest cell configurations are selected.

Starting from the set of cell configurations \( C_{i,j} \) we determine for each column \( j \) the set of possible column widths \( W_j \) as the union of non-spanning cell width values. The set of column widths \( W_j \) is

\[
W_j = \{ w^k_{i,j} \mid 1 \leq i \leq m, 1 \leq k \leq |C_{i,j}|, S^w_{i,j} = 0 \}
\]

and the set of row heights \( H_i \) is defined as

\[
H_i = \{ h^k_{i,j} \mid 1 \leq j \leq n, 1 \leq k \leq |C_{i,j}|, S^h_{i,j} = 0 \},
\]

where \( S^w_{i,j} \) is the number for additional spanned columns and \( S^h_{i,j} \) is the number of additional spanned rows by each cell \( i, j \).

The width \( w_j \) of column \( j \) is now determined by finding an index \( c_j \) with \( 1 \leq c_j \leq |W_j| \) while the height \( h_i \) of each row \( i \) is determined by selecting an index \( r_i \) with \( 1 \leq r_i \leq |H_i| \):

\[
w_j = W^c_j, \quad h_i = H^r_i
\]

and the layout width \( width(\mathcal{L}) \) and the layout height \( height(\mathcal{L}) \) are defined as:

\[
width(\mathcal{L}) = \sum_{j=1}^{n} w_j \quad (4.1)
\]
\[
height(\mathcal{L}) = \sum_{i=1}^{m} h_i \quad (4.2)
\]

The table layout problem is to minimize the table layout \( height(\mathcal{L}) \) by finding for each row the height with index \( r_i \), for each column the width with index \( c_j \) and
for each cell $i, j$ the configuration with the index $k$ with $1 \leq k \leq |C_{i,j}|$, such that

\[
\text{width}(\mathcal{L}) \leq W
\]

\[
w_{i,j}^k \leq \mathcal{W}_j^{c_i}
\]

\[
h_{i,j}^k \leq \mathcal{H}_i^{h_i}.
\]

From a layout perspective, when a cell $i, j$ spans a number of columns there is an interdependence between the cell’s width $w_{i,j}$ and the width of the columns it spans, i.e. the cell’s width must be equal to the sum of the spanned columns widths. Similarly, when a cell spans a number of rows the cell’s height $h_{i,j}$ must be equal to the sum of the spanned rows heights. Thus

\[
w_{i,j} = \sum_{c=j}^{j+S_{i,j}^w} w_c
\]

(4.3)

and

\[
h_{i,j} = \sum_{r=i}^{i+S_{i,j}^h} h_r.
\]

(4.4)

When tables contain multiple spanning cells, these cells can span over a disjoint set of columns or, when the spanning cells are on different rows, the spanned columns can overlap. This adds another level of complexity to the problem. In Figure 4.1 cell 2,2 spans over three columns while cell 8,3 also spans three columns with two of the spanned columns in common. It is easy to observe the interdependency between these 2 spanning cells. When solving the table layout problem, any change in the configuration of cell 2,2 or 8,3 will directly affect not only the configuration of the columns that each cell spans but indirectly the configuration of the columns spanned by both cells.

A case that needs consideration is presented in Figure 4.2(a) where cell 1, 1 has only one configuration which is wider than its spanning columns.
Glue is required for narrow columns.

The glue is displayed in a gray background.

Figure 4.2: A spanning cell over narrow columns requires glue

When computing the width of a spanning cell two cases must be considered:

Case 1 when the total width of the spanned columns is larger than the width of the spanning cell as presented in Figure 4.3(a).

Case 2 when the total width of the spanned columns is less than the width of the spanning cell as presented in Figure 4.3(b).

In the first case, as the widest cell configuration is selected for the spanning cell, the responsibility of drawing it using the dimensions of the spanned columns is left with a post-processing drawing program.

For the second case we introduce the glue variable, similar to the model introduced by Knuth [51] when determining the required character and word space for aligning text in paragraphs. The glue represents the amount that the spanned columns need to stretch in order to have a total width equal to the width of the spanning cell. In Figure 4.2(b) the glue is presented in a grey background.
4.1.1 Integer Programming Definition

The MIP model defined in Section [3.4] requires the width of a cell to be less than or equal to the selected width for the column. An integer programming model for tables with spanning cells requires additional constraints: the width of a spanning cell must be equal to the sum of widths for the columns it spans and a similar constraint for row heights. When the width of a spanning cell is less than the sum of column widths the spanning cell’s width is set during the drawing phase. When the width of a spanning cell is greater than the sum of column widths then the glue value is computed for each spanned column.

Thus, for a spanning cell $i, j$ equations 4.3 and 4.4 above are changed to:

\[
\begin{align*}
    w_{i,j} &= \sum_{c=j}^{j+S_{i,j}} (w_c + g_c^w) \\
    h_{i,j} &= \sum_{r=i}^{i+S_{i,j}} (h_r + g_r^h)
\end{align*}
\] (4.5)

where $w_c$ is the width of column $c$, $g_c^w$ is the horizontal glue of column $c$, $h_r$ is the height of row $r$ and $g_r^h$ is the vertical glue for each row $r$.

The layout width $\text{width}(\mathcal{L})$ and the layout height $\text{height}(\mathcal{L})$ are now defined as:

\[
\begin{align*}
    \text{width}(\mathcal{L}) &= \sum_{j=1}^{n} (w_j + g_j^w) \\
    \text{height}(\mathcal{L}) &= \sum_{i=1}^{m} (h_i + g_i^h)
\end{align*}
\] (4.7)

The definition of TSC - the integer programming model for the table layout
optimization problem for tables with spanning cells is:

\[
\text{minimize } \sum_{i=1}^{m} h_i^r \cdot y_i^r + g_i^h \tag{4.9a}
\]

subject to

\[
\sum_{j=1}^{n} w_j^c \cdot z_j^c + g_j^w \leq W \tag{4.9b}
\]

\[
w_{i,j} \leq \sum_{c=j}^{j+k_{i,j}} w_c + g_c^w \quad \forall i, j \tag{4.9c}
\]

\[
h_{i,j} \leq \sum_{r=i}^{i+h_{i,j}} h_r + g_r^h \quad \forall i, j \tag{4.9d}
\]

\[
w_{i,j} = \sum_{k} w_{i,j}^k \cdot x_{i,j}^k \quad \forall i, j \tag{4.9e}
\]

\[
h_{i,j} = \sum_{k} h_{i,j}^k \cdot x_{i,j}^k \quad \forall i, j \tag{4.9f}
\]

\[
w_j = \sum_{j=1}^{n} w_j^c \cdot z_j^c \tag{4.9g}
\]

\[
h_i = \sum_{i=1}^{m} h_i^r \cdot y_i^r \tag{4.9h}
\]

\[
\sum_{k} x_{i,j}^k = 1, \quad \forall i, j \tag{4.9i}
\]

\[
\sum_{j=1}^{n} z_j^c = 1 \tag{4.9j}
\]

\[
\sum_{i=1}^{m} y_i^r = 1 \tag{4.9k}
\]

where

\[
w_j^c \in W_j, \ h_i^r \in H_i
\]

\[
y_i^r, \ g_j^w, g_c^w \in \mathbb{R}^+
\]

\[
x_{i,j}^k, \ y_i^r, \ z_j^c \in \{0, 1\},
\]

\[
1 \leq k \leq |C_{i,j}|, \ 1 \leq c \leq |W_j|, \ 1 \leq r \leq |H_i|
\]
In the integer programming model defined in Section 3.2, the constraint 3.1c requires that for each cell \( i, j \) one cell configuration must be selected. To satisfy this constraint for cells that are spanned by another cell and therefore have no configurations, we introduce dummy cells. The dummy cells have only one configuration with the width and height set to 0 and thus the constraint 3.1c is satisfied.

For non-spanning cells expressions 4.9c and 4.9d become \( w_{i,j} \leq w_j \) and \( h_{i,j} \leq h_i \). To simplify the constraint expressions we define cell dimensions as in expression 4.9e and 4.9f. Also, the width of each column and the height for each row are defined using decision variables \( z \) and \( y \) as in equations 4.9g and 4.9h.

4.1.2 TSC Model for Tables with Spanning Cells

The listings in this section show only the relevant parts of the TSC model. The full code is available in Appendix A.

We start our TSC model as presented in Listing 4.1 with two variables rowSpan and colSpan which represent the sets of row and column spanning cells as tuples \(<i,j,s>\) where \( s \) is the number of rows/columns spanned by cell \( i,j \). We build two vectors Sh[<i,j>] and Sw[<i,j>] where we store for each cell the number of additional rows and respectively columns it spans, i.e. for a non spanning cell Sw[i,j] contains 0 and if the cell spans 2 columns its value is 1.

Based on the observation that each row cannot be any shorter than the maximum value of each cell’s minimum height we compute these minimum values in minH[i]. In a similar manner we record for each column the maximum of minimum cell widths in minW[j]. These values are used to filter the number of possible row heights and column widths.

It is important to note that the width of column spanning cells is ignored when determining the minimum width for each column. In a similar way, the height of row spanning cells is ignored when computing the minimum row heights. In Listing 4.1 the widths of spanning cells are multiplied with \( 1-Sw[i,j] \) which is always 0 or
Listing 4.1: TSC - Initial data

```cpp
int pageW = ...; // page width
{CellConf} configs = ...; // the set of cell configurations
{CellSpan} rowSpan = ...; // the set of row spanning cells
{CellSpan} colSpan = ...; // the set of column spanning cells

// build the vectors of rows/columns spanned by each cell
int Sh[cells] = [i, j] : s | <i, j, s> in rowSpan];
int Sw[cells] = [i, j] : s | <i, j, s> in colSpan];

// compute minimum row heights and column widths
int minH[i in rows] = max(j in cols) min(<i, j, k> in configs) (k.h * (1 - Sh[i, j]));
int minW[j in cols] = max(i in rows) min(<i, j, k> in configs) (k.w * (1 - Sw[i, j]));
```

Listing 4.2: TSC - computing the set of column widths and row heights

```cpp
{Pair} rowHset = {<i, k.h> | <i, j, k> in configs : k.h >= minH[i] && Sh[i, j] == 0};
{Pair} colWset = {<j, k.w> | <i, j, k> in configs : k.w >= minW[j] && Sw[i, j] == 0};
```

a negative number and therefore the width value will always be less than the width of non-spanning cells.

First, as displayed in Listing 4.2, we determine the set of row heights rowHset from the set of cell configurations filtering for only cell heights greater than minimum row height and only for non spanning cells.

We define binary decision variables as presented in Listing 4.3. The selected cell configurations are stored in x[] while y[] and z[] indicate the selected heights and widths for each row and column. Two vectors Gh[] and Gw[] store the vertical and horizontal glue.

Table layout dimensions are computed as presented in Listing 4.4. The row height is computed as defined by equation 4.9h by selecting a height value from the set of row heights. The table width and height are computed, as defined by equation 4.7 and equation 4.8, by adding the horizontal and vertical glue.

In Listing 4.5 constraint ct2 deals with the interdependence between the height of the spanning cell and the height of the spanned rows. For a spanning cell its height has to be at most equal to the sum of heights for the rows it spans and the associated row glue. For a simple cell its height has to be at most equal with the
Listing 4.3: TSC - Decision variables

```plaintext
// cell configuration selector
dvar int x[configs] in 0..1;

// row height selector
dvar int y[rowHset] in 0..1;

// column width selector
dvar int z[colWset] in 0..1;

// row glue
dvar int Gh[rows] in 0..maxint;

// column glue
dvar int Gw[columns] in 0..maxint;
```

Listing 4.4: TSC - Layout dimensions

```plaintext
// compute cell height
dexpr int cellH[i,j] in cells = sum<i,j,k> in configs) x[i,j,k] * k.h;

// compute row height
dexpr int rowH[i in rows] = sum<i,h> in rowHset) y[i,h] * h;

// compute table height
dexpr int tableH = sum(i in rows) (rowH[i] + Gh[i]);

// compute cell width
dexpr int cellW[i,j] in cells = sum<i,j,k> in configs) x[i,j,k] * k.w;

// compute column width
dexpr int colW[j in columns] = sum<j,w> in colWset) z[j,w] * w;

// compute table width
dexpr int tableW = sum(j in columns) (colW[j] + Gw[j]);
```
Listing 4.5: TSC Model constraints

1 minimize tableH;
2
3 constraints
4 {
5  ct1: // table width must be less than page width
6     tableW <= pageW;
7
8  ct2: // height constraint
9      forall (i in rows, j in columns)
10     if (Sh[i,j] > 0)
11        cellH[i,j] <= sum(row in i..i+Sh[i,j]) (rowH[row] + Gh[row]);
12     else
13        cellH[i,j] <= rowH[i];
14
15  ct3: // width constraint
16      forall (j in columns, i in rows)
17     if (Sw[i,j] > 0)
18        cellW[i,j] <= sum(col in j..j+Sw[i,j]) (colW[col] + Gw[col]);
19     else
20        cellW[i,j] <= colW[j];
21
22  ct4: // only one selected cell configuration
23      forall (i in rows, j in columns)
24     sum(<i,j,k> in configs) x[i,j,k] == 1;
25
26  ct4_1: // only one selected row height
27      forall (i in rows)
28     sum(<i,k> in rowHset) y[i,k] == 1;
29
30  ct4_2: // only one selected column width
31      forall (j in cols)
32     sum(<i,k> in colWset) z[j,k] == 1;
33 }
Listing 4.6: TSC - Equal glue distribution

```
...  

dexpr int glueW[i, j, s] in colSpan = 
  cellW[i, j] - sum(col in j..j+s) colW[col];

constraints
{
  ... 
  ct5: // equal glue distribution
    forall<(i, j, s) in colSpan)
      forall(col in j..j+s)
        Gw[col] <= glueW[i, j, s] / (s+1);
}
```

constraint ct5 can be added to the TSC model as presented in Listing 4.6 where the total glue $\text{glueW}$ is equally distributed to all $s$ columns spanned by cell $i, j$. The horizontal glue $\text{glueW}$ is defined on line 2 as the difference between the cell’s width and the sum of column widths for the columns that is spans. The vertical glue that applies to row spans can be computed in a similar way.

Such a constraint may cause a conflict when determining the minimum height layout especially when the table contains multiple spanning cells on overlapping column sets. If a column is spanned by two cells as in Figure 4.1, an equal distribution of glue is not possible as the glue is constraint by the two spanning cells. The second spanning cell can have a cell configuration with a smaller height that will not be selected and thus the minimum height layout is affected by the constraint for the equal distribution of glue.
4.1.3 Performance Evaluation of the TSC vs MIP Models

In this section we present experimental results using the TSC model and we compare its run time with the MIP model presented in Section 3.4. In the MIP model, which only works on simple tables, we use a simplified strategy where we first find column width configurations from a predefined set, then we select cell configurations that can fit in the selected column width. The row height is computed as the maximum cell height for all the cells in the row. In the TSC model, both column widths and row heights are selected from a pre-determined set of possible values with the condition that one cell configuration which has suitable dimensions can be selected. In order to compare the performance of the two models we run the TSC model with the same hardware specifications, same CPLEX engine and the same data sets as we used for the MIP model (see Section 3.7 for details).

Compared with the running time of the MIP model, our experiments showed a substantial improvement on the computational time using the TSC model.

Table 4.1 shows the running time for tables with 400 cells, in increasing size, up to tables with 3,600 cells (from 20x20 tables up to 60x60 tables) with maximum 6 words in each cell, for both MIP and TSC models. The times are in the format h:mm:ss. \(\text{minW}\) and \(\text{maxW}\) represent the minimum and maximum table width, computed from the cell-width configurations. We report the running time for page width constraint to be \(\text{minW}\) plus 25%, 50% and 75% of the difference between \(\text{maxW}\) and \(\text{minW}\). The summary line provides information regarding the minimum and maximum table configurations, the number of cell configurations \(\text{cnf}\), distinct column widths \(\bigcup_j W_j\) and row heights \(\bigcup_i H_i\).

For a 20x20 table the run time of the TSC model was halved or reduced to under 1s when the layout was constraint to 75% of its maximum width. The improvement in the run time can especially be observed in the case of larger tables. The TSC model finds the same solutions as the MIP model but only in a fraction of the time.
Table 4.1: Run time for MIP and TSC models.

<table>
<thead>
<tr>
<th>Table</th>
<th>width</th>
<th>height</th>
<th>+25%</th>
<th>+50%</th>
<th>+75%</th>
<th>maxW</th>
</tr>
</thead>
<tbody>
<tr>
<td>20x20x6</td>
<td>1,320</td>
<td>1,040</td>
<td>2000</td>
<td>600</td>
<td>2500</td>
<td>380</td>
</tr>
<tr>
<td></td>
<td>440</td>
<td>3,000</td>
<td>2,000</td>
<td>00:01</td>
<td>00:21</td>
<td>00:06</td>
</tr>
<tr>
<td></td>
<td>2,500</td>
<td>00:27</td>
<td>00:01</td>
<td>380</td>
<td>00:02</td>
<td>00:01</td>
</tr>
<tr>
<td></td>
<td>4,284</td>
<td>00:01</td>
<td>summary</td>
<td>m=1320x1170; M=4284x200; cnf=1382;</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
|        | 1,040 | 30x30x6 |              | \[
\bigcup_j W_j \bigcup_i H_i = 592; \bigcup_i H_i = 79
\] |
| 30x30x6 | 2,094 | 1560 | 3,500 | 770  | 4,500 | 570  | 440  | 300  |
|        | 2,094 | 00:10 | 00:02 | 00:01 | summary | m=2094x1780; M=6709x300; cnf=3082; | |
|        | 00:04 | 00:01 | \[
\bigcup_j W_j = 1145; \bigcup_i H_i = 110
\] |
| 40x40x6 | 2,879 | 2,140 | 5,000 | 1280 | 6,000 | 780  | 680  | 400  |
|        | 2,140 | 01:48 | 01:57 | 1:26:10 | summary | m=2879x2400; M=9189x400; cnf=5546; | |
|        | 00:17 | 00:05 | \[
\bigcup_j W_j = 1908; \bigcup_i H_i = 149
\] |
| 50x50x6 | 3,603 | 2,790 | 5,200 | 1,700 | 7,000 | 1,110 | 870  | 500  |
|        | 2,790 | n/a   | n/a   | n/a  | summary | m=3603x3000; M=11670x500; cnf=8850; | |
|        | 00:43 | 00:05 | \[
\bigcup_j W_j = 2877; \bigcup_i H_i = 198
\] |
| 60x60x6 | 4,368 | 3,3660 | 7,000 | 1,720 | 9,000 | 1,210 | 880  | 600  |
|        | 3,3660 | n/a | n/a | n/a | summary | m=4368x3600; M=13918x600; cnf=12580; | |
|        | 00:04 | 00:01 | \[
\bigcup_j W_j = 3775; \bigcup_i H_i = 246
\] |
In Figure 4.4 we show that for a 30x30 table the TSC running time for a page width of 3,500 points was only 4s compared with the running time of the MIP model of 1 hour 39 minutes and 49 seconds. A similar reduction of run time occurred on a 40x40 table for a page width of 7,000 points where the running time was reduced from 1 hour 26m 10s for the MIP model to only 40s for the TSC model.

As the TSC model proved to be much more efficient we are now able to report the results for larger tables with 2,500 cells (50x50) and even 3,600 cells (60x60). The running time of the MIP model could not be realistically reported for these large tables but using the TSC model we found solutions in acceptable running time as presented in Table 4.1. For a 50x50 table with 8,850 cell configurations it took only 28s to determine a layout that was constrained to 75% of its maximum width and only 2 minutes and 9s for a 60x60 table with 12,580 configurations.

It is important to note that the running time is highly dependent on the complexity of the data in each sample. In some cases, finding a solution for a 30x30 table can take longer than finding solutions for a 60x60 table.
4.2 Local Search Model - LSP

It has been highlighted on many occasions that authors that cannot afford the time to find an optimal solution or authors that are not necessarily interested in the optimal solution would be satisfied if a table layout that meets certain criteria is identified in an acceptable time frame.

One of the methods that has been proposed in the field of operations research for solving computationally hard optimization problems such as the table layout problem is *local search*. Even if this method does not guarantee optimal solutions we introduce a local search model for the table layout optimization problem because we aim at comparing its effectiveness in contrast with the integer and constraint programming methods. Our interest in this method has been triggered by the recent development of *LocalSolver*, a local search engine [10] developed at Bouygues e-lab, Paris, France. Also, we have noted instances where neither the CPLEX or CP Optimizer engines can determine in an acceptable time frame if a solution is optimal. Therefore we want to determine if an alternative method such as local search can be as effective in practical cases when optimality is not required.

In this section we introduce a local search model developed for LocalSolver 2.0. This engine has been released in March 2012. Compared with its previous version the new version of the engine allows the developer to model optimization problems in a high level mathematical language similar to OPL or other mathematical programming languages. In addition, LocalSolver 2.0 has the strength of standard programming languages allowing the author to include pre-processing or post-processing functions in a simple manner. It also allows the integration of the optimization engine in larger programs through its APIs for C++, Java and .NET technologies.

In Listing 4.7 we present the main function of a local search model LSP for solving the simple table layout problem. The strategy behind this model has been suggested by Thierry Benoist of Bouygues e-lab and it is somehow different to the
strategies that we presented so far. In a pre-processing step, an array of minimal cell heights \( \text{minHeightForWidthIndex}[r][c][n] \) is built for each possible column width. In order to minimize the number of decision variables the model finds only which column width must be selected from the array of possible column widths \( \text{colWidths}[c][n] \). The LSP model only finds values for the array of binary variables \( \text{xcol}[c][n] \) where \( c \) is the column index and \( n \) is the width index for column \( c \).

The array \( \text{nbWidths}[c] \) on line 2 stores the number of possible column widths for column \( c \). The only additional constraint to the table width being less than the page width (line 16) is as defined on line 5 that for each column only one width must be selected in \( \text{xcol}[c][n] \).

The key difference of this model is that the cell height is not computed but simply selected from the array \( \text{minHeightForWidthIndex} \) using the same selector variable \( \text{xcol}[c][n] \) as shown online 10. The row height is then computed on line 11 in a similar manner to the BMIP model as the maximum cell height for all the cells of a row \( r \).
4.3 An Alternative CP Model - CPI

In Section 3.5 we presented a simple CP model where the binary decision variables from the MIP model are replaced with integer variables for which the optimization engine finds values in the domains defined by the set of cell configurations.

We now define a constraint programming model CPI1 for the table layout optimization problem where instead of finding width/height values for the integer decision variables we use the constraint programming engine’s ability to find solutions to models that use arrays indexed by decision variables. A similar model has been suggested by Gange et al. [40] in 2011 using the Zinc [60] modelling language.

In Listing 4.8 the arrays of decision variables $sc[r][c]$, $sw[c]$ and $sh[r]$ are used to store the indexes of the selected cell configurations, column widths and row heights respectively.

We have pre-computed the array of cell widths $cellWarr[r][c][kc]$, cell heights $cellHarr[r][c][kc]$, column widths $colWarr[c][kw]$ and row heights $rowHarr[r][kh]$ from the set of cell configurations $cellC[]$ where:

1. $r$ is a range from 0 to $m-1$,
2. $c$ is a range from 0 to $n-1$,
3. $kw$ is a range 0 to $Kw-1$ – the maximum number of column widths for any column $j$, and
4. $kh$ is a range 0 to $Kh-1$ – the maximum number of row heights for any row $i$.

Constraint ct2 ensures that only valid cell configurations are selected (excluding dummy configurations) while constraints ct3 and ct4 ensure that the selected cell configuration can fit in the select column width and cell height respectively.

Listing 4.8: CPI1 - Constraint Programming model with index decision variables

```plaintext
1 . . .
2 {Conf} cellC[i,j] in cells = {k | <i,j,k> in configs};
```
// sets of column widths and row heights for each column j and row i

sorted \{ \text{int} \} \text{colWset}[j \text{ in } \text{cols}] = \{ k.w | <i,j,k> \text{ in } \text{configs} : k.w \geq \text{minW}[j] \};

sorted \{ \text{int} \} \text{rowHset}[i \text{ in } \text{rows}] = \{ k.h | <i,j,k> \text{ in } \text{configs} : k.h \geq \text{minH}[i] \};

\text{int} m = \text{card(rows)};
\text{int} n = \text{card(cols)};

range r = 0..m-1;
range c = 0..n-1;
range kw = 0..Kw-1;
range kh = 0..Kh-1;
range kc = 0..Kc-1;

\text{int} \text{cellWarr}[r][c][kc] = \{ i : \{ j : \{ \text{ord} (\text{cellC} \{i,j\}, q) : q.w \} | i \text{ in } \text{rows}, j \text{ in } \text{cols}, q \text{ in } \text{cellC} \{i,j\} \};

\text{int} \text{cellHarr}[r][c][kc] = \{ i : \{ j : \{ \text{ord} (\text{cellC} \{i,j\}, q) : q.h \} | i \text{ in } \text{rows}, j \text{ in } \text{cols}, q \text{ in } \text{cellC} \{i,j\} \};

\text{int} \text{colWarr}[c][kw] = \{ j : \{ \text{ord} (\text{colWset}[j], q) : q \} | j \text{ in } \text{cols}, q \text{ in } \text{colWset}[j] \};

\text{int} \text{rowHarr}[r][kh] = \{ i : \{ \text{ord} (\text{rowHset}[i], q) : q \} | i \text{ in } \text{rows}, q \text{ in } \text{rowHset}[i] \};

dvar \text{int} \text{sc}[r][c] \text{ in } \text{kc}; \quad // \text{cell configuration index}

dvar \text{int} \text{sw}[c] \text{ in } \text{kw}; \quad // \text{column width index}

dvar \text{int} \text{sh}[r] \text{ in } \text{kh}; \quad // \text{row height index}

dexpr \text{int} \text{cellW}[i \text{ in } \text{r}][j \text{ in } \text{c}] = \text{cellWarr}[i][j][\text{sc}[i][j]];

dexpr \text{int} \text{cellH}[i \text{ in } \text{r}][j \text{ in } \text{c}] = \text{cellHarr}[i][j][\text{sc}[i][j]];

dexpr \text{int} \text{colW}[j \text{ in } \text{c}] = \text{colWarr}[j][\text{sw}[j]];

dexpr \text{int} \text{rowH}[i \text{ in } \text{r}] = \text{rowHarr}[i][\text{sh}[i]];

dexpr \text{int} \text{tableW} = \sum (j \text{ in } \text{c}) \text{colW}[j];

dexpr \text{int} \text{tableH} = \sum (i \text{ in } \text{r}) \text{rowH}[i];

\text{minimize} \text{tableH};

\text{subject to}
{
  \text{ct1:} \quad // \text{table width must be less than page width}
  \text{tableW} \leq \text{pageW};
}
For this model we need to use a multi-dimensional array of cell configurations. However, during the development of this model we identified that while IBM ILOG’s CP engine allows an array of tuples Conf v1[r][c][kw] to be defined and validated, an exception occurs at run time. Therefore we were forced to use an array of integers cellWarr[r][c][kc] to store cell width values and another array of integers cellHarr[r][c][kc] to store cell height values. While this can be treated as a simple alternative solution it has a direct impact on the performance of the CP engine as we found out during the evaluation of the model which is presented in the following section.

In order to reduce the number of constraints as in the strategy implemented by the local search model we also implement a variation CPI2 of the constraint programming model described above. In this CPI2 model we only define decision variables for selecting column widths and row heights.

The CPI2 model is shown in Listing 4.9 where the decision variables sw[c] and sh[r] are similar to the CPI1 but no decision variables are defined for selecting cell configurations. The model has only 2 constraints: ct1 which ensures that the width of overall layout is less than the page width and ct2 which ensures that for any selected combination of column widths/row heights there is at least one suitable cell configuration.
Listing 4.9: CPI2 - Constraint Programming model with reduced index decision variables

```plaintext
1 int colWarr[c][kw] = [j : [ord(colWset[j], q): q | j in c, q in colWset[j]]];
2 int rowHarr[r][kh] = [i : [ord(rowHset[i], q): q | i in r, q in rowHset[i]]];
3 dvar int sw[c] in kw; // column width index
4 dvar int sh[r] in kh; // row height index
5 dexpr int colW[j in c] = colWarr[j][sw[j]];
6 dexpr int rowH[i in r] = rowHarr[i][sh[i]];
7 dexpr int tableW = sum(j in c) colW[j];
8 dexpr int tableH = sum(i in r) rowH[i];
9 minimize tableH;
10 subject to
11 { ct1: tableW <= pageW;
12   ct2: // at least one suitable cell configuration exists
13     forall(i in rows, j in cols)
14       sum(<i, j, k> in configs) (k.w <= colW[j] & k.h <= rowH[i]) >= 1;
15 }
```

4.4 Overall Evaluation of Optimization Methods

To evaluate the performance of the models and the strategies proposed so far we generated a new set of tables. We want to understand and report on the limits of the techniques presented in this thesis and therefore we increased the table size from 10x10 tables, going up to 200x200 tables. While such extra large tables cannot be easily generated manually they appear in practice for example when data is generated by computer systems. We generated tables with random text configurations with up to 6 words in each cell in a similar manner to the test data presented in Section 3.7 and Section 4.1.3.

As shown in our previous evaluations and also reported by Gange et al. [40], due to the random nature of the text selected in each cell, these tests have proven to be the most challenging for the optimization engines. Gange et al. found after collecting over 50,000 tables from the web by crawling 10,000 web pages that most tables were easy to optimize. Therefore from this set of tables only 2,063 (4%) were selected as being relevant for evaluating the performance of the optimization
engines.

The results presented in this section have been obtained by using IBM ILOG OPL IDE 6.3 with CPLEX engine 12.3 running on a 64-bit Linux laptop with an Intel i7-2640M CPU at 2.80GHz and 8Gb RAM. For the LSP model we used LocalSolver 2.0 on the same hardware platform.

We will first report in Section 4.4.1 the performance of the TSC model for table size ranging from 10x10 to 40x40. In Section 4.4.2 a present comparative evaluation of the performance of all the models suggested in this thesis.

4.4.1 Mixed Integer Programming - TSC Model

In this section we report on the performance of the TSC model presented in Section 4.1. All the values reported are based on tests performed on batches of tables with cell configurations determined by selecting for each each cell a random number of words. It is important to note that only the number of words in the text of each cell is randomly generated. The text is selected from a real document partially presented in Appendix E and the dimensions of cell configurations are determined using graphical functions for text size that take into account graphical parameters such as font size, font style and spacing between characters, words and lines. Each batch consists of at least 100 tests for an incremental squeeze factor $s_f$ of 10%. The squeeze factor $s_f$ varies from 0% to 100% meaning the table width is constrained between its minimum and maximum values.

We first focus on the evaluation of the running time for various table sizes. We then report on the optimal table height compared to the maximum table height in order to estimate the intervals where optimal table heights can be found.

Running time evaluation

We report the median, minimum and maximum running times because in many instances the average time does not always reflect correctly the overall performance.
Figure 4.5: TSC model running time evaluated for 100 tables 10x10 and up to 6 words of the TSC method as we will show in the case of the 30x30 tables. The summary line in each of the tables presented in this section shows the overall average, median, minimum and maximum values.

For each test we set a maximum time limit of 300 seconds. This time limit was never reached for 10x10 and 20x20 tables and for 30x30 tables it was reached in only one test. The time limit played a significant role for the 40x40 tables as we show below. With the exception of Figure 4.5 the running times presented in all the

Table 4.2: Summary of TSC running time (s) for 10x10x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.06</td>
<td>0.05</td>
<td>0.04</td>
<td>0.18</td>
</tr>
<tr>
<td>10%</td>
<td>0.11</td>
<td>0.10</td>
<td>0.06</td>
<td>0.22</td>
</tr>
<tr>
<td>20%</td>
<td>0.14</td>
<td>0.12</td>
<td>0.06</td>
<td>0.28</td>
</tr>
<tr>
<td>30%</td>
<td>0.12</td>
<td>0.11</td>
<td>0.07</td>
<td>0.23</td>
</tr>
<tr>
<td>40%</td>
<td>0.10</td>
<td>0.09</td>
<td>0.06</td>
<td>0.20</td>
</tr>
<tr>
<td>50%</td>
<td>0.12</td>
<td>0.12</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td>60%</td>
<td>0.12</td>
<td>0.11</td>
<td>0.06</td>
<td>0.24</td>
</tr>
<tr>
<td>70%</td>
<td>0.12</td>
<td>0.11</td>
<td>0.06</td>
<td>0.22</td>
</tr>
<tr>
<td>80%</td>
<td>0.10</td>
<td>0.10</td>
<td>0.04</td>
<td>0.19</td>
</tr>
<tr>
<td>90%</td>
<td>0.09</td>
<td>0.09</td>
<td>0.05</td>
<td>0.18</td>
</tr>
<tr>
<td>100%</td>
<td>0.05</td>
<td>0.05</td>
<td>0.05</td>
<td>0.08</td>
</tr>
<tr>
<td></td>
<td>0.10</td>
<td>0.10</td>
<td>0.04</td>
<td>0.28</td>
</tr>
</tbody>
</table>
In Figure 4.5 we show the average, medium, minimum and maximum running times for 10x10 tables with maximum 6 words per cell. The shortest running time was 0.04s and the maximum running time was 0.28s with an overall average of 0.1s to determine optimal solutions. The running time for this batch of tables is presented in Table 4.2.

Similarly we show the running times for 20x20, 30x30 and 40x40 tables in Figure 4.6: TSC model running time evaluated for 100 tables 20x20 and up to 6 words

figures of this section are logarithmic.

![Graph](image.png)

**Table 4.3:** Summary of TSC running time (s) for 20x20x6 tables

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.28</td>
<td>0.27</td>
<td>0.21</td>
<td>0.39</td>
</tr>
<tr>
<td>10%</td>
<td>0.90</td>
<td>0.83</td>
<td>0.42</td>
<td>2.18</td>
</tr>
<tr>
<td>20%</td>
<td>1.23</td>
<td>1.09</td>
<td>0.53</td>
<td>3.91</td>
</tr>
<tr>
<td>30%</td>
<td>0.67</td>
<td>0.64</td>
<td>0.31</td>
<td>1.65</td>
</tr>
<tr>
<td>40%</td>
<td>0.58</td>
<td>0.56</td>
<td>0.28</td>
<td>1.01</td>
</tr>
<tr>
<td>50%</td>
<td>1.25</td>
<td>1.01</td>
<td>0.34</td>
<td>6.55</td>
</tr>
<tr>
<td>60%</td>
<td>1.45</td>
<td>1.07</td>
<td>0.54</td>
<td>5.64</td>
</tr>
<tr>
<td>70%</td>
<td>1.32</td>
<td>1.00</td>
<td>0.41</td>
<td>5.36</td>
</tr>
<tr>
<td>80%</td>
<td>1.01</td>
<td>0.78</td>
<td>0.3</td>
<td>5.11</td>
</tr>
<tr>
<td>90%</td>
<td>0.47</td>
<td>0.44</td>
<td>0.22</td>
<td>1.28</td>
</tr>
<tr>
<td>100%</td>
<td>0.18</td>
<td>0.18</td>
<td>0.14</td>
<td>0.74</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th>sf (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.85</td>
<td>0.78</td>
<td>0.14</td>
<td>6.55</td>
<td></td>
</tr>
</tbody>
</table>
Figure 4.7: TSC model running time evaluated for 100 tables 30x30 and up to 6 words per cell. For 20x20 tables with up to 6 words per cell the running time was between 0.14s and 6.55s with an average of 0.85s and a median time of 0.78s as shown in Table 4.3.

For 30x30 tables with up to 6 words per cell the running time was between 0.31s and 300.25s with an average of 10.16s and a median time of only 2.61s as displayed in Table 4.4. This shows that even for this larger instance, most of the solutions can be found under 2 seconds but we noticed in only one case that while a solution was

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.67</td>
<td>0.64</td>
<td>0.44</td>
<td>1.37</td>
</tr>
<tr>
<td>10%</td>
<td>3.08</td>
<td>2.61</td>
<td>1.32</td>
<td>11.71</td>
</tr>
<tr>
<td>20%</td>
<td>7.15</td>
<td>4.88</td>
<td>1.62</td>
<td>19.61</td>
</tr>
<tr>
<td>30%</td>
<td>3.04</td>
<td>2.11</td>
<td>0.90</td>
<td>15.53</td>
</tr>
<tr>
<td>40%</td>
<td>3.02</td>
<td>2.32</td>
<td>0.82</td>
<td>12.66</td>
</tr>
<tr>
<td>50%</td>
<td>19.48</td>
<td>15.50</td>
<td>2.71</td>
<td>146.56</td>
</tr>
<tr>
<td>60%</td>
<td>33.81</td>
<td>20.17</td>
<td>3.08</td>
<td>170.64</td>
</tr>
<tr>
<td>70%</td>
<td>27.88</td>
<td>15.03</td>
<td>2.15</td>
<td>300.25</td>
</tr>
<tr>
<td>80%</td>
<td>10.81</td>
<td>9.26</td>
<td>0.80</td>
<td>100.40</td>
</tr>
<tr>
<td>90%</td>
<td>2.40</td>
<td>1.44</td>
<td>0.49</td>
<td>11.28</td>
</tr>
<tr>
<td>100%</td>
<td>0.40</td>
<td>0.38</td>
<td>0.31</td>
<td>0.79</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>10.16</td>
<td>2.61</td>
<td>0.31</td>
<td>300.25</td>
</tr>
</tbody>
</table>
found, TSC could not determine if the solution was optimal in the preset time limit of 300s. This explains the maximum time of 300 seconds trying to prove optimality when the maximum table width was *squeezed* to 70% of the maximum table width. As a result the overall average time is increased to 10.16s and therefore the median running time of only 2.61s is more representative for this batch of tables.

For 40x40 tables with up to 6 words per cell the running time was between 0.56s and 306.65s with an average of 86.74s. As these instances are larger with 1,600 cells and 5,600 configurations the maximum running time of 300s was reached more often. Therefore in Table 4.5 we introduce an additional column T\textsubscript{TL} which shows how many tests reached this time limit. For a squeeze factor of 50\%, 60\% and 70\% the time limit was reached in 40, 78 and 49 tests respectively. This shows that only in 174 cases (17.01\% ) TSC could not prove optimality for tables with 40x40 and up to 6 words per cell. However, as we can see in Table 4.5 the median time for these harder cases, was only 16.46s which shows that optimal solutions can be identified in acceptable times for most of the 40x40 tables.

The overall average running times for all table sizes evaluated in this section
Table 4.5: Summary of TSC running time (s) for 40x40x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>TL</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>1.34</td>
<td>1.26</td>
<td>0.89</td>
<td>4.08</td>
<td>0</td>
</tr>
<tr>
<td>10%</td>
<td>8.13</td>
<td>6.62</td>
<td>2.83</td>
<td>49.83</td>
<td>0</td>
</tr>
<tr>
<td>20%</td>
<td>33.10</td>
<td>35.66</td>
<td>6.50</td>
<td>74.64</td>
<td>0</td>
</tr>
<tr>
<td>30%</td>
<td>16.37</td>
<td>8.69</td>
<td>2.35</td>
<td>50.36</td>
<td>0</td>
</tr>
<tr>
<td>40%</td>
<td>16.90</td>
<td>16.46</td>
<td>3.63</td>
<td>60.21</td>
<td>0</td>
</tr>
<tr>
<td>50%</td>
<td>238.63</td>
<td>284.37</td>
<td>23.98</td>
<td>306.65</td>
<td>40</td>
</tr>
<tr>
<td>60%</td>
<td>280.63</td>
<td>300.36</td>
<td>62.12</td>
<td>304.09</td>
<td>78</td>
</tr>
<tr>
<td>70%</td>
<td>263.93</td>
<td>300.00</td>
<td>28.88</td>
<td>304.26</td>
<td>49</td>
</tr>
<tr>
<td>80%</td>
<td>84.99</td>
<td>49.45</td>
<td>7.37</td>
<td>301.67</td>
<td>7</td>
</tr>
<tr>
<td>90%</td>
<td>9.49</td>
<td>5.85</td>
<td>1.52</td>
<td>47.26</td>
<td>0</td>
</tr>
<tr>
<td>100%</td>
<td>0.70</td>
<td>0.63</td>
<td>0.56</td>
<td>1.88</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>sf</th>
<th>10x10</th>
<th>20x20</th>
<th>30x30</th>
<th>40x40</th>
<th>Avg</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>0.06</td>
<td>0.28</td>
<td>0.67</td>
<td>1.34</td>
<td>0.59</td>
</tr>
<tr>
<td>10%</td>
<td>0.11</td>
<td>0.90</td>
<td>3.08</td>
<td>8.13</td>
<td>3.06</td>
</tr>
<tr>
<td>20%</td>
<td>0.14</td>
<td>1.23</td>
<td>7.15</td>
<td>33.10</td>
<td>10.40</td>
</tr>
<tr>
<td>30%</td>
<td>0.12</td>
<td>0.67</td>
<td>3.04</td>
<td>16.37</td>
<td>5.05</td>
</tr>
<tr>
<td>40%</td>
<td>0.10</td>
<td>0.58</td>
<td>3.02</td>
<td>16.90</td>
<td>5.15</td>
</tr>
<tr>
<td>50%</td>
<td>0.12</td>
<td>1.25</td>
<td>19.48</td>
<td>238.63</td>
<td>64.87</td>
</tr>
<tr>
<td>60%</td>
<td>0.12</td>
<td>1.45</td>
<td>33.81</td>
<td>280.63</td>
<td>79.00</td>
</tr>
<tr>
<td>70%</td>
<td>0.12</td>
<td>1.32</td>
<td>27.88</td>
<td>263.93</td>
<td>73.31</td>
</tr>
<tr>
<td>80%</td>
<td>0.10</td>
<td>1.01</td>
<td>10.81</td>
<td>84.99</td>
<td>24.23</td>
</tr>
<tr>
<td>90%</td>
<td>0.09</td>
<td>0.47</td>
<td>2.40</td>
<td>9.49</td>
<td>3.11</td>
</tr>
<tr>
<td>100%</td>
<td>0.05</td>
<td>0.18</td>
<td>0.40</td>
<td>0.70</td>
<td>0.33</td>
</tr>
</tbody>
</table>

are displayed in Figure 4.9. When the TSC model is evaluated over hundreds of test tables it can be seen that the problem is harder to solve when the table width is squeezed between 50% and 80%, with an average running time between 24.23s and 79.00s as displayed in Table 4.6. There is also a slight increase in the average running time of around 10.40s when the squeeze factor is around 20%. This effect can be observed for all table sizes, from 10x10 to 40x40. When the squeeze factor is closer to the minimum and maximum table widths the problem becomes easier to solve with the average running times of up to 4s.

Based on our tests the overall average running time for the TSC model is 24.46s with tables ranging from 10x10 to 40x40, with the average number of configurations ranging from 318 to 5,801 configurations when the cells have text with up to 6 words.
Optimal table height evaluation

From a table height perspective our tests have shown that the optimal table height can be expressed in terms of table’s maximum height for different squeeze factors.

There are many table layout algorithms implemented by various document authoring tools which will find a layout with the size based on user’s requirements. Evaluating if this layout has a height optimal or close to optimal is not easy because the optimal table height depends on many factors including the squeeze factor, the font size, the number of words in each cell and thus the number of cell configurations, etc. Even when an optimization engine is used to determine the optimal table height using the methods presented in this thesis we showed that there are instances when most of the running time is consumed by demonstrating that a solution is optimal. Some methods do not even attempt to prove optimality, i.e. heuristics or local search methods. If a feasible solution is found the optimization engines will, in some cases, continue to search the solution space in an attempt to prove that the solution is optimal or until a specified time limit is reached.
Table 4.7: Optimal height ratio for 10x10x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>σ</th>
<th>σ_{err}</th>
<th>Lw</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>71.65%</td>
<td>71.93%</td>
<td>64.41%</td>
<td>80.00%</td>
<td>3.55%</td>
<td>0.36%</td>
<td>70.96%</td>
<td>72.35%</td>
</tr>
<tr>
<td>10%</td>
<td>52.81%</td>
<td>53.33%</td>
<td>47.46%</td>
<td>57.89%</td>
<td>2.00%</td>
<td>0.20%</td>
<td>52.42%</td>
<td>53.21%</td>
</tr>
<tr>
<td>20%</td>
<td>44.35%</td>
<td>44.64%</td>
<td>40.74%</td>
<td>47.37%</td>
<td>1.45%</td>
<td>0.14%</td>
<td>44.07%</td>
<td>44.64%</td>
</tr>
<tr>
<td>30%</td>
<td>37.79%</td>
<td>37.93%</td>
<td>35.00%</td>
<td>40.74%</td>
<td>1.28%</td>
<td>0.13%</td>
<td>37.54%</td>
<td>38.04%</td>
</tr>
<tr>
<td>40%</td>
<td>33.56%</td>
<td>33.33%</td>
<td>30.51%</td>
<td>37.04%</td>
<td>1.14%</td>
<td>0.11%</td>
<td>33.34%</td>
<td>33.79%</td>
</tr>
<tr>
<td>50%</td>
<td>30.92%</td>
<td>31.03%</td>
<td>27.12%</td>
<td>33.33%</td>
<td>1.34%</td>
<td>0.13%</td>
<td>30.66%</td>
<td>31.18%</td>
</tr>
<tr>
<td>60%</td>
<td>28.38%</td>
<td>28.20%</td>
<td>25.00%</td>
<td>31.58%</td>
<td>1.38%</td>
<td>0.14%</td>
<td>28.11%</td>
<td>28.65%</td>
</tr>
<tr>
<td>70%</td>
<td>25.69%</td>
<td>25.86%</td>
<td>22.81%</td>
<td>29.09%</td>
<td>1.27%</td>
<td>0.13%</td>
<td>25.44%</td>
<td>25.94%</td>
</tr>
<tr>
<td>80%</td>
<td>22.92%</td>
<td>22.81%</td>
<td>20.34%</td>
<td>26.32%</td>
<td>1.22%</td>
<td>0.12%</td>
<td>22.69%</td>
<td>23.16%</td>
</tr>
<tr>
<td>90%</td>
<td>20.53%</td>
<td>20.69%</td>
<td>18.64%</td>
<td>22.81%</td>
<td>0.93%</td>
<td>0.09%</td>
<td>20.34%</td>
<td>20.71%</td>
</tr>
<tr>
<td>100%</td>
<td>17.28%</td>
<td>17.24%</td>
<td>16.67%</td>
<td>18.52%</td>
<td>0.39%</td>
<td>0.04%</td>
<td>17.20%</td>
<td>17.35%</td>
</tr>
</tbody>
</table>

Knowing with some level of confidence that the solution determined so far is close to the optimal value can be very relevant. The user can decide to stop the optimization engine from exploring the solution space as soon as the feasible solution found is within a pre-determined range. If the running time of the optimization process takes a long time, the user interface of a document authoring tool is expected to show a progress bar or a status bar that will indicate the estimated time until a solution is found. Knowing the range where optimal or close to optimal solutions are found could be used to indicate the progress in the optimization process.

Therefore, in this section we determine the range of the optimal table height using confidence intervals. We report on the ratio between the optimal table height and the maximum table height for various table sizes and squeeze factors based on a large set of tests. In Table 4.7, Table 4.8, Table 4.9 and Table 4.10 we show the average, median, minimum and maximum of the ratio between optimal and maximum table heights. We compute the standard deviation $\sigma$ and the standard error of the mean $\sigma_{err}$ using the average table height values. Thus we can compute approximate confidence intervals for the average optimal table height values. The $Lw$ and the $Up$ represent the lower and upper confidence limits for a probability of 95%. This means that each time an average is computed for the ratio between optimal and maximum table height, in 95% of the cases, this average will fall in these intervals.
Table 4.8: Optimal height ratio for 20x20x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>σ</th>
<th>σ_{err}</th>
<th>Lw</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>73.25%</td>
<td>73.33%</td>
<td>67.50%</td>
<td>79.17%</td>
<td>2.15%</td>
<td>0.22%</td>
<td>72.83%</td>
<td>73.67%</td>
</tr>
<tr>
<td>10%</td>
<td>53.60%</td>
<td>53.78%</td>
<td>50.00%</td>
<td>57.98%</td>
<td>1.37%</td>
<td>0.14%</td>
<td>53.33%</td>
<td>53.87%</td>
</tr>
<tr>
<td>20%</td>
<td>44.53%</td>
<td>44.54%</td>
<td>41.67%</td>
<td>47.06%</td>
<td>0.98%</td>
<td>0.10%</td>
<td>44.34%</td>
<td>44.72%</td>
</tr>
<tr>
<td>30%</td>
<td>36.80%</td>
<td>36.67%</td>
<td>35.00%</td>
<td>39.17%</td>
<td>0.72%</td>
<td>0.07%</td>
<td>36.66%</td>
<td>36.94%</td>
</tr>
<tr>
<td>40%</td>
<td>32.80%</td>
<td>32.50%</td>
<td>31.62%</td>
<td>34.19%</td>
<td>0.53%</td>
<td>0.05%</td>
<td>32.70%</td>
<td>32.91%</td>
</tr>
<tr>
<td>50%</td>
<td>30.92%</td>
<td>30.83%</td>
<td>29.06%</td>
<td>32.48%</td>
<td>0.62%</td>
<td>0.06%</td>
<td>30.80%</td>
<td>31.04%</td>
</tr>
<tr>
<td>60%</td>
<td>28.73%</td>
<td>28.57%</td>
<td>26.50%</td>
<td>30.77%</td>
<td>0.71%</td>
<td>0.07%</td>
<td>28.60%</td>
<td>28.87%</td>
</tr>
<tr>
<td>70%</td>
<td>26.03%</td>
<td>25.83%</td>
<td>23.93%</td>
<td>28.33%</td>
<td>0.81%</td>
<td>0.08%</td>
<td>25.87%</td>
<td>26.19%</td>
</tr>
<tr>
<td>80%</td>
<td>22.82%</td>
<td>22.69%</td>
<td>20.83%</td>
<td>25.64%</td>
<td>0.84%</td>
<td>0.08%</td>
<td>22.65%</td>
<td>22.98%</td>
</tr>
<tr>
<td>90%</td>
<td>19.67%</td>
<td>19.49%</td>
<td>18.33%</td>
<td>21.37%</td>
<td>0.62%</td>
<td>0.06%</td>
<td>19.55%</td>
<td>19.79%</td>
</tr>
<tr>
<td>100%</td>
<td>16.74%</td>
<td>16.67%</td>
<td>16.67%</td>
<td>17.09%</td>
<td>0.11%</td>
<td>0.01%</td>
<td>16.71%</td>
<td>16.76%</td>
</tr>
</tbody>
</table>

Table 4.9: Optimal height ratio for 30x30x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>σ</th>
<th>σ_{err}</th>
<th>Lw</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>74.27%</td>
<td>74.37%</td>
<td>70.39%</td>
<td>79.89%</td>
<td>1.79%</td>
<td>0.18%</td>
<td>73.92%</td>
<td>74.62%</td>
</tr>
<tr>
<td>10%</td>
<td>54.07%</td>
<td>53.89%</td>
<td>51.67%</td>
<td>56.11%</td>
<td>0.98%</td>
<td>0.10%</td>
<td>53.88%</td>
<td>54.27%</td>
</tr>
<tr>
<td>20%</td>
<td>45.06%</td>
<td>45.00%</td>
<td>42.78%</td>
<td>47.22%</td>
<td>0.79%</td>
<td>0.08%</td>
<td>44.91%</td>
<td>45.22%</td>
</tr>
<tr>
<td>30%</td>
<td>36.95%</td>
<td>36.67%</td>
<td>35.56%</td>
<td>38.89%</td>
<td>0.74%</td>
<td>0.07%</td>
<td>36.81%</td>
<td>37.10%</td>
</tr>
<tr>
<td>40%</td>
<td>32.98%</td>
<td>32.78%</td>
<td>32.22%</td>
<td>33.52%</td>
<td>0.28%</td>
<td>0.03%</td>
<td>32.92%</td>
<td>33.03%</td>
</tr>
<tr>
<td>50%</td>
<td>31.45%</td>
<td>31.67%</td>
<td>30.56%</td>
<td>32.22%</td>
<td>0.36%</td>
<td>0.04%</td>
<td>31.38%</td>
<td>31.52%</td>
</tr>
<tr>
<td>60%</td>
<td>29.44%</td>
<td>29.44%</td>
<td>27.93%</td>
<td>30.73%</td>
<td>0.55%</td>
<td>0.06%</td>
<td>29.34%</td>
<td>29.55%</td>
</tr>
<tr>
<td>70%</td>
<td>26.74%</td>
<td>26.67%</td>
<td>24.44%</td>
<td>28.89%</td>
<td>0.79%</td>
<td>0.08%</td>
<td>26.58%</td>
<td>26.89%</td>
</tr>
<tr>
<td>80%</td>
<td>23.28%</td>
<td>23.33%</td>
<td>20.56%</td>
<td>25.14%</td>
<td>0.82%</td>
<td>0.08%</td>
<td>23.12%</td>
<td>23.44%</td>
</tr>
<tr>
<td>90%</td>
<td>19.69%</td>
<td>19.44%</td>
<td>18.33%</td>
<td>21.11%</td>
<td>0.53%</td>
<td>0.05%</td>
<td>19.50%</td>
<td>19.79%</td>
</tr>
<tr>
<td>100%</td>
<td>16.68%</td>
<td>16.67%</td>
<td>16.67%</td>
<td>16.85%</td>
<td>0.04%</td>
<td>0.00%</td>
<td>16.67%</td>
<td>16.69%</td>
</tr>
</tbody>
</table>

Table 4.10: Optimal height ratio for 40x40x6 tables

<table>
<thead>
<tr>
<th>sf</th>
<th>Avg</th>
<th>Med</th>
<th>Min</th>
<th>Max</th>
<th>σ</th>
<th>σ_{err}</th>
<th>Lw</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>75.24%</td>
<td>75.24%</td>
<td>71.67%</td>
<td>78.33%</td>
<td>1.84%</td>
<td>1.51%</td>
<td>74.96%</td>
<td>75.51%</td>
</tr>
<tr>
<td>10%</td>
<td>54.80%</td>
<td>54.80%</td>
<td>52.92%</td>
<td>57.50%</td>
<td>0.94%</td>
<td>0.14%</td>
<td>54.61%</td>
<td>54.99%</td>
</tr>
<tr>
<td>20%</td>
<td>45.77%</td>
<td>45.77%</td>
<td>44.58%</td>
<td>47.08%</td>
<td>0.58%</td>
<td>0.10%</td>
<td>45.65%</td>
<td>45.89%</td>
</tr>
<tr>
<td>30%</td>
<td>37.16%</td>
<td>37.16%</td>
<td>35.83%</td>
<td>38.33%</td>
<td>0.58%</td>
<td>0.06%</td>
<td>37.04%</td>
<td>37.28%</td>
</tr>
<tr>
<td>40%</td>
<td>33.05%</td>
<td>33.05%</td>
<td>32.50%</td>
<td>33.47%</td>
<td>0.21%</td>
<td>0.06%</td>
<td>33.00%</td>
<td>33.09%</td>
</tr>
<tr>
<td>50%</td>
<td>31.77%</td>
<td>31.77%</td>
<td>30.83%</td>
<td>32.50%</td>
<td>0.34%</td>
<td>0.02%</td>
<td>31.70%</td>
<td>31.84%</td>
</tr>
<tr>
<td>60%</td>
<td>30.04%</td>
<td>30.04%</td>
<td>28.75%</td>
<td>31.67%</td>
<td>0.55%</td>
<td>0.04%</td>
<td>29.93%</td>
<td>30.15%</td>
</tr>
<tr>
<td>70%</td>
<td>27.36%</td>
<td>28.00%</td>
<td>25.42%</td>
<td>29.17%</td>
<td>0.71%</td>
<td>0.06%</td>
<td>27.22%</td>
<td>27.50%</td>
</tr>
<tr>
<td>80%</td>
<td>23.67%</td>
<td>23.67%</td>
<td>21.67%</td>
<td>26.25%</td>
<td>0.77%</td>
<td>0.07%</td>
<td>23.51%</td>
<td>23.83%</td>
</tr>
<tr>
<td>90%</td>
<td>19.73%</td>
<td>19.73%</td>
<td>18.33%</td>
<td>20.92%</td>
<td>0.49%</td>
<td>0.08%</td>
<td>19.63%</td>
<td>19.83%</td>
</tr>
<tr>
<td>100%</td>
<td>17.00%</td>
<td>17.00%</td>
<td>16.67%</td>
<td>16.74%</td>
<td>0.00%</td>
<td>0.05%</td>
<td>16.67%</td>
<td>16.67%</td>
</tr>
</tbody>
</table>
Table 4.11: Optimal vs maximum table height ratios

<table>
<thead>
<tr>
<th>sf</th>
<th>10x10</th>
<th>20x20</th>
<th>30x30</th>
<th>40x40</th>
<th>Avg</th>
<th>σ</th>
<th>Lw</th>
<th>Up</th>
</tr>
</thead>
<tbody>
<tr>
<td>0%</td>
<td>71.65%</td>
<td>73.25%</td>
<td>74.27%</td>
<td>75.24%</td>
<td>73.60%</td>
<td>1.53%</td>
<td>72.10%</td>
<td>75.10%</td>
</tr>
<tr>
<td>10%</td>
<td>52.81%</td>
<td>53.60%</td>
<td>54.07%</td>
<td>54.80%</td>
<td>53.82%</td>
<td>0.83%</td>
<td>53.01%</td>
<td>54.64%</td>
</tr>
<tr>
<td>20%</td>
<td>44.35%</td>
<td>44.53%</td>
<td>45.06%</td>
<td>45.77%</td>
<td>44.93%</td>
<td>0.64%</td>
<td>44.31%</td>
<td>45.55%</td>
</tr>
<tr>
<td>30%</td>
<td>37.79%</td>
<td>36.80%</td>
<td>36.95%</td>
<td>37.16%</td>
<td>37.18%</td>
<td>0.44%</td>
<td>36.75%</td>
<td>37.60%</td>
</tr>
<tr>
<td>40%</td>
<td>33.56%</td>
<td>32.80%</td>
<td>32.98%</td>
<td>33.05%</td>
<td>33.10%</td>
<td>0.33%</td>
<td>32.77%</td>
<td>33.42%</td>
</tr>
<tr>
<td>50%</td>
<td>30.92%</td>
<td>30.92%</td>
<td>31.45%</td>
<td>31.77%</td>
<td>31.26%</td>
<td>0.42%</td>
<td>30.85%</td>
<td>31.67%</td>
</tr>
<tr>
<td>60%</td>
<td>28.38%</td>
<td>28.73%</td>
<td>29.44%</td>
<td>30.04%</td>
<td>29.15%</td>
<td>0.74%</td>
<td>28.42%</td>
<td>29.88%</td>
</tr>
<tr>
<td>70%</td>
<td>25.69%</td>
<td>26.03%</td>
<td>26.74%</td>
<td>27.36%</td>
<td>26.45%</td>
<td>0.75%</td>
<td>25.72%</td>
<td>27.18%</td>
</tr>
<tr>
<td>80%</td>
<td>22.92%</td>
<td>22.82%</td>
<td>23.28%</td>
<td>23.67%</td>
<td>23.17%</td>
<td>0.39%</td>
<td>22.79%</td>
<td>23.55%</td>
</tr>
<tr>
<td>90%</td>
<td>20.53%</td>
<td>19.67%</td>
<td>19.69%</td>
<td>19.73%</td>
<td>19.90%</td>
<td>0.41%</td>
<td>19.50%</td>
<td>20.31%</td>
</tr>
<tr>
<td>100%</td>
<td>17.28%</td>
<td>16.74%</td>
<td>16.68%</td>
<td>17.00%</td>
<td>16.92%</td>
<td>0.27%</td>
<td>16.66%</td>
<td>17.19%</td>
</tr>
</tbody>
</table>

The overall ratio between optimal and maximum table heights is displayed in Table 4.11 where we show the average ratios for 10x10 to 40x40 tables with up to 6 words per cell and with a squeeze factor incrementing from 0% to 100%. The σ column represents the standard deviation while the Lw and Up columns show the confidence interval limits for a probability of 95%.

### 4.4.2 Comparative evaluation

In this section we report a comparative evaluation on the performance of the three optimization engines used in our tests: CPLEX, CP and LocalSolver engine. For CPLEX we use the TSC model presented in Section 4.1. CPI1 and CPI2 are the constraint programming models presented in Section 4.3 and LSP is the local search model presented in Section 4.2.

We include the second constraint programming model CPI2 in our evaluation because our tests showed that this model outperforms the CPI1 model for larger tables. However, as mentioned in Section 4.3, CPI2 does not reflect entirely the mathematical model of the table layout optimization problem; it only finds column widths and cell heights without selecting cell configurations.

The running time for all our tests is presented in Table 4.12. minW and maxW represent the minimum and maximum table width, computed from the cell-width configurations. For all experiments we set a maximum time limit of 300 seconds.
The times marked with indicate that the optimal value was found but the engine was unable to determine in the maximum time limit if this solution was optimal. The times marked with indicate that only a feasible solution close to optimal was determined within the maximum time limit.

For small tables all the engines performed comparatively well. When the tables increase in size most of our tests show that TSC was the best model and the CPLEX engine outperformed all the other engines. It is also the only model that, with very few exceptions, finds and stops when the optimal solution is found. The CP models, even when index decision variables are used (models CPI1 and CPI2) were in most of the cases the slowest. We had to evaluate the performance of the CPI2 model even if it does not reflect entirely the table layout optimization problem because CP and CPI1 models simply do not find any solutions when the tables were 50x50 or larger. The local search method was faster in some cases than the TSC and CP models to reach the optimal solution but, as expected, without stopping when the solution was found. For very large tables (over 100x100) local solver was the only engine that found at least a feasible solution while both CPLEX and CP failed to identify any solution. This shows that local search techniques can be used in very large cases of table layout optimization where proving optimality is not necessarily required.

We also identified a common characteristic of all the models: in at least one case some solutions were found but the engines were unable to reach the optimal solution.

In the remaining of this section we summarise our conclusions regarding the performance of each model/engine.

**Mixed Integer Programming (TSC model)**

- The fastest method/engine in most of the cases;
Table 4.12: Run time for TSC, CP and LSP models - Part 1

<table>
<thead>
<tr>
<th>Table</th>
<th>minW</th>
<th>+25%</th>
<th>+50%</th>
<th>+75%</th>
<th>maxW</th>
</tr>
</thead>
<tbody>
<tr>
<td>10x10x6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>586</td>
<td>920</td>
<td>1,254</td>
<td>1,587</td>
<td>1,921</td>
</tr>
<tr>
<td>height</td>
<td>450</td>
<td>250</td>
<td>190</td>
<td>150</td>
<td>100</td>
</tr>
<tr>
<td>time TSC</td>
<td>00:00.13</td>
<td>00:00.26</td>
<td>00:00.23</td>
<td>00:00.20</td>
<td>00:00.12</td>
</tr>
<tr>
<td>time CPI1</td>
<td>00:00:24</td>
<td>00:01:25</td>
<td>00:00:95</td>
<td>00:00:50</td>
<td>00:00:37</td>
</tr>
<tr>
<td>time CPI2</td>
<td>00:00:71</td>
<td>00:01:00</td>
<td>00:00:92</td>
<td>00:00:37</td>
<td>00:00:22</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>00:01:00</td>
<td>00:01:00</td>
<td>00:01:00</td>
<td>00:01:00</td>
</tr>
<tr>
<td>summary</td>
<td>m=586x590; M=1921x100; cnf=381;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp; ( \bigcup_j W_j \mid =154; \bigcup_i H_i \mid =59</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20x20x6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1,282</td>
<td>1,989</td>
<td>2,697</td>
<td>3,404</td>
<td>4,111</td>
</tr>
<tr>
<td>height</td>
<td>840</td>
<td>480</td>
<td>370</td>
<td>290</td>
<td>200</td>
</tr>
<tr>
<td>time TSC</td>
<td>00:00.37</td>
<td>00:00.89</td>
<td>00:01.20</td>
<td>00:01.25</td>
<td>00:00.30</td>
</tr>
<tr>
<td>time CPI1</td>
<td>00:02:29</td>
<td>00:04:40</td>
<td>00:03:07</td>
<td>00:02:09</td>
<td>00:01:92</td>
</tr>
<tr>
<td>time CPI2</td>
<td>00:05:10</td>
<td>01:02:00</td>
<td>00:01:20</td>
<td>00:02:19</td>
<td>00:00:60</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>00:01:00</td>
<td>00:01:00</td>
<td>02:00:00</td>
<td>00:01:00</td>
</tr>
<tr>
<td>summary</td>
<td>m=1,282x1,200; M=4,111x200; cnf=1404;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp; ( \bigcup_j W_j \mid =901; \bigcup_i H_i \mid =120</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>30x30x6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td>1,993</td>
<td>3,114</td>
<td>4,236</td>
<td>5,357</td>
<td>6,478</td>
</tr>
<tr>
<td>height</td>
<td>1,320</td>
<td>740</td>
<td>560</td>
<td>440</td>
<td>300</td>
</tr>
<tr>
<td>time TSC</td>
<td>00:00.82</td>
<td>00:07:49</td>
<td>00:08:72</td>
<td>00:02:42</td>
<td>00:00.50</td>
</tr>
<tr>
<td>time CPI1</td>
<td>00:10:63</td>
<td>00:24:00</td>
<td>00:22:00</td>
<td>00:52:00</td>
<td>00:10:89</td>
</tr>
<tr>
<td>time CPI2</td>
<td>00:13:00</td>
<td>05:10:00</td>
<td>01:09:00</td>
<td>01:07:00</td>
<td>00:01:65</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>01:01:00</td>
<td>00:07:00</td>
<td>00:02:00</td>
<td>00:01:00</td>
</tr>
<tr>
<td>summary</td>
<td>m=1,993x1,800; M=6,478x300; cnf=3,158;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&amp; ( \bigcup_j W_j \mid =1,806; \bigcup_i H_i \mid =180</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 4.13: Run time for TSC, CP and LSP models - Part 2

<table>
<thead>
<tr>
<th>Table</th>
<th>minW</th>
<th>+25%</th>
<th>+50%</th>
<th>+75%</th>
<th>maxW</th>
</tr>
</thead>
<tbody>
<tr>
<td>width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>40x40x6</td>
<td>2,721</td>
<td>4,279</td>
<td>5,836</td>
<td>7,394</td>
<td>8,951</td>
</tr>
<tr>
<td>height</td>
<td>1,810</td>
<td>970</td>
<td>770</td>
<td>590</td>
<td>400</td>
</tr>
<tr>
<td>time TSC</td>
<td>0:01:66</td>
<td>00:29.37</td>
<td>02:01.00</td>
<td>00:53.50</td>
<td>00:01.00</td>
</tr>
<tr>
<td>time CPI1</td>
<td>00:40.57</td>
<td>01:29.16</td>
<td>01:02.72</td>
<td>01:36.00</td>
<td>00:38.90</td>
</tr>
<tr>
<td>time CPI2</td>
<td>00:01.33</td>
<td>03:08.00</td>
<td>00:10.90</td>
<td>00:11.00</td>
<td>00:03.82</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>00:02.00</td>
<td>00:21.00</td>
<td>00:30.00</td>
<td>00:01.00</td>
</tr>
<tr>
<td>summary</td>
<td>m=2,721x2,400; M=8,951x400; cnf=5,629;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>50x50x6</td>
<td>3,474</td>
<td>5,473</td>
<td>7,473</td>
<td>9,472</td>
<td>11,471</td>
</tr>
<tr>
<td>height</td>
<td>2,310</td>
<td>1,270</td>
<td>970</td>
<td>800</td>
<td>500</td>
</tr>
<tr>
<td>time TSC</td>
<td>00:02.62</td>
<td>01:36.06</td>
<td>01:25.00</td>
<td>34:55.00</td>
<td>00:01.59</td>
</tr>
<tr>
<td>time CPI1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>time CPI2</td>
<td>01:02:00</td>
<td>02:53.00</td>
<td>01:12.00</td>
<td>02:30.00</td>
<td>00:07.37</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>01:33.00</td>
<td>00:26.00</td>
<td>00:07.00</td>
<td>00:01.00</td>
</tr>
<tr>
<td>summary</td>
<td>m=3,474x3,000; M=11,471x500; cnf=8,663;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>width</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>60x60x6</td>
<td>4,239</td>
<td>6,674</td>
<td>9,109</td>
<td>11,543</td>
<td>13,978</td>
</tr>
<tr>
<td>height</td>
<td>2,690</td>
<td>1,510</td>
<td>1,160</td>
<td>960</td>
<td>600</td>
</tr>
<tr>
<td>time TSC</td>
<td>00:04.52</td>
<td>05:37.61</td>
<td>02:00:00</td>
<td>01:22.00</td>
<td>00:02.13</td>
</tr>
<tr>
<td>time CPI1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>time CPI2</td>
<td>02:50.03</td>
<td>02:29.00</td>
<td>02:29.90</td>
<td>01:37.00</td>
<td>00:13.26</td>
</tr>
<tr>
<td>time LSP</td>
<td>-</td>
<td>00:24.00</td>
<td>00:11.00</td>
<td>00:21.00</td>
<td>00:02.00</td>
</tr>
<tr>
<td>summary</td>
<td>m=4,239x3,600; M=13,978x600; cnf=12,649;</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
• It finds optimal solutions and in most of the cases it stopped immediately after the optimal solution is found;

• There are cases where an optimal solution is found but it can take more than 1 hour to determine that the solution is optimal;

• For table of 100x100 or larger it could not determine any solution;

**Constraint Programming (CPI1 and CPI2)**

• It does not always stop after optimal solution is found. In fact, when the table size was increased to 20x20 or more the CP engine only stopped in the easier cases where the table width was not limited or it was limited to its minimum width;

• CPI1 was faster than CPI2 for small tables. This can be observed for the 20x20, 40x40 and especially 30x30 tables when it reached optimal solutions faster than CPI1 in the majority of the tests;

• CPI2 is slower for smaller cases but it outperforms CPI1 for larger cases. For a 50x50 and 60x60 tables CPI2 was able to find solutions while CPI1 reported that no feasible solutions can be found or it crashed;

• CPI1 model failed for tables 50x50 and higher. It has been observed that the engine uses the entire system memory available which makes the system to eventually come to an unexpected stop;

• If a period of 3-4 minutes elapses without finding improved solutions it has been observed that in most cases it will not find any other solutions even if the engine runs for multiple hours.
Local Search (LSP model)

- As this model is based on local search techniques it only stops if a time limit is set and it cannot guarantee that the solution found is optimal;

- LSP model is very fast to find an initial solution and then it improves on it. For a 50x50 table constrained to 75% of its width, it took only 7s for LSP to find a solution of 820 while TSC needed 34m55s to find only a marginally better solution of 800. The difference between the two solutions is the equivalent of 2 lines of text. For a 40x40 table constrained to take 25% of its maximum width, LSP reached the optimal solution in just 2s while TSC has found the same solution in 29.37s while both CP models failed to find this solution even after 1m29s and 3m8s respectively;

- For very large tables (over 100x100) local solver was the only engine that found at least a feasible solution for problem where the other engines failed;

- The LSP uses a different strategy where only the column widths are selected and minimum cell heights are pre-computed for each column width. We evaluated local search models that implement the MIP and TSC models strategies. The performance of these models was poor and hence a new model with a minimal number of decision variables was required. We also evaluated the LSP strategy with a reduced number of constraints as part of integer and constraint programming models. While it performed well for 10x10 tables, for larger tables it was very slow compared with the performance of the TSC or CP models;

- LSP can be faster than CP on some cases. For a 20x20 table LSP finds solutions faster than CP for both cases when the maximum table width set to be 25% and 75% of the maximum. For a 30x30 table with the width constraint to 75% of the maximum LSP reaches the optimal solution of 440 in 2s while
CPI1 requires 52s and CPI2 requires 1m7s. In this case even TSC takes longer 2.42s to find the optimal solution;

- LSP cannot find feasible solutions for table widths constrained to the minimum values;

- For large tables of 200x200 and over when the table width is not constraint at all, LSP requires a secondary objective to guide the search to the optimal solution. This is caused by the nature of the problem. For example, in a 200x200 instance where the page width is unlimited and the text size is set to 10 points, each cell can have a height of 10 or 20 or more. In order to find the optimal solution where all cells have a height of 10 LSP needs to decrease the height of 20 previously found as a feasible solution. Each change from 20 to 10 has no impact on the maximum function used to compute the row height until the height of all cells is decreased. Therefore, in the case when the width is not constrained, LSP has to find an optimal solution on a landscape with very large plateaus. One of the modelling solutions in order to guide the local search is to give a slight slope to these plateaus, for instance by adding a secondary optimization objective;

- LSP can handle multiple threads. If 4 threads are used (lsNbThreads=4) on a 4-core processor it takes longer to find a solution for a 40x40 table when the width was constraint to 75% of the maximum table width. A solution of 590 was found using 2 threads in 30s, and using 4 threads in 48s. It is important to note the recommendation that the number of threads should be at most \( k/2 \) with \( k \) the number of cores available. This can explain the slower running time when 4 threads are used on a 4-core processor;

- Similar to the CP engine, if no improved solutions are found after 2-3 minutes it can be expected that no other solutions will be found even after 1 hour or
longer and the engine could be stopped;

- LSP can be used in practical applications where the user only wants to find the best solution in the time that he or she is willing to wait.

## 4.5 Nested Tables

Table layout optimization is difficult because tables can contain any type of document objects: text, images, mathematical equations, and even other tables. For example, each cell in the bottom row of the table presented in Figure 1.5 contains inner tables which are displayed outside the body of the table. The difficulty is caused by the variable layout that some of these objects can have. In the previous sections we discussed the problem of tables that contain text. When a cell contains fixed size images or equations with a fixed size layout, the cell has only one configuration and therefore no additional complexities are induced. However, the complexity of the table layout optimization problem is further increased in the case of nested tables when the inner tables contain text. The layout of the inner tables is not fixed because each cell can have multiple configurations and any change in the configuration of a cell will impact the layout of all outer tables.

Based on the definition of the table layout problem presented in Section 3.1 all cell configurations need to be computed in a pre-processing step in order to solve the problem using the optimization models we presented in this thesis. Therefore, if a cell of an outer table contains an inner table the minimal configurations for the inner table must be pre-computed.

To minimize the layout of the 4x4 table displayed in Figure 4.10 all the minimal cell configurations have to be known. The difficulty arises when computing the configurations for cell 3, 3. This cannot be done in isolation as the inner table’s configurations are directly related to the configuration of the parent table. Minimizing the inner table’s height and using that configuration when minimizing the parent
Computing minimal cell configurations for the inner tables suggests a recursive method for finding the configurations with minimum table heights and widths. Starting from the most inner table all table configurations must be computed and then these configurations must be provided to the outer tables’ optimization models until a layout for the parent table is determined such that its height is minimized for the given page width. There are a number of problems that occur with such an approach:

- The optimization language and the optimization engine must allow recursive optimization flow control. While some of the engines, including IBM ILOG CPLEX, provide features for flow control this is not always as powerful as the high level programming languages (C++, Java, C#) for defining recursive functions. Therefore, an optimization model has to include a flow control function that iteratively simulates the mechanisms behind recursive calls by using traditional data structures, i.e. stacks and arrays that hold the status of the function. While this can be achieved the complexity of the optimization model is increased;

- A different problem occurs when trying to find all the minimal table configurations: finding for all possible table heights the minimum table widths. While an optimal minimum table height can be computed when the page width is
known in this case the page width is not known. This optimization method had to be repeated for all the possible table widths. When the process is complete only the minimal table configurations are selected. Because these table widths are given by the combination of all column widths the complexity of the problem is increased and therefore a long running time can be expected. Instead of finding minimal table heights for all the possible widths an alternative approach is to only find for all possible table heights the minimum table widths. The advantage of this method is given by the fact that the number of table heights is usually smaller than the number of table widths because it is most likely that the text in a table has the same font size. Therefore a smaller number of row heights is generated compared to the number of column widths because column widths depend on the length of the words in a cell;

- As cells can contain multiple types of document objects, if the cell contains both text (for example the table’s caption) and a table, finding all the configurations of the inner table is not sufficient. The cell’s configurations are a combination of both the text configurations and the table’s configurations;

- The complexity of the problems presented in the above points will require long computational times and therefore the method of recursively finding optimal table layouts can be inefficient;

The solutions to the table layout optimization problem presented in this thesis require that for each cell its configurations are pre-computed. For a cell that contains text its configurations are not difficult to determine but for a cell that contains an inner table, computing its configurations becomes difficult as we explained above. Therefore, we suggest as a possible solution merging the inner tables recursively in the parent table and finding a solution for this larger problem. Any change in the configuration selected for a cell in the parent table can potentially change the configuration of the inner table and therefore the configurations of the cells in the
inner table need to be re-computed. When the inner table is merged in the parent
table, all the cell configurations for both outer and inner tables are selected and
changed at the same time, as part of the same optimization process thus avoiding
long computational times required to re-compute inner table’s configurations.

To merge the inner tables all the cells will generate additional rows and columns
spanned by the surrounding cells of the outer table. For example, if an optimal
layout has to be found the table presented in Figure 4.10 by merging the inner table
in the parent table, only the optimal layout of a 6x6 table has to be determined.

This approach has the following advantages:

- there is no need for a recursive optimization model because there is only one
table (the parent table) that needs to be optimized and therefore only one
optimization pass is needed;

- Computing minimal table configurations for the inner tables is no longer re-
required. The algorithm that computes minimal text configurations is sufficient
in this case. All the cells in the inner tables are now part of the outer table
and the cell configurations are computed using the same algorithm, in a single
pass.

One of the downsides of this approach is that the merging process becomes even
more difficult when the inner table has related text i.e. a caption or footer notes.

4.6 Splitting tables

Another problem in automatic tabular layout that we believe has not been given
sufficient attention is splitting wide tables. We highlighted in Section 1.7 that most
of the popular authoring tools while dealing well with long tables do not provide a
function that would allow the author to split a wide table. Therefore, the author
faces a number of design problems: where to split the table so the meaning of
the data is not affected or the impact of splitting the table is minimized; where
to place the continuation headings, how to display the table identifier, the caption
and footnotes on subsequent pages or when should the column headers be repeated.
Wang [86] and Beach [12] also highlighted that these problems that occur when
displaying large tables require further investigation.

There are numerous style manuals and guides that document the best practice
for designing wide tables but the problems that occur when the author follows the
design guidelines are not well documented, studied or supported by the current
document authoring tools. For example, the 16th Edition of the Chicago Manual of
Style [69] indicates that:

“For a vertical table of more than one page, the column heads are re-
peated on each page. For a two-page broadside table – which should be
presented on facing pages if at all possible – column heads need not be
repeated; for broadside tables that run beyond two pages, column heads
are repeated only on each new verso. Where column heads are repeated,
the table number and “continued” should also appear. For any table that
is likely to run to more than one page, the editor should specify whether
continued lines and repeated column heads will be needed and where
footnotes should appear (usually at the end of the table as a whole).”

However, current document authoring tools have poor support to help the de-
signer to implement these guidelines. For example Figure 4.11 shows a large table
split across multiple pages. The table contains the results of the tests presented in
Section 4.4.1. It has 113 columns and 109 rows with 11 groups of 9 columns, each
group containing the results for each squeeze factor value. When printed, the first
row and the first two columns are repeated on each page in order for the user to
be able to understand the meaning of the test results. The start of each group is
visually identified by a column with a dark background. The table has been pro-
Reduced and printed using *Calc* from the *LibreOffice 3* suite. As it can be seen from this figure we were able to setup repeated horizontal and vertical headers and we can control where the pages are divided. We were unable to setup a continuation header & footer to indicate that the table is continued to and from another page. Also, we were unable to select repeated headers only on the left page when the table is printed on multiple facing pages.

In real life, when the author tries for example to present a wide table as a two-page broadside table there are cases when one or two columns require an additional
page. This is especially relevant when the table consists of many wide columns or and the columns became wider following a table height minimization process. Therefore the author needs a strategy to use the space available on each page more efficiently. The options available in this case are to narrow or reorder the columns so the space is used more efficiently but both alternatives have disadvantages. Narrowing the columns may not always be possible as the columns could have fixed widths or the column widths are already set to the minimum width required by the content. Narrowing columns can also increase the table height. This in turn will cause the bottom rows to be displayed on additional pages. Also more space will be required on these additional pages for displaying the repeated table header, the continuation messages (i.e. “continued on next page”, “continued from previous page”), and possibly additional rows for partial sums or overall totals.

The second option available to the author is to reorder the columns with the condition that the meaning and readability of the table is not affected. This is especially relevant when the pages used by the table layout have empty spaces due to the allocation of columns in pages. For example, when a table with two wide consecutive columns is split across pages it will leave a large amount of empty space on each page while the remaining columns require additional pages. By reordering the columns, the space on each page can be used more efficiently and, as a result, fewer pages are necessary.

Using fewer pages to display a table is not only advantageous from an economical point of view in the printing process. It increases the readability of the table and it facilitates data editing. The reader can have a global view of the table, follow the logical relationships between the data items by rows and columns and compare the values of the data items without repeatedly changing pages. The editing process is also made easier when fewer pages are involved. For example, when adding a row or when reordering rows in a wide table displayed on multiple pages, there are fewer chances of errors to be introduced, i.e. when the changes in the first part of the
table are not replicated across all the pages that the table is displayed on.

Reordering of columns raises a number of problems that we will study in this section. The problem of finding an arrangement of columns that minimizes the number of pages used is presented in Section 4.6.1. We present a simple but effective integer programming method for finding solutions to this problem. We show examples where for a table with 50 columns the number of A4 pages used have been reduced from 34 to 26 which represents a reduction of 23.52%.

In many cases, and even more relevant for larger tables, reordering columns freely is not always possible because it impacts on the readers ability to understand the data. This is why we suggest in Section 4.6.2 a solution to the problem of minimizing the number of pages and at the same time the number of changes in the column positions. There may be many arrangements of columns that minimize the number of pages used but finding that arrangement that has the least impact in the meaning of the data is preferred. We show that this problem is not easy to solve and our experiments show that the problem requires long computational times.

A third problem that we discuss in Section 4.6.3 is the column reordering problem with the constraint that some columns should be kept together in as much as possible when tables are split. In many cases the author can suggest which columns must be kept together on the same page. Given that the primary objective is to minimize the number of pages the constraint to keep certain columns together cannot always be completely satisfied as this can have the opposite effect of increasing the number of pages. Therefore we present a solution where the number of pages is minimized and at the same time the number of groups split in this process is also minimized.

We conclude each of these sections with the performance evaluation of the proposed solutions. The results presented in all the evaluations have been obtained by using IBM ILOG OPL 6.3 with CPLEX engine 12.3 running on a 64-bit Linux laptop with an Intel i7-2640M CPU at 2.80GHz and 8Gb RAM.
Listing 4.10: TSMin - Table split with minimum page count

```plaintext
ingt pageW = . . . ; // page width
int NbCols = . . . ; // column count
range Cols = 1 .. NbCols; // column range
range Pages = 1 .. NbCols; // page range; there are maximum NbCols pages
int colW[Cols] = . . . ; // column widths

dvar int+ pageSel[Pages] in 0 .. 1; // contains 1 if a page is needed
dvar int+ X[Pages][Cols] in 0 .. 1; // contains 1 if page p contains column j

dexpr int pageCount = sum(p in Pages) pageSel[p];

subject to {
  ct1 : // each column must be selected only once
    forall(j in Cols)
      sum(p in Pages) X[p][j] == 1;
  ct2 : // do not exceed page width
    forall(p in Pages)
      sum(j in Cols) colW[j] / pageW * X[p][j] <= pageSel[p];
}
```

4.6.1 Minimize Page Count

To minimize the number of pages used when a wide table is split across multiple pages we model the problem as a mixed integer programming model. This is a standard bin packing problem that can be simply modelled as in Listing 4.10. To improve the clarity of the code we defined the Pages range to be the same as the Cols range because there are maximum NbCols pages needed. The colW[Cols] stores the width values for each column.

The objective of this model is to determine values for the arrays of binary variables pageSel[Pages] and X[Pages][Cols] such that the number of pages used is minimized. The array X[Pages][Cols] will only contain the value 1 if column j in Pages is selected on page p in Pages.

Constraint ct1 ensures that each column is only selected once while constraint ct2 guarantees that the sum of column widths selected in a page pageSel[p] does not exceed the page width.

In Table 4.14 we show the running time and the optimum page count (OPC) for tables with up to 60 columns. The column widths were randomly generated and the
Table 4.14: Run time for the Table Split model

<table>
<thead>
<tr>
<th>Columns</th>
<th>10</th>
<th>20</th>
<th>30</th>
<th>40</th>
<th>50</th>
<th>60</th>
</tr>
</thead>
<tbody>
<tr>
<td>PC</td>
<td>7</td>
<td>16</td>
<td>19</td>
<td>29</td>
<td>34</td>
<td>48</td>
</tr>
<tr>
<td>OPC</td>
<td>6</td>
<td>12</td>
<td>15</td>
<td>23</td>
<td>26</td>
<td>39</td>
</tr>
<tr>
<td>%Imp</td>
<td>14.28</td>
<td>25.00</td>
<td>21.05</td>
<td>20.68</td>
<td>23.52</td>
<td>18.75</td>
</tr>
<tr>
<td>Time</td>
<td>00:02.25</td>
<td>00:00.13</td>
<td>00:00.17</td>
<td>00:01.18</td>
<td>04:30.12</td>
<td>00:01.52</td>
</tr>
</tbody>
</table>

We can see that by using this integer programming model the number of pages PC can be reduced with between 14.28% for a small table with 10 columns and 23.52% for a table with 50 columns. As in our previous tests, we found that the difficulty of the problem is not directly linked to the problem size but to the data itself. For example, in the case of the table width 50 columns it can be seen that the minimum number of pages has been determined in 4m30s which is a lot more than the larger case of a table with 60 columns.

4.6.2 Minimize Page Count and Column Position Changes

In order to use the page space more efficiently and thus reduce the number of pages used when a wide table is split across multiple pages the author can decide to reorder the columns. Using the method presented in the previous section we can determine the ordering of columns that will minimize the number of pages. Usually, there are many ways of ordering the columns that will allow the table to be displayed on this minimum number of pages.

For example, a table with the following column widths will require 7 A4 pages to be displayed when the page width is set to 490 points:

\[
\text{colW} : [210, 140, 210, 420, 280, 350, 70, 140, 140, 350]
\]

\[
\text{pages} : \{210, 140\}, \{210\}, \{420\}, \{280\}, \{350, 70\}, \{140, 140\}, \{350\}.
\]
By reordering the columns with the positions shown in \texttt{colIdx} only 5 pages are required:

\[ \text{colIdx} : [1, 7, 8, 5, 2, 9, 6, 10, 3, 4] \]

\[ \text{pages} : \{210, 280\}, \{140, 350\}, \{420, 70\}, \{140, 210\}, \{350, 140\}. \]

It can easily be observed that the original order of columns is substantially changed. The second column is placed in position 7, the 3rd in position 8, the 4th in position 5 and the fifth column in position 2, etc. but the author is interested to find that ordering that has the least impact on the meaning of the data. An alternative ordering of columns that still requires only 5 pages to be used when splitting the table is presented below:

\[ \text{colIdx} : [1, 2, 3, 5, 4, 7, 6, 8, 9, 10] \]

\[ \text{pages} : \{210, 140\}, \{210, 280\}, \{420, 70\}, \{350, 140\}, \{140, 350\}. \]

This example shows that simply by swapping the positions of two sets of columns, i.e. column 4 with 5 and column 6 with 7 the number of pages can still be reduced to 5 but only with a few changes. Therefore the meaning of the data is less likely to be affected in this case compared with the previous ordering of columns.

To measure how many changes an ordering has compared to the original order of columns we introduce the \texttt{posDiff} variable, given by

\[
\text{posDiff} = \sum_{j_1 < j_2} |\text{posO}(j_1, j_2) - \text{posN}(j_1, j_2)|
\]

(4.10)

The \texttt{posO}(j_1, j_2) function returns 1 if column \( j_1 \) is placed before \( j_2 \) and 0 otherwise. Because in the initial ordering of columns the position of column \( j \) is in fact \( j \) the function \texttt{posO} returns 1 when \( j_1 < j_2 \) and 0 otherwise. \texttt{posO} abd \texttt{posN} are
given by:

\[
\text{posO}(j_1, j_2) = \begin{cases} 
1 & j_1 < j_2, \\
0 & j_1 \geq j_2
\end{cases}, \quad 1 \leq j_1 < j_2 \leq n 
\tag{4.11}
\]

\[
\text{posN}(j_1, j_2) = \begin{cases} 
1 & P[j_1] < P[j_2], \\
0 & P[j_1] \geq P[j_2]
\end{cases}, \quad 1 \leq j_1, j_2 \leq n 
\tag{4.12}
\]

The \(\text{posN}(j_1, j_2)\) function returns 1 if the new position of a column \(P[j_1]\) is before the position of a second column \(P[j_2]\), and 0 otherwise. Thus, \(\text{posDiff}\) counts how many times the new position \(P[j]\) of any column \(j\) has changed in relation to any another column.

We model the problem of minimizing the number of pages used to split a wide table while minimizing relative column positioning changes as a multi-objective mixed integer programming model presented in Listing 4.11. We initialize \(\text{colP}[\text{Cols}]\) with the page index for each column. The initialisation code has been omitted for space reasons but the full code is available in Appendix A. The variables \(a, b\) and \(\text{obj1Val}\) are used for the multi-objective flow control of the optimization which we will describe later in this section.

The objective of this model is to determine values for the page selection variable \(\text{pageSel}[\text{Pages}]\), new column index \(\text{colIdx}[\text{Cols}]\) which indicates the new position of columns and the new page index for each column \(\text{pageIdx}[\text{Cols}]\) such that the number of pages \(\text{pageCount}\) and the number of column changes \(\text{posDiff}\) is minimized. The decision expressions \(\text{posO}[j_1, j_2], \text{posN}[j_1, j_2]\) and \(\text{posDiff}\) are expressed as in equations 4.10, 4.11 and 4.12.

The model has four constraints: \(\text{ct1}\) ensures that the total width of the columns assigned in a page does not exceed the page width. It also selects a page index \(p\) in the array of binary variables \(\text{pageSel}[p]\); the \(\text{ct2}\) constraint ensures that if a column \(j_1\) is selected on a previous page to column \(j_2\) then its new index \(\text{colIdx}[j_1]\) is less then the new index of column \(j_2\). Constraint \(\text{ct3}\) makes sure that unique values
Listing 4.11: TSMinCol - Table split with minimum page count and column position

```plaintext
1  int pageW = ...; // page width
2  int NbCols = ...; // column count
3  range Cols = 1...NbCols; // column range
4  range Pages = 1...NbCols; // page range; there are maximum NbCols pages
5  int colW[Cols] = ...; // column widths
6  int colP[j in Cols]; // the page index for each column j
7  ...
8  int a = ...; // flow control: initialized to 1
9  int b = ...; // flow control: initialized to 0
10  int obj1Val = ...; // flow control: initialized to -1
11  
12  dvar int+ pageSel[Pages] in 0..1; // contains 1 if a page is needed
13  dvar int+ colIdx[Cols] in Cols; // new column index
14  dvar int+ pageIdx[Cols] in Cols; // new page index for each column
15  
16  // check if column j1 is placed on a page before column j2
17  dexpr int posO[j1 in Cols, j2 in Cols] = j1 < j2 - 1;
18  dexpr int posN[j1 in Cols, j2 in Cols] = (colIdx[j1] <= colIdx[j2] - 1);
19  dexpr float posDiff = sum(j1, j2 in Cols: j2 < j1) abs(posO[j1, j2] - posN[j1, j2]);
20  dexpr int pageCount = sum(i in Pages) pageSel[i];
21  
22  minimize a * pageCount + b * posDiff;
23  
24  subject to {
25   ct1: // do not exceed page width
26      forall(p in Pages)
27         sum(j in Cols) (colW[j] * (p == pageIdx[j]) / pageW) <= pageSel[p];
28    ct2: // maintain relationship between page index and column index
29      forall(ordered j1, j2 in Cols)
30         (pageIdx[j1] <= pageIdx[j2] - 1) - (colIdx[j1] <= colIdx[j2] - 1) == 0;
31  ct3: // new column index must have unique values
32      forall(ordered j1, j2 in Cols)
33         colIdx[j1] != colIdx[j2];
34    }
35  
36  // if the minimum page count obj1Val has has been determined
37  // maintain this value for subsequent searches
38  ct4: if (obj1Val > 0) pageCount == obj1Val;
39  }
```

123
are assigned for each column index.

The role of constraint ct4 is to guarantee that the minimum number of pages found in a first pass of optimization is maintained in a second pass of the optimization flow which seeks to minimize the number of column position changes. The flow of the optimization is controlled by using OPL script as presented in Listing 4.12. As variable \( a \) is initialized with 1, \( b \) with 0 and \( \text{obj1Val} \) with -1 the first pass of the optimization will minimize the number of pages. The second objective, minimizing the number of column changes is then reached in a second pass by setting the \( a \) variable to 0, \( b \) to 1 and \( \text{obj1Val} \) with the minimum number of pages just found.

If this model is implemented using a different optimization engine which does not allow flow control the objective function can be defined as:

\[
\text{minimize } m \cdot \text{pageCount} + \text{posDiff}
\]

with

\[
m > \max \text{posDiff}.
\]

In this model \( m \) can be set to \( NbCols^2 \) because this is the maximum number of relative column position changes. Any change of \( \text{pageCount} \) value has a higher
impact on the overall solution than any change in the value of $posDiff$ and therefore an optimal solution can be found in only one optimization phase.

In our tests with column widths generated with random values between 20 and 450 and the page width set to 450 points we found that for a 10 column table that required 9 pages to be displayed it took 2.25 seconds to find an ordering of columns that minimized the number of pages to 8 and the number of column position changes to 4. By only optimizing the number of pages to 8 without minimizing column position changes the table would have had a $posDiff$ of 33 changes so the advantage of this method can be easily observed. This has also been confirmed for a larger table with 20 columns that initially required 13 A4 pages when split. We determined an arrangement of columns that needed only 11 pages but it had a $posDiff$ of 194. Within 1m29.68s an arrangement of columns was determined that had a $posDiff$ of only 4 while the minimum page count was maintained at 11.

The problem proved to be more difficult to solve as the number of column increased but, as found in the evaluation of the page count minimization, the difficulty of the problem depends on test data, case by case, and not solely on the problem size.

4.6.3 Minimize Page Count and Group Splitting

When wide tables have to be split across pages the order of columns needs to be changed for a more efficient use of the page space and to reduce the number of pages used. As we previously explained, tables can have a number of column orderings that minimize the number of pages and users should be allowed to indicate which ordering is preferred so the effect of the column re-ordering is minimized on the meaning of the data.

In this section we discuss the problem of minimizing page count when splitting tables across pages while minimizing the number of group splits. In practical cases, the user needs to specify which columns should preferably be kept together when a
new ordering is determined. Here, the term preferably is essential because if columns must be kept together there are a number of simple solutions that can be used in this case (for example, a column group could be treated as one single wider column).

If $C$ is the set of columns we define a group $C_g$ as a subset of $C$

$$C_g \subseteq C, \quad C_g \cap C_g' = \emptyset$$

and $\mathcal{G}$ is the set of column groups:

$$\mathcal{G} = \{C_g\}.$$

When the user indicates which columns should preferably be kept together the problem is to find that ordering of columns that minimizes page count and at the same time it minimizes the number of groups that are split across pages.

For the same example as in the previous section, a table with the following column widths will require 7 A4 pages to be displayed when the page width is set to 490 points:

$$colW : [210, 140, 210, 420, 280, 350, 70, 140, 140, 350]$$

$$pages : \{210, 140\}, \{210\}, \{420\}, \{280\}, \{350, 70\}, \{140, 140\}, \{350\}$$

A possible reordering of the columns requires only 5 pages:

$$colIdx : [3, 5, 4, 7, 10, 6, 8, 1, 2, 9]$$

$$pages : \{210, 280\}, \{420\}, \{70, 350\}, \{350, 140\}, \{210, 140, 140\}$$

When the user requires that columns 2,3 and 7 preferably be kept together the ordering above is not satisfactory because column 7 is far from column 2 which is
far from column 3. Therefore a better ordering is presented below:

$$colIdx: [2, 3, 7, 4, 9, 10, 6, 8, 1, 5]$$

$$pages: \{140, 210, 70\}, \{420\}, \{140, 350\}, \{350, 140\}, \{210, 280\}$$

If, for a table we have $|\mathcal{G}|$ number of groups, for each column group $g$ with $1 \leq g \leq |\mathcal{G}|$ we have $C_g$ a set of column indices of the form $\{j \mid j \in C_g\}$ which records which columns $j$ belong to the group index $g$. To count how many groups are split across pages we need to introduce an array $F[g]$ which specifies for each group index $g$ the index $j$ of the first column that belongs to that group. The problem is to determine for each column $j$ the page index $P[j]$ that minimizes the number of pages. We will count in $S[g]$ how many columns $j$ of $C_g$ have a different page index $P[j]$ than the first column’s page index $P[F[g]]$.

$$S[g] = \sum_{j \in C_g} \delta(g, j)$$

$$\delta(g, j) = \begin{cases} 
1 & P[j] \neq P[F[g]] \\
0 & P[j] = P[F[g]] 
\end{cases}, \quad 1 \leq g \leq |\mathcal{G}|$$

If $\gamma(g)$ which takes the value 1 if a group $g$ is split across multiple pages is defined as

$$\gamma(g) = \begin{cases} 
1 & S[g] > 0 \\
0 & S[g] = 0 
\end{cases}, \quad 1 \leq g \leq |\mathcal{G}|$$

the number $\Gamma$ of groups split across pages is defined as in equation 4.13

$$\Gamma = \sum_{g=1}^{\mathcal{G}} \gamma(g) \quad (4.13)$$

We model this problem as a mixed integer programming problem TSMinGroup with two objectives as presented in Listing 4.13. We introduce a new array $\text{colG}[\text{Cols}]$.
which contains for each column \( j \) the group index. If the user does not specify any column groupings \( \text{colG} \) there is only one group with a 0 index. On line 8 we identify the set of groups and on line 9 we store in \( \text{gFirstCol}[g \text{ in groups}] \) array the first column of each group \( g \).

This optimization model finds values for an array of binary variables \( \text{pageSel}[\text{Pages}] \) and an array \( \text{pageIdx}[\text{Cols}] \) that stores the page index for each column such that the page count and the number of groups split across pages \( g\text{SplitCount} \) are minimized.

The decision expression \( g\text{Split}[g \text{ in groups}] \) counts how many columns are on a different page than the group’s first column and the expression \( g\text{SplitCount} \) counts how many groups are split across pages.

This model has only two constraints: \( \text{ct1} \) ensures that the sum of column widths is less than the page width. Constraint \( \text{ct2} \) is used to maintain the minimum number of pages determined in the first optimization phase during the second optimization phase when the engine seeks the minimum number of groups split across pages.

The flow control procedure is the same as in Listing 4.12.

In our tests we identified cases where TSMinGroup found suitable solutions in a short time. For the case shown below where 7 pages are required to display a table with 10 columns TSMinGroup must determine a solution where the number of pages are minimized and two groups of columns 1,3,8,9 and columns 5 and 7 which must to be kept together. Given \( \text{colW} \) as

\[
\text{colW} = [130, 414, 79, 331, 129, 190, 284, 413, 120, 96]
\]

a possible arrangement of columns that minimizes the number of pages is:

\[
\{130, 190, 96\}, \{129, 120\}, \{79, 284\}, \{413\}, \{414\}, \{331\}.
\]

This arrangement does not take into account the need to keep together the columns of the two groups. Using TsMinGroup a final solution is found in just 0.33
Listing 4.13: TSMinGroup - Table split with minimum page count and group splits

```plaintext
int pageW = ...; // page width
int NbCols = ...; // column count
range Cols = 1..NbCols; // column range
range Pages = 1..NbCols; // page range; there are maximum NbCols pages
int colW[Cols] = ...; // column widths
int colG[Cols] = ...; // column groups

{int} groups = {colG[j] | j in Cols}; // the groups set
int gFirstCol[g in groups] = first({j | j in Cols : colG[j] == g});

dvar int+ pageSel[Pages] in 0..1; // contains 1 if a page is needed
dvar int+ pageIdx[Cols] in Cols; // new page index for each column

// counts how many columns of a group are on a different page than the
// first group’s column
dexpr int gSplit[g in groups] = sum(j in Cols: colG[j] == g)
    (pageIdx[j] != pageIdx[gFirstCol[g]]);

dexpr int gSplitCount = sum(g in groups) (gSplit[g] >= 1);
dexpr int pageCount = sum(p in Pages) pageSel[p];

minimize a * pageCount + b * gSplitCount;

subject to {
    ct1: // do not exceed page width
        forall(p in Pages)
            sum(j in Cols) (colW[j] * (p == pageIdx[j]) / pageW) <= pageSel[p];
    ct2: if (objVal >= 0) pageCount == objVal;
}
```

129
of a second where no groups are split:

\[
\text{pages} : \{130, 79, 120, 96\}, \{129, 284\}, \{413\}, \{414\}, \{331\}, \{190\}
\]

As the number of columns increases the time required to find suitable solutions for similar problems also increases. For a table with 20 columns which has 3 column groups \{2,4\},\{5,10\},\{17,20\}:

\[
\text{colW} = [186, 80, 203, 74, 200, 152, 429, 191, 50, 225, 53, 393, 20, 94, 251, 290, 147, 158, 413, 280]
\]

a solution that keeps the columns together while minimizing page count to 9 pages instead of 12 was found in 1 minute:

\[
\text{pages} : \{80, 74, 290\}, \{147, 280\}, \{200, 225\}, \{429\}, \{186, 152, 94\}, \{203, 53, 158\}, \{20, 413\}, \{50, 393\}, \{191, 251\}
\]

We also identified cases for table with 30 and 40 columns but the time required to find solutions increased from under 2 minutes to over 12 minutes in some larger instances.

It is important to note that some column groupings can make the problem difficult to solve and therefore finding a solution may become time consuming. In practice, there may be cases where the table author could use the solution found to a more relaxed problem (with fewer column groups) to adjust the ordering of columns in a shorter time than the time required to automatically find such a solution.
4.6.4 Conclusions

In this section we discussed the problem of splitting wide tables across pages. We identified a number of problems that occur in practice and we provided a number of solutions based on combinatorial optimization methods. We defined the problems of minimizing the number of pages used and at the same time the number of relative column positions changes and the problem of minimizing page count while keeping together groups of columns as integer programming problems. Following their evaluation we can draw the following conclusions:

- splitting wide tables in a manner that minimizes the impact on the meaning of the data raises a number of computationally hard problems that require additional research;

- finding an optimal arrangement of columns such that the page count is minimized when splitting wide tables can be achieved in relatively short running time; for tables with 60 columns a solution has been found in less than 2s;

- if additional criteria are added, for example minimizing the number of relative column positions changes, the problems become harder as the number of columns increase; for the problem where the number of split column groups must be minimized the problem becomes harder as the number of groups increases;

- the difficulty of the problems does not only depend on the problem size but on the complexity of the data. We identified cases where solutions for larger instances are found faster than tables with less columns.

- additional criteria can be easily modelled to resolve other preferred constraints, i.e. find the ordering of columns that require an even number of pages so the table can be displayed as a series of two page broadside tables.
Chapter 5

Prototype of a Table Drawing Tool

In this chapter we introduce a Table Drawing Tool prototype that in addition to being useful in its own right can be used to visually inspect the output of the optimization models presented in this thesis. Through its API, users can control standard table formatting features such as dimensions, background and text colours, font style and size, spanning cells, borders width, rotated text and also features that are not always available in WYSIWYG tools: cell padding, margins, inner tables, equations and especially table layout optimization. We discuss the challenges to overcome when developing table authoring tools and we review the typographical issues which we had to solve in the development of this prototype: parameters propagation in a hierarchical document structure, line styles and line intersection for drawing cells or inner tables, table layout elements, etc.

A table is usually represented as a two dimensional rectangular grid with rectangular cells arranged in rows and columns in order to show the logical relationships between data items. In Figure 5.1 we show an example of a table that has an irregular shape with one header at the bottom of the table and data items rotated to a fixed angle. This example shows that other table shapes and cell arrangements can be used to represent logical relationships in a more meaningful way. Another example is a circular table displayed in Figure 5.2 without rows and columns where
the cells have an irregular shape. The information in this table can also be displayed
by using a table with a rectangular layout as shown in Figure 5.3. Wang also
highlights that tabular information can be displayed using different presentation
methods such as bar graphics, line graphics, pie charts, etc.

It is obvious that a new range of typographical problems need to be addressed in
each case depending on the type of layout chosen to display the data. To describe
a rectangular table it is enough to record or determine the four coordinates top-left,
top-right, bottom-right, bottom-left. For the table in Figure 5.1 we have to ensure
that for the last entry in the header (Athlone) a line is drawn from the top-right
corner of the cell to the bottom-right corner of the data item in order to aesthetically
close the header row. For a circular table, the shape of each cell is determined by
two arcs connected at both ends with straight lines. The length of each arc depends
on the angle at which the cell is placed relative to a reference point.

Sometimes, as Wright stated, diagrams, graphs or flowcharts can be more
appropriate to display the data in order to convey its meaning and to allow the user
to answer a wider range of questions related to the data.

The Table Drawing Tool – TDT – has been developed as an API with the
5.1 Introduction

Table Drawing Tool prototype – TDT – has been developed at University of Limerick as part of a larger project Table Tool System – TTS –, a tool for tabular expressions. This tool uses the logical table model introduced by Parnas where tables are defined as an indexed set of grids and a grid is an indexed set of expressions as we explained in Section 2.1. Tabular expressions are well understood mathematical expressions that can be evaluated and validated. This ensures that tabular expressions are valid. For example, the data values in the header are unique or data ranges are completely and correctly defined. Tabular expressions have been used in practice to document
software systems with a high level of accuracy in order to improve quality and reduce development costs as shown by Alspaugh et al. [5]. The TTS tool consists of three independent modules:

- **Table Input Tool** that deals with problems related to data input. This tool allows input of both tabular expressions and conventional expressions. Editing of tabular expressions allows the creation of grids while the conventional expressions, created with a variety of input tools, are placed in cells within the grids [71].

- **Table Kernel Module** which handles the logical model of tables, is a library that all tools will use to perform common tasks such as storage (in both volatile and persistent memory) and evaluation. In the TTS, the kernel provides facilities for storing, retrieving, evaluating and checking tabular expressions. There are also facilities for defining new types of tables [10].

- **Table Output (Drawing) Tool - TDT** which is responsible for drawing tables.

  This clear separation of concerns allows any module of the table tool to be independently modified without affecting or relying on the functionality of the other modules.

  TDT is concerned only with drawing tables. It is not aware of the data input methods or the logical table model nor it should be. It takes input data in XML format and it draws tables to a rendering device depending on the selected table layout with the help of a table layout optimizer. Thus, we can define any table layout, that can be drawn on any rendering device and apply any table layout optimization technique as long as a predefined set of input/output functions are provided.

  TDT currently draws rectangular layout tables on a PDF device. The layout is optimized by an IBM ILOG CPLEX OPL optimizer which finds the minimum height layout for a given page width.
5.2 Why a Hierarchical Model

Beach acknowledged that “Table entries can be arbitrary document objects such as text, illustrations, mathematical expressions or other tables” [12]. Therefore a table layout tool needs to be as powerful as a page layout tool.

To deal with different types of document objects TDT implements an abstract object called container that can store other container objects. This creates a hierarchical structure that is essential in TDT. A container is said to be the parent to its inner containers which are called child containers. All the children of a parent container are called siblings. The root container holds document wide settings such as custom lines styles, numbering styles, table caption styles and other global settings.

This model allows:

hierarchical propagation of typographical parameters. The value of an inherited parameter is given by the value of the same parameter assigned to its parent container. Line style, background colour and pattern, font size, etc. can be set at lower level in the hierarchy and then inherited by the containers placed at a higher level. For tables, some parameters can be set at the table level and then all the cells inherit the settings unless they are set specifically for
Figure 5.5: Layers in a rectangular table provide control over the parameter’s priority that cell. In Figure 5.4 if box A contains another box B that has no value provided for the colour parameter, the value will be inherited from the colour parameter of box A.

layered control of typographical parameters. When objects overlap the user needs to control what part of an object is visible. In the case of a table with a rectangular layout, where settings are applied at table, row, column or cell level the user needs to prioritize which settings will take priority. This can be achieved by grouping typographical parameters in layers and simply changing their order. In Figure 5.5 row settings are prioritised over column settings which are then prioritised over table settings. In CSS specifications rows are prioritised over columns [81] but we believe that this option should be provided to the user and not imposed by the system.

In this hierarchical model, it is each container’s responsibility to format itself; each container must implement two functions: computeLayout and render on device. For example, rectangular tables implement a layout algorithm that organizes cells in rows and columns, uses the global line settings, caption settings and numbering styles. Their render function uses formatting parameters such as colour, font family and size, etc. to render the content on a device. As each device has its own
The box element defines margins, borders, padding and content spaces. implemention of colours, line styles and other graphical elements its main role is to provide functions that translate TDT’s graphical elements and to convert units from TDT’s coordinate system in the coordinate system implemented by the device. We present the properties of the device object in Section 5.4.

5.3 Tabular Layout in TDT

For drawing tables, TDT implements a box model similar to the CSS model where margin, border, padding settings can be set for each top, bottom, left and right sides as displayed in Figure 5.6.

A box object has four spaces: the margin space, border space, padding space and the content space divided on the fours sides of the box, top, right, bottom and left. The margin is the transparent space used to separate two adjacent boxes. The border space is usually filled with a pattern and creates a visual separation between the content of adjacent boxes. The padding space introduces a space between the border and the content space. The content space is the rectangular area that will be used to display the content of the box.

Many prototypes for table authoring tools have been developed throughout the years: TABPRINT, a special program running on an IBM 709/90 computer in 1962.
as mentioned by Bartnett [11], Tbl in 1979, the prototype developed by Beach in 1985 [12], and the prototype developed by Vanoirbeek in 1989 [55] just to mention a few. Only recently WYSIWYG tools, browsers and specialised Desktop Publishing software provide a similar level of typographical detail as provided by TDT. As we show in Section 1.7 not all the WYSIWYG tools allow the control of margins and padding, and the control of borders needs to be improved.

While allowing an increased level of control, this flexible model creates a number of problems: how TDT aligns boxes in a normal flow, how it deals with margins and borders in a grid layout, how to control lines style, parameters inheritance, etc. We discuss some of these problems in the following sections.

5.3.1 Laying Containers: Fixed and Floating Positioning

As table entries can contain any type and number of document objects a drawing algorithm needs to determine how to layout these objects. Knuth [51] documented the problem in detail and his research unearthed subsequent difficult problems such as paragraphing. The main idea implemented in TDT is to lay floating (non-fixed) containers on lines without exceeding a maximum width given by the available space. These containers are then vertically aligned, as specified by the user, on the top, centre, bottom or on the text base line.

The complexity of the problem is increased when some containers have fixed positions or fixed dimensions and other containers have to be arranged in the available space. This makes harder the aesthetical arrangement of floating containers because these containers have to be layed-out in such a way that the empty space around fixed containers is minimized. Marriott et al. [59] discuss in detail the problem of laying out floating containers in multi-column documents.

Figure 5.7 shows how TDT determines the arrangement of containers. Because the dimensions of these containers have been fixed by the user, the first container is wider than its content and the text is wrapped in the second and in the last container.
All containers are aligned to top. Figure 5.8 shows how the same containers are arranged when they are placed in a box with fixed dimensions with centrally aligned content. The first two containers are arranged in a first line, then the equation is placed on a second line, followed on a third line by the last container. We can also see that the containers on the first line are aligned on their top edge.

### 5.3.2 Line Styles and Line Joins

TDT defines line styles in a similar way to the PDF specifications [3] by defining a pattern, a line cap, a line join and a miter limit. The line pattern is defined as a dash array and a phase. The dash array specifies the lengths of alternating dashes and gaps. The phase specifies the starting point of the pattern. For example, Figure 5.9 defines a line pattern with 2 strokes and 3 gaps with a phase of 11.

A line style in TDT must define how a continuous line will start and end by defining the line cap. Figure 5.10 shows the line caps as defined by the PDF format.

When two lines meet at an angle the join point needs to be aesthetically drawn.
This is defined by the *line join* parameter as shown in Figure 5.11.

The *miter limit* is a maximum ratio between the miter length and the line width. More information about line style definition is available in Chapter 4, Section 4.3.2 of the Adobe PDF Specifications [3].

### 5.3.3 Borders

Without a clear definition of the line styles, dealing with borders can be a major cause of frustration when authoring tables as we showed in Section 1.7. A number of problems need careful consideration:

- **border space**: should the space required by the border width be added to the box dimensions, taken from the content space or it should be divided equally between the two? If the border width is divided equally, then the case with an odd border width needs to be clarified;
**border alignment**: because cells in a rectangular table are aligned based on a grid layout the join between borders of different widths, colours and styles must be established;

**border priority**: if borders of different colours overlap, which border will take priority: top border over the bottom border, the border of the last modified cell or based on a user setting?

TDT implements a *normal flow* of laying out containers and a *grid flow*. If boxes are arranged in the normal flow their borders are displayed next to each other as in Figure 5.12. In the *grid flow* borders are centred on each other and therefore they collapse similar to the CSS model [81].

### 5.3.4 Margins

Dealing with margins needs careful consideration. When two adjacent boxes have margins the drawing algorithm needs to decide if the margins should be added or they should merge by selecting the largest margin value as presented in Figure 5.13. Also, the algorithm must decide how to deal with margins for the case of inner boxes. TDT collapses margins for inner boxes as displayed in Figure 5.14.

### 5.3.5 Co-ordinate Space System

In any drawing system it is essential to define the co-ordinate space system: the origin, axes and the drawing units in use. Every drawing operation uses the co-ordinate
space as a canvas where various graphical elements are placed at the specified location, with the dimensions or font size provided using the drawing units.

For convenience we selected for TDT a co-ordinate system where the origin is in the bottom-left corner with the x-axis extending horizontally to the right and the y-axis extending upwards. The units are defined as points (1/72 of an inch) in both horizontal vertical direction.

5.3.6 Text Direction

Text is normally displayed from left-to-right but other text directions are required depending on the culture: right-to-left or top-to-bottom as displayed in Figure 5.15. This feature is particularly relevant for drawing tables. To reduce the horizontal space required to draw a table the text in the top header could be rotated. Text rotation is also important when determining the number and the dimensions of text configurations.
5.3.7 Images

TDT can insert images in tables in JPEG, PNG and GIF format. As the dimension of images are stored in pixels, TDT has to translate these dimensions in its own co-ordinate system and drawing units.

5.3.8 Equations

TDT allows users to insert equations provided in MathML format. WebEQ, a specialized library for rendering mathematical expressions developed by Design Science [35] is used to render each equation as an image which is then inserted in the table layout.

For example, Figure 5.16 shows an equation rendered with WebEQ as included in a table layout by TDT.

5.4 Generic Device and PDF Device

A device is an abstract object that must provide functions to assist in the rendering process: open and close the device, set colours, fill colours, line styles with all the properties described in Section 5.3.2 font settings (font family, style, size), vertical
and horizontal alignment, text spacing (word, character and line spacing), draw Bézier curves when irregular shapes are required and insert images.

The device object must provide functions that translate drawing units from TDT co-ordinate space in the co-ordinate space of the device implemented. Also, it has to translate font families from internal TDT fonts to fonts available in the device’s graphical environment. When a suitable font is not available the default font is applied or the user should be able to specify which font should be used.

It is important for any device object to indicate if it supports paging. For example, the table layout algorithm should be able to split tables across pages and if so, it needs information regarding the page size and its default settings such as margins, borders, paddings and the dimensions of any header and footer spaces.

In its current version TDT implements a PDF device which allows tables to be generated in PDF format. TDT uses the iText open source PDF library. This device provides specific settings for PDF documents such as the title, author, keywords, initial page index and initial zoom factor.

5.5 The Layout of Expressions

The main purpose of TDT is to draw tabular expressions. The layout of a tabular expression consists of two components: a caption box and a layout. The layout can be of any type: rectangular, circular or any other shape. TDT defines a programmable interface which must be implemented by any layout component. Thus, tabular expressions in TDT can use any layout as long as it implements this interface, including the computeLayout() and render() functions. Figure 5.17 shows the layout components of tabular expressions: the caption in a lighter shade of gray and the layout of the grid or the circular layout. The two components reside in the expression’s layout which is in fact another box element.

As displayed in Figure 5.17, the caption can be placed on any side of the layout.
It has a label, a number and a caption text. The number is formatted using any of the global numbering styles. The label and number are especially relevant when, due to space constraints, inner tables are displayed outside the parent table. In this case the inner table’s label is used as reference in the parent table.

TDT implements a rectangular layout for tabular expressions based on the definition given by Parnas [48]: a tabular expression is an indexed set of grids and a grid is an indexed set of expressions. To draw a tabular expression with a rectangular layout, from a layout perspective, there are at maximum 5 grids required: a two dimensional “main” grid that contains the data items and four uni-dimensional “header” grids, one on each side of the main grid. At least two grids are required at any point in time: the main grid and at least one header grid.

A grid is simply a collection of indexed cells. A cell is represented graphically by using the box element. Figure 5.18 shows a tabular expression with two header grids $H_1$ and $H_2$ where cells are indexed $H_1-1$ to $H_1-n$ and $H_2-1$ to $H_2-m$ and a main grid with $n$ columns and $m$ rows.

5.6 The Layout Control File

The TDT (Table Drawing Tool) is a prototype part of a larger Table Tool System – TTS –, a tool for tabular expressions which has three modules: Kernel, Input Tool and Output Tool.
To display a tabular expression TDT requires two sets of information to be made available in the XML file:

- Content Information. This data is provided by the Kernel Module;
- Layout Information. The layout parameters are provided by the Input Tool by means of a file in XML format.

This section is a high level presentation of how TDT controls the drawing parameters for tabular expressions. Appendix B contains the full specifications of XML layout control file format.

The XML file contains two sets of information:

- Layout control. The list of tabular expressions to be displayed and global layout settings;
- Device control. Information about the device used to display the tabular expressions;

5.6.1 Layout Settings Descriptor

The global layout settings section contains the definition for various global parameters such as the caption format, custom line styles, etc. Other settings can be added
The caption format contains the following elements:

- **Label.** A non empty string expression that will be used to name tabular expressions, i.e. “Expression”, “Function”, etc.;

- **Numbering style.** Introduced for future compatibility, the numbering style sets the format of the identifiers attached to each label of a caption: 1,2,3 or A,B,C or i,ii,iii. See Section 5.7 for further information on `NumberingStyle` data type.

- **Separator.** A character that will separate the numbering groups. For example for an inner expression the label number may be “3.4.1”. In this case the separator is the . (dot) character.

The global layout settings section also contains the definitions of a set of custom line styles that can be used throughout the document. Each custom line style is defined as described in Section 5.7 and it is associated with a string identifier. The identifier is used throughout the document to specify this custom style for example for a box element.

The global layout settings section has the format described in Listing 5.1.

5.6.2 Expressions Descriptor

All the expressions to be displayed in a document are defined under the `Expressions` element of the XML file. Each expression has the following elements:
<table>
<thead>
<tr>
<th>Label</th>
<th>Number</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>3.14</td>
<td>Stack definition using tabular expressions</td>
</tr>
</tbody>
</table>

- **ID.** Each expression has a positive ID which is used to identify the tabular expression i.e. when retrieving its contents from the Kernel module the ID of the expression must be provided;

- **Caption.** Contains the caption layout parameters as described below;

- **Layout.** This element groups all the layout parameters for a tabular expression layout;

- **Box.** An element that groups all the visual appearance parameters of the expression (dimensions, colours, text size, etc).

The caption of a tabular expression is the text that describes its contents. The format of the caption is shown in Table 5.1 When a tabular expression contains inner expressions that are displayed outside its layout the inner expressions are automatically replaced with a reference to their location. The reference text is in fact the captions label and number.

The expression number is automatically generated based on the position of the expression within the expression sequence in the document. Each time an expression is displayed the expression counter is incremented. This counter will be displayed in the Number field of the caption. When dealing with inner expressions, a new counter will start for each level within an expression. For example, for the first inner expression of expression number 3 the Number will contain the value 3.1, for the second inner expression it will contain 3.2, etc.

**Listing 5.2: The Expression and the Layout elements**

```xml
<Expression ID="1">
  <Caption>...</Caption>
  <Layout>
```
The layout element presented in Listing 5.2 groups all the control parameters that apply to all the layouts, independent of their implementation. To implement a new layout (e.g. circular layout) it is as simple as developing a new Java class. This class must implement a list of predefined functions. The class name must be provided in the <LayoutClass> element and all the control parameters must be specified as part of the <LayoutDescriptor> element.

5.6.3 Rectangular Layout Descriptor

In TDT a rectangular layout consists of a maximum of 5 grids and a maximum of 4 corners. The corners are just placeholders for additional information. In Figure 5.19 the grids are identified with G and the corners with C. The grids have indexes 0 to 4. The grid index 0 is always the main grid that is placed at the center of the layout. The grids with indexes 1 to 4 are the headers of the main grid.

The grids have the following requirements:

- At least the main grid (with index 0) must be specified;

- Grids 1 and 3 must have the same number of rows as G0 and at least one...
column. Grids 2 and 4 must have the same number of columns as G0 and at least one row. Corner elements are not required but they can be used to display additional information;

- No two grids can have the same index;

The layout descriptor of a rectangular layout has the elements shown in Listing 5.3. The \texttt{MergeCells} elements specifies whether adjacent cells with the same content should be merged or not. Merging cells improves the clarity of the expression.

\textbf{Listing 5.3: The Layout Descriptor element}

\begin{verbatim}
1 <Expression ID="1">
2  <Layout>
3   <LayoutDescriptor>
4     <MergeCells>TRUE</MergeCells>
5     <Grids>...
6   </Grids>
7   <Corners>...
8   </Corners>
9  </LayoutDescriptor>
10 </Layout>
11 </Expression>
\end{verbatim}

\section{Grid Descriptor}

A grid is a collection of cells arranged in rows and columns. The grid’s index, the number of rows and columns are specified as attributes of the grid. Its layout is controlled by the parameters specified in the \texttt{box} element.

\textbf{Listing 5.4: The Grid element}

\begin{verbatim}
1 <Grids>...
2  <Grid Index="0" Rows="25" Columns="4">
3     <Cells>
4       <Cell RowIndex="0" ColumnIndex="1">
5         <Box>
6           <Border>
7             <Width>100</Width>
\end{verbatim}
A cell is the smallest unit of a grid and is the final placeholder for the expressions of a tabular expression. Each cell is identified by two indexes: RowIndex and ColumnIndex. It contains text or a MathML expression that will be rendered as an equation. To control the formatting of a cell various parameters can be specified in its box element.

### 5.6.5 Device Descriptor

As we mentioned at the beginning of this section the XML layout file has two parts: the first part lists the global layout settings and the expressions to be rendered; the second part provides the specifications of the device to be used for rendering content.

A device is an abstract class that implements functions that render graphical elements to a specified output device: file, screen, printer, etc. This class also represents a contract (or interface) that any new device class must implement. For example, if required to render data on a new “printer” device then a new “printer device” class must be developed. This class must implement all the functions specified for the device as described in Appendix B.

In this version TDT implements a PDF device which renders expressions to a PDF file. To use the PDF device the <DeviceClass> is set to the PdfDevice Java class. All the configuration parameters of this class must be specified as sub-elements of the DeviceDescriptor element as shown in Listing 5.5.
The PDF class requires the output file name, all the details about the PDF file such as the title, author, keywords, etc. and the initial settings of the file such as the initial page to be displayed when the PDF file is opened and the initial zoom factor.

5.7 Custom Data Types

The following custom data types are defined in TDT:

- Line Style. This element can have a predefined value (i.e. SOLID, DOTTED, DASHED, etc.) or specified by a dash array, a phase, line cap, line join and a miter limit. See Section 5.3.2 for details on custom line styles;

- Line Cap Style. The line cap style specifies the decoration applied at the ends of a dashed or unclosed line;
• Line Join Style. The line join style specifies the shape that will be used when two consecutive segments of a path meet at an angle;

• Colour. The Colour Type is a structure with 3 integer variables R, G, B with a range between 0 and 255. These variables represent the amount of red, blue and green that a colour contains. Using these 3 values the user can specify up to 16,777,215 colours;

• Numbering Style. This type has the following values: NUMERIC, ROMAN, CHAR, LOW_I;

• Position. This is an enumerated data type with integer values: HIDDEN, TOP, RIGHT, BOTTOM, LEFT;

• Font Style is an enumerated data type with the following integer values: NORMAL, BOLD, ITALIC, BOLDITALIC.

5.8 Exceptions

As TDT has been developed in Java 1.6 it takes advantage of the resource management features to implement a flexible management of exceptions. The exception messages are defined in resources files (text files with.properties extension). The messages displayed in Table 5.2 can be amended by simply editing these resource files.

This table does not show an exhaustive list of exceptions and it has been provided just to show how exceptions are managed in TDT. The full list of exceptions is available in Appendix B.
### Table 5.2: Exceptions in TDT

<table>
<thead>
<tr>
<th>Exception Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX_REQUIRED_ELEMENT</td>
<td>A required value has not been provided.</td>
</tr>
<tr>
<td>EX_REQUIRED_LS_NAME</td>
<td>When defining a new line style, the style name must be provided.</td>
</tr>
<tr>
<td>EX_INVALID_INTEGER</td>
<td>The value provided cannot be converted to a value of type Integer</td>
</tr>
<tr>
<td>EX_DUPLICATE_INDEX</td>
<td>The index provided has already been used. The indexes must be unique.</td>
</tr>
</tbody>
</table>

### Listing 5.6: Solving OPL models in Java

```java
private void solveModel(String model, String modelTxt,
                        ArrayList<CellConfiguration> configurations) {
    IloOplFactory opf = new IloOplFactory();
    IloOplErrorHandler errHandler = opf.createOplErrorHandler();
    IloOplModelSource modelSource = opf.createOplModelSource(model);
    IloOplSettings settings = opf.createOplSettings(errHandler);
    IloOplModelDefinition def = opf.createOplModelDefinition(modelSource, settings);
    IloCplex cplex = opf.createCplex();
    cplex.setOut(null);
    IloOplModel opl = opf.createOplModel(def, cplex);
    IloOplDataSource dataSource=oplF.createOplDataSourceFromString(modelTxt, "table");
    opl.addDataSource(dataSource);
    IloOplDataElements dataElements = oplF.makeDataElements();
    opl.generate();
    if (cplex.solve()) {
        Double selected;
        IloIntVarMap sw = opl.getElement("x").asIntVarMap();
        for (int i = 0; i < sw.getSize(); i++) {
            IloTuple k = dataElements.getElement("configs").asTupleSet().makeTuple(i);
            configurations.get(i).selected = selected >= 0.99 ? true : false;
        }
        opl.printSolution(System.out);
    } else System.out.println("No solution!");
}
```

### 5.9 Integrating Optimization Engines

In this thesis we presented combinatorial optimization solutions to the table layout problem using OPL. As the Table Drawing Tool prototype is developed in Java, to implement these solutions and evaluate their effectiveness we used the IBM ILOG Concert technology. This technology provides interfaces for Java, C, C++, C#, Visual Basic and FORTRAN which allows developers to embed CPLEX algorithms within their application programs.

We defined a Layout Optimizer class that allowed us to determine for a grid what is the shortest layout that can be displayed in the given page width.
The `solveModel()` function in Listing 5.6 shows how the IBM ILOG Concert Technology is used in Java to interface with the OPL model. The `configurations` array contains all the cell configurations for a grid structure. The `model` variable contains the optimization model that is used to find a solution to the problem with the configurations input data provided in the `modelTxt` variable. The input data file contains all the necessary variables for the TSC model such as the page width, the sets of row and column spanning cells and the cell configurations. The TSC model is presented in Section 4.1.

After the model is initialised and the input data provided the CPLEX engine will try to resolve the problem and find a solution. If a solution is found for each binary variable $x[i]$ of the TSC model the corresponding cell configuration is selected and its width/height will be used by the drawing algorithm. The `configurations` array contains those cell configurations that should be used to obtain a layout that minimizes table height for the given page width.

In a similar way, layout optimizers can be defined for all the models presented in this thesis including CP Optimizer, the constraint programming engine and the LocalSolver the local search engine.

### 5.10 Tabular Layouts Generated with TDT

In this section give examples of how TDT renders tables by comparison with two original tables presented throughout this thesis.

We first show how the table in Figure 5.20 introduced in Chapter 1 is rendered using TDT. Only a partial layout is presented here to facilitate the comparison of the two layouts. The full table layout is presented in Appendix D where we include a larger selection of sample table layouts generated with TDT.

The minimum table height has been computed using IBM ILOG CPLEX solver [34], OPL [83] and the TSC model for tables with spanning cells presented in Section 4.1.
**Table 3.4-b**

Transitions Between Alignment, Navigation, and Test Modes
While the Aircraft Is Not Airborne (/ACARIB=SNNo)

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lastocal&quot;</td>
<td>&quot;Landln&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;L&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;OLB&quot;</td>
</tr>
</tbody>
</table>

**Figure 5.20:** An illustration of Table 3.4-b from the A-7E Documentation, also presented in Figure 1.4.

**Condition Table 3.4-b**

Transitions Between Alignment, Navigation, and Test Modes
While the Aircraft Is Not Airborne (/ACARIB=SNNo)

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Lastocal&quot;</td>
<td>&quot;Landln&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;L&quot;</td>
</tr>
<tr>
<td></td>
<td>&quot;OLB&quot;</td>
</tr>
</tbody>
</table>

**Figure 5.21:** The original table from Figure 5.20 is rendered with TDT.
It can be observed that compared with the original layout in the new layout presented in Figure 5.21, the text of the header cells 2, 10, 13 and 20 to 23 has been arranged in two lines. This has been determined following the optimization process, in order to minimize the table height. In this example we show how TDT handles fonts, background colours, spanning cells and border settings. The font in the table caption is *Arial* size 10 points while the font in the table content is set to *Courier* of size 8 points.

When the text is rotated by 90 or 270 degrees TDT is still able to computed the optimal layout. TDT computes cell configurations as when the text is layed-out horizontally and then mirroring the withs and heights. When the text is rotated freely at any angle, (i.e. in the original layout the text in the header cells is rotate at 60 degrees) cell configurations could be computed based on horizontal and vertical projections.

In Figure 5.23 we also show how the table presented in Figure 5.22, introduced in Chapter 1, can be optimized while taking into account various formatting parameters such as different font size, padding, border widths, etc. We selected this table because it demonstrates how TDT deals with inner tables: in cell 7,1 in the original layout the author makes reference to an external “Table 4.3.1-d”. While in the original documentation the author presented the inner table on its own in our layout we included this table as in the layout of cell 7,1. To optimize the table layout we applied our technique presented in Section 4.5 of merging the inner table in the parent table before computing the layout with the minimum height. This clearly demonstrates the advantages given by the methods presented in this thesis. It is important to note that in this example we focus only on the layout issues and not the logical connections between various cells when an inner table is include or excluded from the larger table. This area will require further investigation and demonstrates again the complexities of table layout.

Finally in Figure 5.24 we show how the minimum height can be computed for
Figure 5.22: An illustration of Table 4.3.1.c from the A-7E Documentation.

Figure 5.23: The original table from Figure 5.22 is rendered using TDT.
a larger table with 18 rows and 6 columns and up to 6 words in each cell when rendered for an A4 page size. In Appendix D we show a larger selection of table layouts generated with TDT.

5.11 Conclusions

The Table Drawing Tool prototype is a layout engine developed in Java 1.6 that allows users to render tables presented in an XML file format using the logical model of tabular expressions introduced by Jin and Parnas [48].

TDT combines the flexibility of the drawing model implemented by CSS with the precision given by the Adobe PDF file format. The user can precisely control location, borders, margins, padding, line styles, numbering and caption styles, colours and fonts.

TDT can find optimal table layouts for tables that contains text by using the IBM ILOG Concert Technology to provide an interface to optimization models developed in OPL for the CPLEX engine. This allowed us to evaluate the effectiveness of the solutions to the table layout problem provided in Chapter 3 and Chapter 4.

TDT allows users to render expressions with a rectangular layout to a PDF device. The architecture of TDT allows additional tabular layouts to be defined such as circular layouts or other custom layouts. Also, additional rendering devices can be added to TDT by simply developing Java classes that implement a predefined set of functions detailed in Appendix B.
| Figure 5.24: A 18x6 table with up to 6 words per cell which has the minimum height computed by TDT |
Chapter 6

Conclusions and Future Directions

6.1 Conclusions

In this thesis we review the challenges of authoring tables and show how optimal table layout can automatically be determined. We highlight that while WYSIWYG tools have been widely used, supported and developed they lack support for operations at the table structure level. These tools also lack features for automatic table layout optimization. Therefore formatting operations that require changes at the table structure level are still hard, time consuming and prone to errors.

We propose new mathematical programming solutions for the table layout problem, the problem of automatically finding the layout that minimizes table height for a given page width. We model the problem in OPL using mixed integer programming, constraint programming and local search methods and we provide a comparative evaluation of the performance of these solutions:

1. OPL provides viable solutions for applications where the table layout with the minimum height for a given page width must be found. The mixed integer programming model TSC which uses the CPLEX optimization engine generally provides faster results than the Constraint Programming and the Local Solver engines;
2. If proving that a solution is optimal is not essential, Local Solver can provide solutions in a shorter time than CPLEX. As this model is based on local search techniques it only stops if a time limit is set; it cannot guarantee that the solution found is optimal;

3. The Constraint Programming engine can find in some instances a feasible solution faster than the CPLEX engine, but it continues to explore the solution space until it determines whether the solution found is the final objective;

4. As all the optimization models use the set of cell configurations as input, *paragraphing* – determining cell configurations or how to split the text on consecutive lines – is an essential aspect as it impacts on the quality and computational time of these solutions;

5. Using the models presented in this thesis it is more difficult to solve the table layout problem when there is a large number of rows, columns and column configurations than when the table has a large number of cell configurations;

6. Performance depends highly on the input data. Generally, the table layout problem is more difficult to solve (it takes longer to find the solution) when the page width is closer to the minimum table width. Nonetheless, the experimental results showed exceptions to this;

7. The advantage of using a modelling language to model the table layout problem is that other user imposed constraints can be easily added to the model.

We also discuss the problem of nested tables. Because our models require that minimal cell configurations to be known, computing these configurations for the inner tables suggests a recursive method for finding the configurations with minimum table heights and widths and then recursively finding optimal solutions. We show that a recursive optimization model raises a number of difficult problems that require long computational times and it can can be inefficient. Therefore we suggest as a
possible solution merging the inner tables recursively in the parent table and finding a solution for this larger problem.

Dynamically formatting content on portable devices raises difficult problems because of the variance of screen resolutions, processing power and user interface designs of these devices. Besides HTML and eBooks formats, PDF is widely used to deliver electronic content because the designers can precisely control how the document is presented. As PDF content is formatted across multiple pages, we address the problem of formatting wide tables that span multiple pages in such a way that the impact on the meaning of data is minimized.

We present new combinatorial optimization solutions for the problems of reordering columns in wide tables such that the page count is minimized while minimizing relative column positioning changes and the number of column groups split across pages. In our evaluations we found that the page count can be reduced up to 25% under 2s. Our experiments showed that the difficulty of the problems does not only depend on the problem size but on the complexity of the data.

To evaluate our solutions we developed a Table Drawing Tool prototype, TDT, that implements automatic table layout. TDT combines the flexibility of a presentation model based on a hierarchical structure of box elements with the precision given by the PDF format to draw tabular expressions. We presented a number of problems raised by the drawing process which need to be addressed in order to produce high quality tables.

We developed a set of tables that can be used by the research community as starting point for developing a standardised set of benchmark tables. This set can be used for comparative evaluations of further algorithms and methods that will be developed in the future. Our set of tables and additional information about table layout is available at

http://www.tabularlayout.org/
6.2 Future work

As constraint optimization methods are further developed and the optimization languages evolve the problems that seem hard to solve today will become more and more accessible. Further research could focus on recursively solving the problem of optimising the layout of nested tables especially if optimization engines will provide improved support for recursive optimization flow control.

One possible area of further research is the study of alternative constraint optimization models, in particular a deeper investigation of the effects of branching on column widths. While we suggested three constraint optimization models there could potentially be alternative searching strategies that will perform better than the default search strategies implemented in the CP Optimizer. Also, an evaluation of modelling the table layout problem using a generic placement engine such as the geost kernel introduce by Beldiceanu et al. [14]. It would be interesting to find out the effects of using global positioning constraints such as the non-overlapping constraint.

Another area that could be explored further is an evaluation of alternative combinatorial optimization engines performance in order to establish if any other strategy/engine combination works better than the solutions presented in this thesis. A particular focus could be on local search engines that currently are under development and also on open source optimization engines that have the potential to be easily integrated into existing document tools.

Developing table tools is hard and the number of engineering problems encountered in the development of such tools may become easier if fundamental logical table models are developed. If new improved tools are developed there will always be a need to evaluate their performance and compare it with other solutions that have already been evaluated. Therefore it is essential that a set of benchmark tables and their optimal layout is developed for comparative evaluations of future opti-
mization methods or other tabular tools. While we aim to start this process by making available the datasets used in the evaluation of our methods we hope that others researchers will join and contribute to this objective.

As with any other tool more work can be done to improve the TDT prototype: computing cell configurations for text rotated at any degree (currently TDT handles only text rotated by 90 or 270 degrees), the introduction of table templates, improved graphical styles, a study of how lines of different styles and widths should intersect; also a graphical user interface for table input/output that takes advantages of the evaluation results presented in Section 4.4.1 and shows an estimation of the optimal table height value.

The methods described in this thesis could be added to commercially available document authoring tools or to open source projects such as LibreOffice in order to provide the users with table layout optimization features which currently are not available. This could also unearth further engineering or optimization problems.
Appendix A

Optimization Models

Throughout this thesis we presented a number of models for automatic table layout optimization using Optimization Programming Language - OPL and the IBM ILOG Optimization Studio. For clarity, only the relevant parts of the optimization models are presented and discussed throughout the thesis. In order to avoid any confusion or omissions, we present in this appendix the full code for each optimization models BMIP, MIP, CP, TSC, LSP, etc. and also a sample data input file in Section A.10.
A.1 Basic MIP Model for Table Layout - BMIP

Listing A.1: BMIP model for the table layout problem (part 1)

1 /***********************************************************/
2 /* OPL 5.5 Model */
3 /* Author: Mihai Bilauca. Presented at DocEng2010 */
4 /* Creation Date: 09/08/2009 at 15:58 */
5 /* Description: In this model, in order to minimize h, table layout height (the sum */
6 /* of rows max height) a cell configuration is selected for each cell in such a way */
7 /* that two conditions are satisfied: */
8 /* 1) the sum of the maximum widths for each cell in each column, w is less than */
9 /* page width pageW */
10 /* 2) each cell can have only 1 selected configuration */
11 /******************************************************************************/
12
tuple Conf{
    // configuration data type
    int w;
    int h;}

tuple CellConf{
    // configuration for cell i,j
    int i;
    int j;
    Conf c;}

tuple Cell{
    // cell data type
    int i;
    int j;}

int pageW = ...; // page width
{CellConf} configs = ...; // cell configurations
{Cell} cells = {<i,j> | <i,j,k> in configs}; // the set of cells
{int} rows = {i | <i,j> in cells}; // the set of rows
{int} columns = {j | <i,j> in cells}; // the set of columns

dvar int cellSel[configs] in 0..1; // cell configuration selector

// compute table width
dexpr int cellW[<i,j> in cells] = sum(<i,j,k> in configs) cellSel[<i,j,k>] * k.w;
dexpr int colW[j in columns] = max(i in rows) cellW[<i,j>];
dexpr int tableW = sum(j in columns) colW[j];
// compute table height
def int cellH[i, j in cells] = sum<i, j, k in configs> cellSel<i, j, k> * k.h;
def int rowH[i in rows] = max<j in columns> cellH[i, j];
def int tableH = sum<i in rows> rowH[i];

minimize tableH;

constraints
{
    ct1: // table width must be less than page width
tableW <= pageW;

    ct2: // each cell can have only one selected configuration
        forall<i in rows, j in columns>
            sum<i, j, k in configs> cellSel<i, j, k> == 1;
}
A.2 A MIP Model

Listing A.2: MIP model for the table layout problem

```
/* OPL 5.5 Model
* Author: Mihai Bilaucu. Presented at DocEng2010
* Creation Date: 02/01/2010 at 13:30
* Description: In this model, in order to minimize h, table layout height (the sum
* of rows max height), a width for each column is first selected in colSet[] from
* the set of possible column widths colWset. The condition is that the sum of the
* column widths w must be less or equal with the page width pageW. Also, for each
* cell in each column, only one cell configuration is selected only if its width
* is less or equal with the selected column width

 resemblance to the previous model
***************/
tuple Conf{
    // configuration data type
    int w;
    int h;
}
tuple CellConf{
    // configuration for cell i,j
    int i;
    int j;
    Conf c;
}
tuple Cell{
    // cell data type
    int i;
    int j;
}
tuple ColW{
    // column width data type
    int j;
    int w;
}
int pageW = ...; // page width
{CellConf} configs = ...; // the set of cell configurations
{Cell} cells = {<i,j> | <i,j,k> in configs}; // the set of cells
{int} rows = {i | <i,j> in cells}; // the set of rows
{int} columns = {j | <i,j> in cells}; // the set of columns
```
int minW[ j in columns] = max( i in rows) min( i, j, k in configs) k.w;
int minH[ i in rows] = max( j in columns) min( i, j, k in configs) k.h;
{ColW} colWset = {{ j, k.w} | < i, j, k> in configs : k.w >= minW[ j]};
int minPageW = sum( j in columns) minW[ j];
int minPageH = sum( i in rows) minH[ i];
int maxPageW = sum( j in columns) max( i in rows) max(< i, j, k> in configs) k.w;
int maxPageH = sum( i in rows) max( j in rows) max(< i, j, k> in configs) k.h;
int confs = card( configs);
int wSet = card( colWset);
dvar int colSel[ colWset] in 0..1; // column width selector
dvar int cellSel[ configs] in 0..1; // cell configuration selector

// compute table height
dexpr int cellH[ < i, j> in cells] = sum(< i, j, k> in configs) cellSel[< i, j, k>] * k.h;
dexpr int rowH[ i in rows] = max( j in columns) cellH[< i, j>];
dexpr int tableH = sum( i in rows) rowH[ i];
dexpr int cellW[< i, j> in cells] = sum(< i, j, k> in configs) cellSel[< i, j, k>] * k.w;
dexpr int colW[ j in columns] = sum(< j, w> in colWset) colSel[< j, w>] * w;
dexpr int tableW = sum( j in columns) colW[ j];

minimize tableH;

constraints
{
  ct1: // table width must be less than page width
      tableW <= pageW;
  ct2: // each column can have only one select configuration
     forall( j in columns)
      sum(< j, k> in colWset) colSel[< j, k>] == 1;
  ct3: // cell width must be less or equal with column width
     forall( j in columns, i in rows)
      cellW[< i, j>] <= colW[ j];
  ct4: // each cell can have only one select configuration
     forall( i in rows, j in columns)
      sum(< i, j, k> in configs) cellSel[< i, j, k>] == 1;
}
A.3 Constraint Programming Model - CP

Listing A.3: CP model for the table layout problem

1 /**********************************************************************************
2 * OPL 5.5 Model
3 * Author: Mihai Bilauca. Presented at DocEng2010
4 * Creation Date: 10/01/2010 at 22:39
5 * Description: In this CP model, in order to minimize table height h, a width for
6 * each column is determined in colW[] and then a height for each row in rowH[
7 * 
8 * The range of each column’s width is defined between minW and maxW
9 * The range of each row’s height is defined between minH and maxH
10 *
11 * The CP engine controls the order in which these variables are fixed:
12 * first, a column width is selected; the width of every other column is selected
13 * in the descending order of its minimum regret, that is the difference between
14 * its minimum width and its next minimum width;
15 * the same method is applied when determining row heights;
16 **********************************************************************************/
17 using CP; // use Constraint Programming engine
18
tuple Conf{ // configuration data type
19  int w;
20  int h;
21 }
22
tuple CellConf{ // configuration for cell i,j
23  int i;
24  int j;
25  Conf c;
26 }
27
tuple Cell{ // cell data type
28  int i;
29  int j;
30 }
31
tuple Dim{ // dimension data type
32  int j;
33  int w;
34 }
```plaintext
int pageW = ...; // page width

{CellConf} configs = ...; // the set of cell configurations

{Cell} cells = {<i,j> | <i,j,k> in configs}; // the set of cells

{int} rows = {i | <i,j> in cells}; // the set of rows

{int} columns = {j | <i,j> in cells}; // the set of columns

int minW[j in columns] = max(i in rows) min(<i,j,k> in configs) .w;
int minH[i in rows] = max(j in columns) min(<i,j,k> in configs) .h;

{Dim} colWset = {<j,k.w> | <i,j,k> in configs : k.w >= minW[j] && k.h >= minH[i]};
{Dim} rowHset = {<i,k.h> | <i,j,k> in configs : k.w >= minW[j] && k.h >= minH[i]};

sorted {CellConf} fConfigs = {<i,j,k> | <i,j,k> in configs};

int cons = card(configs);

int wSet = card(colWset);
int hSet = card(rowHset);

execute{
  writeln("Cnf: ", cons);
  writeln("wSet: ", wSet);
  writeln("hSet: ", hSet);
}

dvar int colW[j in columns] in min(<j,w> in colWset) w .. max(<j,w> in colWset) w;
dvar int rowH[i in rows] in min(<i,h> in rowHset) h .. max(<i,h> in rowHset) h;

// compute table width and table height

dexpr int tableW = sum(j in columns) colW[j];
dexpr int tableH = sum(i in rows) rowH[i];

execute{
  var f = cp.factory
  var phase0 = f.searchPhase(colW, f.selectLargest(f.regretOnMax()),
                             f.selectLargest(f.value()));
  var phase1 = f.searchPhase(rowH, f.selectLargest(f.regretOnMin()),
                             f.selectSmallest(f.value()));
  cp.setSearchPhases(phase0, phase1);
}
```

minimize tableH;

constraints
{
  ct1: // table width must be less than page width
tableW <= pageW;

  ct2: // only one column width can be selected
forall (j in columns)
    sum(<j,w> in colWset) (colW[j]==w) == 1;

  ct3: // only one row height can be selected
forall (i in rows)
    sum(<i,h> in rowHset) (rowH[i]==h) == 1;

  ct4: // at least one suitable cell configuration exists
forall (i in rows, j in columns)
    sum(<i,j,k> in fConfigs) (k.w <= colW[j] && k.h <= rowH[i]) >= 1;
}
A.4 Constraint Programming Model - CPI1

Listing A.4: CPI1 - Constraint Programming model with index decision variables

/* ************************************************************************* *
 * OPL 6.3 Model
 * Author: Mihai Bilauca
 * Creation Date: 16 Jul 2011 at 23:05:40
 * Description: Table Layout Constraint Programming model where cell, column
 * and row configuration indexes are selected in sc[], sw[] and sh[] respectively
 ***************************************************************************/

using CP;  // use Constraint Programming engine

tuple Conf{  // configuration data type
    int w;
    int h;
}
tuple CellConf{  // configuration for cell i,j
    int i;
    int j;
    Conf c;
}
tuple Cell{  // cell data type
    int i;
    int j;
}

int pageW = . . . ;  // page width

{CellConf} configs = . . . ;  // the set of cell configurations
{Cell} cells = {<i,j> | <i,j,k> in configs};  // set of cells
{int} rows = {i | <i,j> in cells};  // set of rows
{int} cols = {j | <i,j> in cells};  // set of cols
int minW[j in cols] = max(i in rows) min(<i,j,k> in configs) k.w;
int minH[i in rows] = max(j in cols) min(<i,j,k> in configs) k.h;

// the set of cell configs for each cell i,j
{Conf} cellC<i,j> in cells | {k | <i,j,k> in configs};

// sets of column widths and row heights for each column j and row i
sorted {int} colWset[j in cols] = {k.w | <i,j,k> in configs : k.w >= minW[j]};
sorted {int} rowHset[i in rows] = {k.h | <i,j,k> in configs : k.h >= minH[i]};
int Kc = \max(i \text{ in } \text{rows}, j \text{ in } \text{cols}) \ \text{card}\{(i,j,k) \mid (i,j,k) \text{ in } \text{configs})\};

int Kw = \max(j \text{ in } \text{cols}) \ \text{card}(\text{colWset}[j]);

int Kh = \max(i \text{ in } \text{rows}) \ \text{card}(\text{rowHset}[i]);

int m = \text{card}(\text{rows});

int n = \text{card}(\text{cols});

range r = 0..m-1;

range c = 0..n-1;

range kw = 0..Kw-1;

range kh = 0..Kh-1;

range kc = 0..Kc-1;

int cellWarr[r][c][kc] = \sum(i \text{ in } \text{rows}, j \text{ in } \text{cols}, q \text{ in } \text{cellC}[i,j]) q.w;

int cellHarr[r][c][kc] = \sum(i \text{ in } \text{rows}, j \text{ in } \text{cols}, q \text{ in } \text{cellC}[i,j]) q.h;

int colWarr[c][kw] = \sum(j \text{ in } \text{cols}, q \text{ in } \text{colWset}[j]) q.w * q.h;

int rowHarr[r][kh] = \sum(i \text{ in } \text{rows}, q \text{ in } \text{rowHset}[i]) q.w;

int sc[r][c] in kc; // cell configuration index

dvar int sw[c] in kw; // column width index

dvar int sh[r] in kh; // row height index

dexpr int cellW[i \text{ in } r][j \text{ in } c] = cellWarr[i][j][sc[i][j]];

dexpr int cellH[i \text{ in } r][j \text{ in } c] = cellHarr[i][j][sc[i][j]];

dexpr int colW[j \text{ in } c] = colWarr[j][sw[j]];

dexpr int rowH[i \text{ in } r] = rowHarr[i][sh[i]];

dexpr int tableW = \sum(j \text{ in } c) colW[j];

dexpr int tableH = \sum(i \text{ in } r) rowH[i];

minimize tableH;

subject to

\{ 

ct1: // table width must be less than page width

\text{tableW} \leq \text{pageW};

ct2: // only valid configurations (exclude dummy configs)

forall(i \text{ in } \text{rows}, j \text{ in } \text{cols})

\text{cellW}[i][j] * \text{cellH}[i][j] > 0;

\}
ct3:  // cell's width must be less than column width
forall (i in rows, j in cols)
    cellW[i][j] <= colW[j];

t4:  // cell's height must be less than row height
forall (i in rows, j in cols)
    cellH[i][j] <= rowH[i];
### Listing A.5: CPI2 - Constraint Programming model with reduced decision variables

```plaintext
using CP;  // use Constraint Programming engine

tuple Conf{ // configuration data type
    int w;
    int h;
}

tuple CellConf{ // configuration for cell i,j
    int i;
    int j;
    Conf c;
}

tuple Cell{ // cell data type
    int i;
    int j;
}

tuple CellSpan{ // the set of cell configurations
    int i;
    int j;
    int s;
};

int pageW = ...;  // page width
{CellConf} configs = ...; // the set of cell configurations
{CellSpan} rowSpan = ...; // the set of row spanning cells
{CellSpan} colSpan = ...; // the set of column spanning cells
{Cell} cells = {<i,j> | <i,j,k> in configs}; // the set of cells
```
\{(\text{int})\text{ rows} = \{i \mid <i,j> \in \text{cells}\}; \quad \text{// the set of rows}\)
\{(\text{int})\text{ cols} = \{j \mid <i,j> \in \text{cells}\}; \quad \text{// the set of cols}\)

\text{int} minW[j \in \text{cols}] = \max(i \in \text{rows}) \min(<i,j,k> \in \text{configs}) k.w;

\text{int} minH[i \in \text{rows}] = \max(j \in \text{cols}) \min(<i,j,k> \in \text{configs}) k.h;

\text{int} Kw = \max(j \in \text{cols}) \text{card}(\text{colWset}[j]);
\text{int} Kh = \max(i \in \text{rows}) \text{card}(\text{rowHset}[i]);
\text{int} m = \text{card(rows)};
\text{int} n = \text{card(cols)};

\text{range} r = 0..m-1;
\text{range} c = 0..n-1;
\text{range} kw = 0..Kw-1;
\text{range} kh = 0..Kh-1;

\text{dvar int} sw[c] in kw; \quad \text{// column width index}
\text{dvar int} sh[r] in kh; \quad \text{// row height index}

\text{dexpr int} colW[j in c] = \text{colWarr}[j][sw[j]];
\text{dexpr int} rowH[i in r] = \text{rowHarr}[i][sh[i]];

\text{dexpr int} tableW = \sum(j in c) \text{colW}[j];
\text{dexpr int} tableH = \sum(i in r) \text{rowH}[i];

\text{minimize} \text{tableH};

\text{subject to}
\{
\text{ct1:} \quad \text{// table width must be less than page width}
\quad \text{tableW} \leq \text{pageW};

\text{ct2:} \quad \text{// at least one suitable cell configuration exists}
\quad \forall (i \in \text{rows}, j \in \text{cols})
\quad \sum(<i,j,k> \in \text{configs}) \{k.w \leq \text{colW}[j] \&\& k.h \leq \text{rowH}[i]\} \geq 1;
\}

181
A.6 TSC Model


```plaintext
/* ************************************************************************* *
 * OPL 6.3 Model                                                        *
 * Author: Mihai Bilauca. Presented at DocEng2011                        *
 * Creation Date: 23 Jan 2011 at 15:11:05                               *
 * Description: In this MIP model, in order to minimize the height of a table with *
 * spanning cells, row heights and column widths are selected in y[i] and z[j] for *
 * each row and column from predefined sets rowHset[i] and colWset[j]. Then, cell *
 * configurations that fit in the selected column widths and row heights are *
 * selected in z[j]. For narrow columns that are spanned by large cells horizontal *
 * and vertical 'glue' is also determined                                 *
 ************************************************************************** *

tuple Conf{ // configuration data type
    int w;
    int h;
}

tuple CellConf{ // configuration for cell i,j
    int i;
    int j;
    Conf c;
}

tuple Cell{ // cell data type
    int i;
    int j;
}

tuple Pair{ // dimension
    int idx;
    int val;
}

tuple CellSpan{ // spanning cell data type
    int i;
    int j;
    int s;
};
```
```plaintext
40 \{CellConf\} configs = \ldots; \quad \text{// the set of cell configurations}
41 \{CellSpan\} rowSpan = \ldots; \quad \text{// the set of row spanning cells}
42 \{CellSpan\} colSpan = \ldots; \quad \text{// the set of column spanning cells}
44 \{Cell\} cells = \{<i,j> | <i,j,k> in configs\}; \quad \text{// the set of cells}
45 \{int\} rows = \{i | <i,j> in cells\}; \quad \text{// the set of rows}
46 \{int\} columns = \{j | <i,j> in cells\}; \quad \text{// the set of columns}

48 \text{// the number of rows spanned by a cell}
49 int Sh[cells] = [<i,j>: s | <i,j,s> in rowSpan];

51 \text{// the number of cols spanned by a cell}
52 int Sw[cells] = [<i,j>: s | <i,j,s> in colSpan];

54 int minH[i in rows] = max(j in columns) min(<i,j,k> in configs) (k.h * (1-Sh[i,j]));
55 int minW[j in columns] = max(i in rows) min(<i,j,k> in configs) (k.w * (1-Sw[i,j]));

59 \{Pair\} rowHset = \{<i, k.h> | <i,j,k> in configs : k.h >= minH[i] & Sh[i,j]==0\};
60 \{Pair\} colWset = \{<j, k.w> | <i,j,k> in configs : k.w >= minW[j] & Sw[i,j]==0\};

62 int minPageW = sum(j in columns) minW[j];
63 int minPageH = sum(i in rows) minH[i];
64 int maxPageW = sum(j in columns) max(i in rows) max(<i,j,k> in configs) k.w;
65 int maxPageH = sum(i in rows) max(j in rows) max(<i,j,k> in configs) k.h;

67 dvar int x[configs] in 0..1; \quad \text{// cell configuration selector}
68 dvar int y[rowHset] in 0..1; \quad \text{// row height selector}
69 dvar int z[colWset] in 0..1; \quad \text{// column width selector}
70 dvar int Gh[rows] in 0..maxint; \quad \text{// row glue for merged cells}
71 dvar int Gw[columns] in 0..maxint; \quad \text{// column glue for merged cells}

// table height
74 dexpr int cellH[i,j] in cells] = sum(<i,j,k> in configs) x[i,j,k] * k.h;
75 dexpr int rowH[i in rows] = sum(<i,h> in rowHset) y[i,h] * h;
76 dexpr int tableH = sum(i in rows) (rowH[i] + Gh[i]);
77 \text{// table width}
78 dexpr int cellW[i,j] in cells] = sum(<i,j,k> in configs) x[i,j,k] * k.w;
79 dexpr int colW[j in columns] = sum(<j,w> in colWset) z[j,w] * w;
80 dexpr int tableW = sum(j in columns) (colW[j] + Gw[j]);
```
// used by equal glue distribution
// dexpr int glueW[i,j,s] in colSpan =
//       cellW[i,j] - sum(col in j..j+s) colW[col];
minimize tableH;

constraints
{
c1: // table width must be less than page width
tableW <= pageW;

c2: // height constraint
forall(i in rows, j in columns)
    if (Sh[i,j] > 0)
        cellH[i,j] <= sum(row in i..i + Sh[i,j]) (rowH[row] + Gh[row]);
    else
        cellH[i,j] <= rowH[i];

c3: // width constraint
forall(j in columns, i in rows)
    if (Sw[i,j] > 0)
        cellW[i,j] <= sum(col in j..j + Sw[i,j]) (colW[col] + Gw[col]);
    else
        cellW[i,j] <= colW[j];

c4: // each cell can have only 1 selected configuration
forall(i in rows, j in columns)
    sum(<i,j,k> in configs) x<i,j,k> == 1;

c4_1: // each row can have only 1 selected configuration
forall(i in rows)
    sum(<i,k> in rowHset) y<i,k> == 1;

c4_2: // each column can have only 1 selected configuration
forall(j in columns)
    sum(<j,k> in colWset) z<j,k> == 1;

/* c5: // equal glue distribution */
forall(<i,j,s> in colSpan)
forall(col in j..j+s)
    Gw[col] <= glueW[i,j,s] / (s+1); */
A.7 Local Search Model - LSP

Listing A.7: LSP - Local Search model for the table layout problem.

/* *******************************************************/
/* LocalSolver 2.0 Model */
/* Author: Thierry Benoist */
/* Creation Date: 28 Feb 2012 at 12:41:48 */
/* Description: In this model for Local Solver 2.0, in a pre-processing step, an */
/* array of minimal cell heights is built for each possible column width. The model */
/* finds only which column width must be selected from the array of possible widths */
/* The key difference of this model is that the cell height is not computed but */
/* simply selected from the array minHeightForWidthIndex using the same column */
/* index selector xcol[c]/n*/
/* *******************************************************/

function input() {
  bigInt = 1000000;

  instance = openRead(instancefile);

  readData();
  
}

// returns the min height for cell [row, col] when the width is set to 'width'
function getHeightForWidth(row, col, width) {
  return min[k in 0..nbConfigurations[row][col]−1:configurationsWidths[row][col][k] <= width](configurationsHeights[row][col][k]);
}

// returns the min width for a cell (min over all configurations for this cell)
function getMinWidth(row, col) {
  return min[k in 0..nbConfigurations[row][col]−1](configurationsWidths[row][col][k]);
}

// returns the min width for this column (max over all minWidth for cells of this column)
function getMinPossibleWidth(col) {
  return max[row in 0..nbRows−1](getMinWidth(row,col));
}

// returns the max width for this column
function getMaxPossibleWidth(col) {
    return max[row in 0..nbRows−1][k in 0..nbConfigurations[row][col]−1](
        configurationsWidths[row][col][k]);
}

// returns true if the given width is one of the possible width of this column
function containsWidth(col, width) {
    for[n in 0..nbWidths[col]−1] {
        if (colWidths[col][n]==width) return true;
    }
    return false;
}

// returns the min height for cell [row, col]
// (min over all configurations for this cell)
function getHeight(row, col) {
    return min[k in 0..nbConfigurations[row][col]−1](
        configurationsHeights[row][col][k]);
}

// returns the min possible height for this row
// (max over all minHeight for this row)
function getMinPossibleHeight(row) {
    return max[col in 0..nbCols−1](getMinHeight(row, col));
}

// returns true if the given height is one of the possible height for this row
function containsHeight(row, height) {
    for[n in 0..nbHeights[row]−1] {
        if (rowHeights[row][n]==height) return true;
    }
    return false;
}

function readData() {
    firstInt = readInt(instance);
    if (maxW == nil) maxW = firstInt;
    else println("MaxW fixed by user to "+maxW);
    nbRows = readInt(instance);
    nbCols = readInt(instance);
println("nbRows=nbRows+, nbCols=nbCols+");

nbConfigurations[r in 0..nbRows-1][c in 0..nbCols-1] = 0;
cellMinWidth[r in 0..nbRows-1][c in 0..nbCols-1] = bigInt;
minWidthConfig[r in 0..nbRows-1][c in 0..nbCols-1] = -1;

globalMaxCellHeight = 0;

while (!eof(instance)) {
    row = readInt(instance);
    col = readInt(instance);
    width = readInt(instance);
    height = readInt(instance);
    local k = nbConfigurations[row][col];
    if (height > globalMaxCellHeight) globalMaxCellHeight = height;
    configurationsHeights[row][col][k] = height;
    configurationsWidths[row][col][k] = width;
    if (width < cellMinWidth[row][col]) {
        cellMinWidth[row][col] = width;
        minWidthConfig[row][col] = k;
    }
    nbConfigurations[row][col] = k+1;
}

// then we build the union of widths for each column
minPossibleWidth[c in 0..nbCols-1]=getMinPossibleWidth(c);
maxPossibleWidth[c in 0..nbCols-1]=getMaxPossibleWidth(c);
computeColWidths();
minPossibleHeight[r in 0..nbRows-1]=getMinPossibleHeight(r);
computeRowHeights();

println("MAX CELL HEIGHT = "+globalMaxCellHeight);
globalMinPossibleHeight = sum[r in 0..nbRows-1](minPossibleHeight[r]);
println("MIN TOTAL HEIGHT = "+ globalMinPossibleHeight);

minHeightForWidthIndex[r in 0..nbRows-1][c in 0..nbCols-1][n in 0..nbWidths[c]-1]
    = getMinHeightForWidth(r,c,colWidths[c][n]);
}

function computeColWidths() {
    for [c in 0..nbCols-1] {
        nbWidths[c]=0;
    }
for [r in 0..nbRows-1][k in 0..nbConfigurations[r][c]-1] {
    local w = configurationsWidths[r][c][k];
    if (w >= minPossibleWidth[c] && !containsWidth(c, w)) {
        local n = nbWidths[c];
        colWidths[c][n] = w;
        if (w == minPossibleWidth[c]) indexOfMinWidth[c] = n;
        if (w == maxPossibleWidth[c]) indexOfMaxWidth[c] = n;
        nbWidths[c] = n + 1;
    }
}

function computeRowHeights() {
    for [r in 0..nbRows-1] {
        nbHeights[r] = 0;
        maxh = 0;
        for [c in 0..nbCols-1][k in 0..nbConfigurations[r][c]-1] {
            local h = configurationsHeights[r][c][k];
            if (h >= minPossibleHeight[r] && !containsHeight(r, h)) {
                local n = nbHeights[r];
                rowHeights[r][n] = h;
                if (h > maxh) {
                    indexOfMaxHeight[r] = n;
                    maxh = h;
                }
                nbHeights[r] = n + 1;
            }
        }
    }
}

function model() {
    // decision variables col widths
    xcol[c in 0..nbCols-1][n in 0..nbWidths[c]] <- bool();
    for [c in 0..nbCols-1]
        constraint sum[n in 0..nbWidths[c]-1](xcol[c][n]) == 1;
    xColWidth[c in 0..nbCols-1] <-
        sum[n in 0..nbWidths[c]-1](xcol[c][n] * colWidths[c][n]);
xCellHeight[r in 0..nbRows-1][c in 0..nbCols-1] <= sum[n in 0..nbWidths[c]-1][xcol[c][n] * minHeightForWidthIndex[r][c][n]];
xRowHeight[r in 0..nbRows-1] <= max[c in 0..nbCols-1][xColWidth[c]];
totalWidth <= sum[c in 0..nbCols-1][xCellHeight[r][c]];
totalHeight <= sum[r in 0..nbRows-1][xRowHeight[r]];

constraint totalWidth <= maxW;
minimize totalHeight;

// Second criterion: minimize the number of cells responsible for the height of
// their row == 1 if this cell is responsible for the height of this row
xCellLimit[r in 0..nbRows-1][c in 0..nbCols-1] <=
    xCellHeight[r][c] == xRowHeight[r];
minimize sum[r in 0..nbRows-1][c in 0..nbCols-1][xCellLimit[r][c]];
}

function param() {
    sumOfWidths = sum[c in 0..nbCols-1][minPossibleWidth[c]];
    if (sumOfWidths > maxW)
        error("On this instance page width cannot be smaller than "+sumOfWidths);
    for[c in 0..nbCols-1] {
        local delta = maxPossibleWidth[c] - minPossibleWidth[c];
        local selected = indexOfMinWidth[c];
        if (sumOfWidths + delta <= maxW) {
            selected = indexOfMaxWidth[c];
            sumOfWidths = sumOfWidths + delta;
        }
        for[n in 0..nbWidths[c]]
            setValue(xcol[c][n], n == selected);
    }
    setObjectiveBound(0, globalMinPossibleHeight);
    setObjectiveBound(1, nbRows * nbCols);
    println("INITIAL WIDTH = "+sumOfWidths);
}
function getOutputConfig(row, col) {
    local cw = getValue(xColWidth[col]);
    local rh = getValue(xRowHeight[row]);
    for [k in 0..nbConfigurations[row][col]−1]
        if (configurationsWidths[row][col][k] <= cw &&
            configurationsHeights[row][col][k] <= rh) return k;
    error("No Valid configuration for "+row+","+col+"!");
    return −1;
}

function output() {
    println("BEST SOLUTION FOUND >>>>>>>>>>>>>>>>>>>>>>> "+getValue(totalHeight));
    solution = openWrite(solutionfile);
    for [row in 0..nbRows−1] {
        for [col in 0..nbCols−1] {
            print(solution, getOutputConfig(row, col)+" ");
        }
    }
    println(solution);
}
A.8 Table Split with Minimum Column Positioning Changes

Listing A.8: TS-MinCol model for the table layout problem

```plaintext
/* OPL 6.3 Model
* Author: Mihai Bilauca
* Creation Date: 19 Feb 2012 at 11:47:08
* Description: This model minimizes the number of pages that a wide table requires when split and at the same time it minimizes the number of relative column position changes
*/

int pageW = ...; // page width
int NbCols = ...; // column count
range Cols = 1..NbCols; // column range
range Pages = 1..NbCols; // page range; there are maximum NbCols pages
int colW[Cols] = ...; // column widths
int colP[j in Cols]; // the page index for each column j
int a = ...; // flow control: initialized to 1
int b = ...; // flow control: initialized to 0
int obj1Val = ...; // flow control: initialized to -1

execute {
  var p = 0;
  var s = colW[1];
  for (var j=2; j<= Cols.UB; j++) {
    if (s + colW[j] > pageW)
      { p = p+1; s = 0;}
    colP[j] = p;
    s = s + colW[j];
  }
  int pageC = 1 + max(j in Cols) colP[j]; // page count
  dvar int+ pageSel[Pages] in 0..1; // contains 1 if a page is needed
  dvar int+ colIdx[Cols] in Cols; // new column index
  dvar int+ pageIdx[Cols] in Cols; // new page index for each column
```

191
// check if column j1 is placed before column j2

dexpr int posO [ j1 in Cols, j2 in Cols ] = j1 <= j2 - 1;
dexpr int posN [ j1 in Cols, j2 in Cols ] = ( colIdx [ j1 ] <= colIdx [ j2 ] - 1);

dexpr float posDiff = sum ( j1, j2 in Cols: j2 < j1 ) abs ( posO [ j1, j2 ] - posN [ j1, j2 ] );
dexpr int pageCount = sum ( i in Pages ) pageSel [ i ];

minimize a * pageCount + b * posDiff;

subject to {

c1: // do not exceed page width
forall ( p in Pages )
sum ( j in Cols ) ( colW [ j ] * ( p == pageIdx [ j ] ) / pageW ) <= pageSel [ p ];

c2: // maintain relationship between page index and column index
forall ( ordered j1, j2 in Cols )

( pageIdx [ j1 ] <= pageIdx [ j2 ] - 1 ) - ( colIdx [ j1 ] <= colIdx [ j2 ] - 1 ) == 0;

c3: // new column index must have unique values
forall ( ordered j1, j2 in Cols )

colIdx [ j1 ] != colIdx [ j2 ];

// if the minimum page count obj1Val has been determined
// maintain this value for subsequent searches
ct4: if ( obj1Val > 0 ) pageCount == obj1Val;
}


execute DISPLAY {

writeln ( "Page count = " , pageCount );
writeln ( "colW = " , colW );
writeln ( "colP = " , colP );
writeln ( "" );
writeln ( "Page optim = " , pageCount );
writeln ( "PosDiff = " , posDiff );
writeln ( "colsIdx = " , colIdx );
writeln ( "pageIdx = " , pageIdx );
writeln ( "pages = " , pages );
}

}
main {
writeln("Page count = ", thisOplModel.pageC);
writeln("colW = " , thisOplModel.colW);
writeln("colP = " , thisOplModel.colP);
thisOplModel.generate();
cplex.solve();
var obj1_val = cplex.getObjValue();
writeln("Phase 1: Page optim: ", thisOplModel.pageCount, " Pos diff: ",
       thisOplModel.posDiff, " Pages: " , thisOplModel.pages);
writeln("colIdx = " , thisOplModel.colIdx);
cplex.clearModel();
var opl1 = new IloOplModel(thisOplModel.modelDefinition, cplex);
var data1 = thisOplModel.dataElements;
data1.a = 0;
data1.b = 1;
data1.obj1Val = obj1_val;
opl1.addDataSource(data1);
opl1.generate();
cplex.solve();
       opl1.posDiff, " Pages: " , opl1.pages);
writeln("colIdx = " , opl1.colIdx);
}

A.9 Table Split with Minimum Group Splits

Listing A.9: TS-MinGroup model for the table layout problem

1 /*******************************************************************************************/
2 * OPL 12.3 Model
3 * Author: Mihai Bilauca
4 * Creation Date: 28 Feb 2012 at 11:09:23
5 * Description: Minimizes the number of pages that a table can be split
6 * while minimizing the number of column groups that are split
7 /**********************************************************************************************/
8
9 int pageW = ...;
10 int NbCols = ...; // column count
11 range Cols = 1..NbCols; // column count range
12 range Pages = 1..NbCols; // there are maximum NbCols pages
13 int colW[Cols] = ...; // column widths
14 int colG[Cols] = ...; // column groups
15
16 int a = ...; // initialized to 1
17 int b = ...; // initialized to 0
18 int obj1Val = ...; // initialized to -1
19
20 {int} groups = {colG[j] | j in Cols}; // groups set
21
22 // store the first column of the group
23 int gFirstCol[g in groups] = first({j | j in Cols : colG[j] == g});
24
25 int colP[j in Cols] = 1;
26
27 execute {
28   var p = 1;
29   var s = colW[1];
30   for (var j=2; j<= Cols.UB; j++) {
31     if (s + colW[j] > pageW)
32       { p = p+1; s = 0;}
33     colP[j] = p;
34     s = s + colW[j];
35   }
36 }
```plaintext
dvar int+ pageSel[Pages] in 0..1; // contains 1 if a page is needed
dvar int+ pageIdx[Cols] in Cols; // new page index for each column

// counts how many columns of a group are on a different page than the first group's column
dexpr int gSplit[g in groups] = sum(j in Cols: colG[j] == g )

  (pageIdx[j] != pageIdx[gFirstCol[g]]) ;
dexpr int gSplitCount = sum(g in groups) (gSplit[g] >= 1);
dexpr int pageCount = sum(p in Pages) pageSel[p];

minimize a * pageCount + b * gSplitCount;

subject to {
  ct1: // do not exceed page width
  forall(p in Pages)
  sum(j in Cols) (colW[j] * (p == pageIdx[j]) / pageW) <= pageSel[p];

  // if the minimum page count obj1Val has has been determined
  // maintain this value for subsequent searches
  ct4: if (obj1Val >= 0) pageCount == obj1Val;
};

{ int} pagec[p in Pages] = {j | j in Cols : pageIdx[j]==p};

main {
  writeln("Page W : ", thisOplModel.pageW);
  writeln("Columns : ", thisOplModel.colW);
  writeln("Groups : ", thisOplModel.colG);
  writeln("Pages : ", thisOplModel.colP);
  writeln();
  thisOplModel.generate();
  cplex.solve();
  var obj1_val = cplex.getObjValue();
  writeln("Phase 1: Page count: ", thisOplModel.pageCount,
          " Split groups: ", thisOplModel.gSplitCount);
  writeln("Pages : ", thisOplModel.pages);
  writeln("Groups : ", thisOplModel.pageg);
  writeln("Columns: ", thisOplModel.pagec);
  writeln();
}
```

cplex.clearModel();

var opl1 = new IloOplModel(thisOplModel.modelDefinition, cplex);
var data1 = thisOplModel.dataElements;
data1.a = 0;
data1.b = 1;
data1.obj1Val = obj1.val;
opl1.addDataSource(data1);
opl1.generate();
cplex.solve();

writeln("Phase 2: Page count: ", opl1.pageCount,
" Split groups: ", opl1.gSplitCount);
writeln("Pages :", opl1.pages);
writeln("Groups :", opl1.pageg);
writeln("Columns: ", opl1.pagec);
A.10 Sample Data Input File

Listing A.10: Sample Data Input File

```c
/***************************************************************************/
/* OPL 6.3 Data */
/* Author: Mihai Bilauca */
/* Creation Date: 1 Feb 2011 at 11:07:02 */
***************************************************************************/

pageW = 586;

rowSpan = {<4,0,0>};
colSpan = {<1,1,3>, <3,2,4>};

configs = {<0,0, <65,10>>,
<0,0, <43,20>>,
<0,1, <23,10>>,
<0,1, <13,20>>,
<0,2, <183,10>>,
<0,2, <102,20>>,
<0,2, <77,30>>,
<0,2, <56,40>>,
<0,2, <42,50>>,
<0,2, <42,60>>,
<0,3, <62,10>>,
<0,4, <126,10>>,
<0,4, <73,20>>,
<0,4, <50,30>>,
<0,4, <36,40>>,
<0,4, <34,50>>,
<0,4, <34,60>>,
<0,5, <81,10>>,
<0,5, <42,20>>,
<0,6, <133,10>>,
<0,6, <74,20>>,
<0,6, <56,30>>,
<0,6, <46,40>>,
<0,7, <5,10>>,
<0,8, <62,10>>,
... }
```
Appendix B

TDT Layout Control

Specifications

The Table Drawing Tool – TDT – provides an API that allows any table authoring tools to use rendering and optimization functions. Tabular formatting is controlled through a standardised data input stream. The content, formatting parameters and rendering instructions are provided in XML format. This chapter presents the full specifications of the data input format.

While most of the features presented in this document have been implemented in TDT some features require further development or improvements.
Table Drawing Tool

Layout Control Parameters

v.1.0

23 July 2012

Author: Mihai Bilauca

Reviewers: Patrick Healy, David Parnas

Other reviewers: Tom Arbuckle, Markus Clermont, Simon Marr
# Table of Contents

1 Revision Control ................................................................. 4
  1.1 List of changes ................................................................. 4
    1.1.1 Version 0.4 ................................................................. 4
    1.1.2 Version 0.3 ................................................................. 4
    1.1.3 Version 0.2 ................................................................. 4
2 Introduction ........................................................................... 6
3 Preliminaries .......................................................................... 7
  3.1 Drawing tabular expressions .................................................. 7
  3.2 Tabular Expressions Layout ............................................... 7
  3.3 Coordinate Space ............................................................... 8
  3.4 Containers ......................................................................... 8
  3.5 Parameters Inheritance ....................................................... 9
4 Layout Control File .................................................................. 10
  4.1 Document Conventions ...................................................... 10
  4.2 Data Types ......................................................................... 12
  4.3 Layout control parameters root element: TDTS ....................... 13
    4.3.1 Layout Control ............................................................. 13
    4.3.2 Layout Settings ............................................................ 14
    4.3.3 Caption Format ............................................................ 15
    4.3.4 Custom Line Styles ...................................................... 16
    4.3.5 Expression ................................................................. 16
    4.3.6 Caption ................................................................. 17
    4.3.7 Layout ................................................................. 18
    4.3.8 The Box element ........................................................... 20
  4.4 Device control parameters ................................................. 25
    4.4.1 Device element ........................................................... 25
5 Custom data types .................................................................. 26
  5.1 Line Style ......................................................................... 26
    5.1.1 Predefined Line Styles .................................................... 28
    5.1.2 Line Cap Style ............................................................. 28
    5.1.3 Line Join Style ............................................................. 28
    5.1.4 Miter Limit ................................................................. 29
  5.2 Colour ............................................................................... 29
    5.2.1 Predefined Colours ....................................................... 30
  5.3 Numbering Style ............................................................... 31
  5.4 Position ............................................................................ 31
  5.5 Font .................................................................................. 31
  5.6 Font Style ......................................................................... 32
  5.7 Align Vertical ................................................................. 32
  5.8 Align Horizontal .............................................................. 33
  5.9 Text Direction ............................................................... 34
6 The XML structure .................................................................. 35
7 Layout class ............................................................................ 38
8 Device class ............................................................................ 39
9 Exceptions .............................................................................. 41

Mihai Bilauca - 23 July 2012
1 Revision Control

<table>
<thead>
<tr>
<th>Revision</th>
<th>Date</th>
<th>Reviewers</th>
</tr>
</thead>
<tbody>
<tr>
<td>v.0.4</td>
<td>10 January 2008</td>
<td>David Parnas, Simon Marr, Tom Arbuckle</td>
</tr>
<tr>
<td>v.0.3</td>
<td>25 October 2007</td>
<td>David Parnas, Simon Marr</td>
</tr>
<tr>
<td>v.0.2</td>
<td>16 January 2007</td>
<td>David Parnas, Markus Clermont</td>
</tr>
<tr>
<td>v.0.1</td>
<td>7 August 2006</td>
<td>David Parnas</td>
</tr>
</tbody>
</table>

1.1 List of changes

Compared with the previous version this document contains major changes which are summarized below.

1.1.1 Version 0.4

<table>
<thead>
<tr>
<th>Section</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Changes to the Grid element and the Box element</td>
</tr>
</tbody>
</table>

1.1.2 Version 0.3

<table>
<thead>
<tr>
<th>Section</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Clarifications on section 4.2;</td>
</tr>
<tr>
<td></td>
<td>Corrected element names and behavior to match the</td>
</tr>
<tr>
<td></td>
<td>prototype developed;</td>
</tr>
<tr>
<td></td>
<td>Separation of attributes and parameters tables</td>
</tr>
<tr>
<td></td>
<td>for elements.</td>
</tr>
</tbody>
</table>

1.1.3 Version 0.2

<table>
<thead>
<tr>
<th>Section</th>
<th>Change</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>Exceptions have been added for each element</td>
</tr>
<tr>
<td>4.3.4</td>
<td>A new element called <em>CustomLineStyles</em> has</td>
</tr>
<tr>
<td></td>
<td>been introduced. This element is a list of line</td>
</tr>
<tr>
<td></td>
<td>styles defined by the user. These styles are</td>
</tr>
<tr>
<td></td>
<td>identified by their name. These styles can be</td>
</tr>
<tr>
<td></td>
<td>used on any of the layout elements.</td>
</tr>
<tr>
<td>4.3.8</td>
<td>The <em>padding</em> property of a <em>box</em> must be a</td>
</tr>
<tr>
<td></td>
<td>positive float.</td>
</tr>
<tr>
<td>4.3.8</td>
<td>The <em>border</em> property of a <em>box</em> must be a</td>
</tr>
<tr>
<td></td>
<td>positive float.</td>
</tr>
<tr>
<td>4.3.8</td>
<td>If no <em>colour</em> is specified for the <em>border</em> of</td>
</tr>
<tr>
<td></td>
<td>a <em>box</em> element, then the</td>
</tr>
</tbody>
</table>
5.5 The *text* element has been replaced by the *font* element. The *TextAlignment*, *CharacterSpace*, *WordSpace* and *LineSpace* properties have been moved from the *text* element to the *box* element. This is because the box as a particular type of container can have any containers, not only text.

4.3.7 The structure of the *layout* element has been changed.

4.4 The *Document* element described in the previous version of this document has been removed and a more generic concept of *device* has been introduced. Each device implements a predefined set of functions. Also, a more generic *layout* element has been specified. Each *layout* must know how to render itself using a device and how to load its own structure from XML files.

10 *PdfDevice* is now defined as particular type of *device*. The *PdfDevice* creates documents in PDF format.

10.1 The *created*, *modified* and *description* fields have been removed from the *PdfDevice document summary* element. PDF does not support these fields.

11 *Rectangular Layout* has been specified as a particular type of *layout*.

11 The *rectangular layout* now contains only a list of grids instead of *headers and main grid* as previously defined.
2 Introduction

The Table Drawing Tool (TDT) is responsible for displaying tabular expressions (TE).

The tool accepts constraints and suggestions from its users and then displays the table within those constraints.

The design of TDT is on two layers:

- The low level layer is a library that contains the basic formatting functions for TE. We will refer to this library as TDTS (Table Drawing Tool Standard).
- The high level layer is a library of functions for the optimisation of drawing TE. The optimisation layer will be referred as TDTO (Table Drawing Tool Optimisation)

The appearance of a tabular expression (font size, borders, colours, placement of expressions, etc.) is controlled by the user or by a program using TDTS. If used, TDTO will determine an optimised layout for the tabular expression that complies with different criteria. For example, one of the criteria could be “do no split rows on multiple pages”. TDTO will determine this layout and then use TDTS to display it.

Notes:
- This document contains the control parameters only for TDTS.
- The current version of TDTS displays only tabular expressions that are sets of rectangular grids.

TDTS is a library of functions that allows a program to control the display of tabular expressions. TDTS is part of a larger tabular expression tool. There are two input sources for TDTS. One source is the kernel module that provides TDTS with the tabular expression. Another input source is a file in XML format that contains the layout control parameters.
3 Preliminaries

3.1 Drawing tabular expressions

To display a tabular expression two sets of information are required:
- Content Information. This set is provided by the Kernel Module;
- Layout Information. The layout is managed by the Output Tool but the layout parameters are provided by the Input Tool by means of a file in XML format.

NOTE: In this version, the layout control file also contains the content to be displayed in the tabular expressions.

The XML file contains two sets of information:
- The list of tabular expressions to be displayed;
- A device description;

Each expression can be displayed using different layout types. Each layout type is specified using a layout descriptor. For each expression, a layout descriptor as described in section 4.3.7 is required. The expression is read from the Kernel module.

The layout is generated on a device. Each device is specified by a device descriptor as described in section 4.4
- In this version TDTS implements only the Rectangular Layout.
- In this version TDTS implements only a PDF device. This device generates documents in PDF format.
3.2 Tabular Expressions Layout

The mathematical model allows tabular expressions to be expressed in n-dimensional space but we can print only in a bi-dimensional space. A display device can simulate a tri-dimensional drawing system but this is less effective in the case of tabular expressions. Figure 1 describes possible layouts for tabular expressions.

![Figure 1 - Tabular Expressions Layouts. Each tabular expression has an attached caption.](image)

A tabular expression is an indexed set of grids. There is always relationship between the data that the grids contain. To show this relationship the grids must be positioned respecting *arrangement rules.*

3.3 Coordinate Space

TDTS can draw tabular expressions on different devices such as screens or printers. Each of these devices has different resolutions that can vary horizontally or vertically. Each device has a co-ordinate system with origins that can fall in different positions. Thus the same drawing can look differently when displayed on a screen or when displayed on a printer. Even on two different printers the drawing can look different.

To obtain a device independent representation of a tabular expression TDTS implements its own coordinate system with two axes:
- X-axis that extends horizontally to the right;
- Y-axis that extends vertically upwards.

This coordinate system is presented in Figure 2.
The units of this coordinate system are called *drawing units or points*. The length of a *drawing unit* is 1/72 inch. Thus, an inch contains 72 *drawing units or points*. This is the size of a point widely used in typography and implemented in most printers available on the market. However, there is no universal definition of a point.

By choosing this coordinate system TDTS is fully compliant with the PDF format that implements a similar coordinate system.

### 3.4 Containers

A container is an abstract object that can contain other container objects.

This creates a hierarchical structure that is essential in TDTS. A container is said to be the *parent* to its inner containers which are called *child* containers.

All the children of a parent container are called *siblings*.

The element that stays at the basis of TDTS is called *box*. This structure and its control parameters are presented in section 4.3.8. A *box* is a derived type of *container* with a rectangular shape, margins, borders, padding and content.

Figure 3 shows the relative position between parent and children *box* elements.
Figure 3 - Relative positioning of children containers within a parent box.

In the figure above, box A is a *containing box* for box B and C. Box B is *containing box* for box D. The positions of B and C are relative to A while the position of D is relative to B.

### 3.5 Parameters Inheritance

The value of an inherited parameter is given by the value of the same parameter assigned to its parent container.

That means that if having a box A containing another box B that has no value provided for the colour parameter, the value will be inherited from the colour parameter of box A.
4 Layout Control File

The output of TDTS is a PDF file containing the representations of tabular expressions. To generate this file TDTS requires two sets of information:

- Tabular Expressions;
- Layout parameters;

The tabular expressions are provided by the Kernel module while the Layout parameters must be provided as a file in XML format. Throughout this document the XML file is called the “layout control file”.

This section describes the structure of the layout control file, the parameters that must be provided and their values.

4.1 Document Conventions

This document describes each element of the layout control file. This is a file in XML format and therefore it must comply with the XML format conventions. The version is compliant with Document Object model (DOM) Level 2. XML DOM Level 2 specifications are available at [http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/](http://www.w3.org/TR/2000/REC-DOM-Level-2-Core-20001113/)

Each element described in this file contains the following items:

- Element description;
- Element name;
- Parent element name. For elements that can appear as sub-elements of more than one element the Not Applicable (N.A.) is displayed;
- A table with the element’s attributes; Attributes are provided within the element’s tag.
- A table with the element’s parameters. Parameters are in fact sub-elements of that element;
- Exceptions list;
- An example of how this element can be used to control the layout;

In the attributes and parameters tables each attribute/parameter has:

- Name;
- Description;
- Type. One of the data types as defined in Table 1. If the parameter’s type is “Element” it means that the current parameter is an element that can group other sub-elements.
- If the parameter is required or not. If a required element is missing, an exception is thrown.
- Default value. This is the value that is used when the element is not supplied.
In the XML file the value of a parameter must be enclosed between the opening tag and the closing tag of that parameter (e.g. `<FontSize>12</FontSize>`) or it must contain sub-elements if its type is “Element”.

If an element if specified but the value cannot be evaluated to the element’s type an exception is thrown. E.g. `<FontSize>12a</FontSize>` generates a type mismatch exception. `<FontSize></FontSize>` also generates an exception because a null value cannot be converted to an integer. However and empty string can be specified as `<Author></Author>`

Throughout the document a word displayed using the *italic* style is either a parameter or a data type. A parameter printed in **bold** is a required parameter.
4.2 Data Types

TDTS uses the data types described in Table 1 below. The type’s limits are given by a set of constants as defined in Table 2.

Table 1 Data types implemented in TDTS

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
<th>Min Value</th>
<th>Max Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>The <code>byte</code> data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).</td>
<td>MIN_BYTE</td>
<td>MAX_BYTE</td>
</tr>
<tr>
<td>Integer</td>
<td>The <code>integer</code> data type is a 32-bit two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).</td>
<td>MIN_INT</td>
<td>MAX_INT</td>
</tr>
<tr>
<td>Float</td>
<td>The <code>float</code> data type is a single-precision 32-bit floating point.</td>
<td>MIN_FLOAT</td>
<td>MAX_FLOAT</td>
</tr>
<tr>
<td>Char</td>
<td>The <code>char</code> data type is a single 16-bit Unicode character. It has a minimum value of 0 and a maximum value of '\uFFFF' (or 65,535 inclusive).</td>
<td>MIN_CHAR</td>
<td>MAX_CHAR</td>
</tr>
<tr>
<td>String(x)</td>
<td>An array with a maximum integer number of <code>x</code> characters.</td>
<td>N.A.</td>
<td>N.A.</td>
</tr>
<tr>
<td>Boolean</td>
<td>The <code>boolean</code> data type has only two possible values: <code>true</code> and <code>false</code> (not case sensitive)</td>
<td>FALSE</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Enumerated types

The value of an enumerated type variable must be provided by its identifier. The identifier is not case sensitive. E.g. `<LineStyle>Dotted</LineStyle>` or `<LineStyle>DOTTED</LineStyle>`

Table 2 Data type constants

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIN_BYTE</td>
<td>The minimum value of the <code>byte</code> type.</td>
<td>-128</td>
</tr>
<tr>
<td>MAX_BYTE</td>
<td>The maximum value of the <code>byte</code> type.</td>
<td>127</td>
</tr>
<tr>
<td>MIN_CHAR</td>
<td>The minimum value of the <code>char</code> type.</td>
<td>0</td>
</tr>
<tr>
<td>MAX_CHAR</td>
<td>The maximum value of the <code>char</code> type.</td>
<td>65,535</td>
</tr>
<tr>
<td>MIN_INTEGER</td>
<td>The minimum value of the <code>integer</code> type.</td>
<td>-2(^{31})</td>
</tr>
<tr>
<td>MAX_INTEGER</td>
<td>The maximum value of the <code>integer</code> type.</td>
<td>2(^{31}-1)</td>
</tr>
<tr>
<td>MIN_FLOAT</td>
<td>The minimum value of the <code>float</code> type.</td>
<td>2(^{-1029})</td>
</tr>
<tr>
<td>MAX_FLOAT</td>
<td>The maximum value of the <code>float</code> type.</td>
<td>((2-2^{-23})\cdot2^{127})</td>
</tr>
</tbody>
</table>
4.3 Layout control parameters root element: TDTS

The layout control file has two major sections: The Layout Control section that groups information about the layout of the tabular expressions and a Device section that groups parameters for the control of the device being used to render expressions.

To comply with the XML standard the first line of the input must specify the XML version used.

```xml
<?xml version="1.0" encoding="utf-8" ?>
```

The root element for the layout control file is the `<TDTS>` element.

**Element name:**  `<TDTS>`

**Parent element:**  N.A.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LayoutControl</td>
<td>Information regarding the tabular expressions that will be displayed and their layout. This element is described in Section 4.3.1</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>Device</td>
<td>Encodes information about the device used to render the layout. This element is described in Section 4.4.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Exceptions:**

- **EX_REQUIRED_ELEMENT** - if no `<LayoutControl>` or `<Device>` elements are specified;

**Example:**

```xml
<?xml version="1.0" encoding="utf-8" ?>
<TDTS>
  <LayoutControl/>
  <Device/>
</TDTS>
```
4.3.1 Layout Control

This element groups information regarding the tabular expressions and their layout.

Element name:  <LayoutControl>
Parent element:  <TDTD>

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LayoutSettings</td>
<td>Contains general layout settings. Caption styles, custom line styles, etc., are stored in this element. See Section 4.3.2 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>Expressions</td>
<td>The set of expressions that will be displayed. See Section 4.3.5 for details.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_REQUIRED_ELEMENT** - if no expressions are specified;

Example:

```
<LayoutControl>
  <LayoutSettings/>
  <Expressions>
    <Expression/>
  </Expressions>
</LayoutControl>
```

4.3.2 Layout Settings

This element groups general layout settings such as custom styles, lines styles, caption styles, etc. These settings apply to all tabular expressions throughout the document.

Element name:  <LayoutSettings>
Parent element:  <LayoutControl>
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaptionFormat</td>
<td>Describes the format that should apply to all the captions throughout the document. See Section 4.3.3 for details</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>CustomLineStyles</td>
<td>Contains a set of custom line styles that can be used throughout the document. See Section 5.1 for details. Each element that can set a line style can use a line style defined in this set.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:

none

Example:

```
<LayoutSettings>
  <CaptionFormat/>
  <CustomLineStyles>
    <LineStyle/>
  </CustomLineStyles>
</LayoutSettings>
```

4.3.3 Caption Format

Note: Caption format is not implemented in this version of TDTS. It has been introduced for future compatibility.

The `CaptionFormat` element defines the format that should apply to all the captions throughout the document.

Element name: `<CaptionFormat>`

Parent element: `<LayoutSettings>`

Captions are used to describe tabular expressions. Each caption has a label, a number and the text describing the expression. For more information about captions see Section 4.3.6.

The captions format described by this element is applied to all the captions in the document.
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Label</td>
<td>A non empty string expression that will be used to name tabular expressions. Examples are “Expression”, “Function”, etc.</td>
<td>String(50)</td>
<td>Yes</td>
<td>“Expression”</td>
</tr>
<tr>
<td>NumberingStyle</td>
<td>Introduced for future compatibility, the numbering style sets the format of the identifiers attached to each label of a caption. See section 5.3 for further information on NumberingStyle data type.</td>
<td>Numbering Style</td>
<td>Yes</td>
<td>NUMERIC</td>
</tr>
<tr>
<td>Separator</td>
<td>A character that will separate the numbering groups. For example for an inner expression the label number may be “3.4.1”. In this case the separator is the “.” (dot) character.</td>
<td>Char</td>
<td>Yes</td>
<td>“.”</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_REQUIRED_ELEMENT** – if any of the elements are missing;

Example:

```
<CaptionFormat>
  <Label>Function</Label>
  <NumberingStyle>NUMERIC</NumberingStyle>
  <Separator>-</Separator>
</CaptionFormat>
```

### 4.3.4 Custom Line Styles

A list of custom line styles. These styles can be used throughout the document. Each line style that is added to the list must have a unique name.

**Element name:** `<CustomLineStyles>`

**Parent element:** `<LayoutSettings>`

The children elements of the `<CustomLineStyles>` element are `LineStyle` elements as described in Section 5.1
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LineStyle</td>
<td>A set of LineStyle elements</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_REQUIRED_ELEMENT** – if no line styles have been provided;

### 4.3.5 Expression

An expression element groups all the information about the layout of a uniquely identified tabular expression.

**Element name:** `<Expression>`  
**Parent element:** `<Expressions>`

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Each expression has a positive ID which is used to identify the tabular expression E.g. when retrieving its contents from the Kernel module the ID of the expression must be provided.</td>
<td>Integer</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Caption</td>
<td>Contains the Caption layout parameters. See section 4.3.6 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>Layout</td>
<td>This element groups all the layout parameters for a tabular expression. See section 4.3.7 for details</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>
Exceptions:

- **EX_REQUIRED_ELEMENT** – if ID attribute or Layout element are missing;
- **EX_INVALID_VALUE**: if the value provided for the <ID> attribute is outside the required range;

Example:

```xml
<Expression ID="1">
  <Caption></Caption>
  <Layout></Layout>
  <Box></Box>
</Expression>
```

### 4.3.6 Caption

This element groups the information about the caption layout of a tabular expression. The caption of a tabular expression is the text associated with the tabular expression that describes its contents.

**Element name:** `<Caption>`

**Parent element:** `<Expression>`

As presented in Figure 4 the caption consists of:

- Label;
- Number;
- Text;

<table>
<thead>
<tr>
<th>Label</th>
<th>Number</th>
<th>Text</th>
</tr>
</thead>
<tbody>
<tr>
<td>Expression</td>
<td>3.14</td>
<td>The Stack definition</td>
</tr>
</tbody>
</table>

**Figure 4 Caption definition**

The *label* is a text that denotes the type of expression displayed. The label value is set for all the tabular expressions in the `CaptionsFormat` element as defined in section 4.3.3.

When a tabular expression contains inner expressions that are displayed outside its layout the inner expressions are automatically replaced with a reference to their location. The reference text is in fact the caption’s label and number.

The *Number* is automatically generated based on the position of the expression within the expression sequence in the document.
When an expression is displayed a counter is incremented. This counter will be displayed in the *Number* field of the caption. When dealing with inner expressions, a new counter will start for each level within an expression. For example, for the first inner expression of expression number 3 the *Number* will contain the value “3.1”, for the second inner expression it will contain “3.2”, etc.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Position</td>
<td>The position of the caption relative to the layout of the expression. The caption can be placed on top, to the left, right or on the bottom of the expression. By default the caption is placed on top of the expression layout. See section 5.4 for details.</td>
<td>Position</td>
<td>No</td>
<td>TOP</td>
</tr>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Exceptions:**

- **EX INVALID VALUE** is thrown if the value provided for the `<Position>` element cannot be recognized as a valid value;

**Example:**

```xml
<Caption>
  <Position>Top</Position>
  <Box></Box>
</Caption>
```

**4.3.7 Layout**

The layout element groups all the control parameters that apply to all the layouts, independent of their implementation.

To implement a new layout (e.g. circular layout) it is as simple as developing a new class. This class must implement a list of functions in order to be recognized as a *layout*. These functions are specified in *Section 7 Layout class*. The class name must be provided in the `<LayoutClass>` element and all the control parameters must be specified as part of the `<LayoutDescriptor>` element.

**Note:** TDTS implements only one type of layout: Rectangular layout.

**Element name:** `<Layout>`
Parent element:  <Expression>

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>LayoutClass</td>
<td>The class name that implements the layout functions as specified in Section 8</td>
<td>String</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>LayoutDescriptor</td>
<td>This element groups all the parameters defined in layout class.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_REQUIRED_ELEMENT**: if the <LayoutClass> element or the <LayoutDescriptor> is missing;
- **EX_INVALID_CLASS**: if an error occurs while loading the specified class;

Example:

```
<Layout>
  <Box></Box>
  <LayoutClass>RectangularLayout</LayoutClass>
  <LayoutDescriptor></LayoutDescriptor>
</Layout>
```
4.3.8 The Box element

The box element is a rectangular area that has content, padding, border, and margin areas. The image below describes each of the areas.

Because of its rectangular shape, the box element is maybe the most important element in the tabular expressions drawing process. Most elements have a rectangular shape: a drawing surface, a rectangular grid, rectangular cell, etc. The box element has been introduced as a reference type that is used as an individual entity or as part of other types - cell, grid, layout, etc.

Figure 5 - The Box model: Margin, Border, Padding and Content.

The margin is the transparent space used to separate two adjacent boxes. The border space is usually filled with a pattern and creates a visual separation between the content of adjacent boxes. The padding space introduces a space between the border and the content space. The content space is the rectangular area that will be used to display the expression of the cell.

Each property can be inherited or not. If a property of a box A is inherited and not set, then it will get the value of the same property of the box that contains A. If A is not contained in any box then the default value applies. Please see section 3.5 for details about parameters inheritance.
**Element name:** `<Box>`

**Parent element:** N.A.

### Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Specifies the horizontal coordinate in points relative to the containing box.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Y</td>
<td>Specifies the vertical coordinate in points relative to the containing box.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Width</td>
<td>The width property of a box is given by the sum of the widths for the content, padding left and right, borders left and right and margins left and right. The width is measured in points.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Height</td>
<td>The height property of a box is the sum of the heights for the content, padding top and bottom, borders top and bottom and margins top and bottom. The height is measured in points.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>AutoWidth</td>
<td>If TRUE the width of the box is given by the width of the content.</td>
<td>Boolean</td>
<td>No</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>If FALSE the width of the content is specified by the Width property.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>AutoHeight</td>
<td>If TRUE the height of the box is given by the height of the content.</td>
<td>Boolean</td>
<td>No</td>
<td>True</td>
</tr>
<tr>
<td></td>
<td>If FALSE the height of the content is specified by the Height property.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Margin</strong></td>
<td>The margin is the space in points between the box boundaries and the border of the box. The margin is always transparent. The margin can be split up in four segments: Left, Right, Top and Bottom. When the margin parameter is specified its value will apply to all margin segments. float</td>
<td>No 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MarginLeft, MarginRight, MarginTop, MarginBottom</strong></td>
<td>Specifies the Margin parameter for the respective box segment. float</td>
<td>No 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Border</strong></td>
<td>The border is the space in points between the margin and the padding area. Three properties can be specified for the border of a box: • Width; • Colour; • Line Style; The border is drawn on the background of the box. The border can also be specified by four segments Left, Right, Top and Bottom. Element</td>
<td>No 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Border - Width</strong></td>
<td>The ‘pen’ width, measured in points perpendicularly to the pen trajectory. float</td>
<td>&gt; = 0 No 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Border - Colour</strong></td>
<td>Specifies the colour of the border area. Please see section 5.2 for details on the colour type. If no border colour is specified the value of the Colour property is used. Colour</td>
<td>No BLACK</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Border - Style</strong></td>
<td>Specifies the line style of the border area. See section 5.1 for details on the definition of line styles. LineStyle</td>
<td>No SOLID</td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BorderLeft, BorderRight, BorderTop, BorderBottom</strong></td>
<td>Specifies the Border parameter for the respective box segment. Element</td>
<td>No 0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Parameter</td>
<td>Description</td>
<td>Type</td>
<td>Required</td>
<td>Default</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------</td>
<td>----------</td>
<td>---------</td>
</tr>
<tr>
<td>Padding</td>
<td>The space measured in points between the border and the content of a box.</td>
<td>float</td>
<td>No</td>
<td>1</td>
</tr>
<tr>
<td>PaddingLeft, PaddingRight, PaddingTop, PaddingBottom</td>
<td>Specifies the Padding parameter for the respective box segment.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>Colour</td>
<td>Specifies the colour that will be used to display the contents of the box.</td>
<td>Colour</td>
<td>No</td>
<td>BLACK</td>
</tr>
<tr>
<td>BkColour</td>
<td>Specifies the background colour of the box.</td>
<td>Colour</td>
<td>No</td>
<td>TRANSPARENT</td>
</tr>
<tr>
<td>AlignHorizontal</td>
<td>A numerical proportion value between 0 and 1 specifying the space from the left content edge where the content will be positioned. Specifies where on the horizontal direction the content of the box will be aligned when its width is smaller than the box’s width.</td>
<td>float</td>
<td>No</td>
<td>LEFT</td>
</tr>
<tr>
<td>AlignVertical</td>
<td>A numerical proportion value between 0 and 1 specifying the space from the top content edge where the content will be positioned. Specifies where on the vertical direction the content of the box will be aligned when its height is smaller than the box’s height.</td>
<td>float</td>
<td>No</td>
<td>TOP</td>
</tr>
<tr>
<td>Font</td>
<td>Groups all the parameters that control the appearance of the text displayed in the box.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>TextDirection</strong></td>
<td>The direction of the text. Specifies how the text is going to be oriented in a box. See Figure 8 for examples.</td>
<td><strong>TextDirection</strong></td>
<td>No</td>
<td>HORIZONTAL</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-----</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>TextRotation</strong></td>
<td>The angle that the text is rotated</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>CharacterSpace</strong></td>
<td>Specifies the space in points used in addition to the default space between adjacent characters.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>WordSpace</strong></td>
<td>Specifies the space in points used in addition to the default space between adjacent words.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>LineSpace</strong></td>
<td>Specifies the space in points used in addition to the default space between adjacent lines of text.</td>
<td>float</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>

**Box Example:**

```
<Box>
    <X></X>
    <Y></Y>
    <Width></Width>
    <Height></Height>
    <AutoWidth></AutoWidth>
    <AutoHeight></AutoHeight>
    <Margin></Margin>
    <MarginLeft></MarginLeft>
    <MarginRight></MarginRight>
    <MarginTop></MarginTop>
    <MarginBottom></MarginBottom>
    <Border>
        <Width>0.2</Width>
        <Colour>Red</Colour>
        <LineStyle>Dotted</LineStyle>
    </Border>
    <BorderLeft>
        <Width></Width>
        <Colour></Colour>
        <LineStyle></LineStyle>
    </BorderLeft>
    <BorderTop>
        <Width></Width>
        <Colour></Colour>
        <LineStyle></LineStyle>
    </BorderTop>
    <BorderRight>
        <Width></Width>
        <Colour></Colour>
        <LineStyle></LineStyle>
    </BorderRight>
</Box>
```
4.4 Device control parameters

The second part of the layout control file controls the device used to render the expressions. This section specifies all the parameters that will allow the user to set device specific parameters.

4.4.1 Device element

The device element groups all the control parameters that apply to all the devices, independent of their implementation.

To implement a new device it is as simple as developing a new class. This class must implement a list of functions in order to be recognized as a device. These functions are specified in Section 8 Device class. The class name must be provided in the <DeviceClass> element and all the control parameters must be specified as part of the <DeviceDescriptor> element.

Note: TDTS implements only the PdfDevice class which generates files in Adobe PDF format

Element name:     <Device>
Parent element:   <TDTS>

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DeviceClass</td>
<td>The class name that implement the device.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>MaxPagesNo</td>
<td>The maximum number of pages that this device can handle.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>DeviceDescriptor</td>
<td>This element contains all the elements that are defined for each device. If a new device is added, this element should contain all the device’s parameters defined as sub-elements.</td>
<td>Element</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:
- **EX_REQUIRED_ELEMENT**: if any of the elements are missing;
- **EX_INVALID_CLASS**: if an error occurs while loading the specified class;

Example:
<Device>
    <DeviceClass>PdfDevice</DeviceClass>
    <MaxPagesNo>100</MaxPagesNo>
    <DeviceDescriptor></DeviceDescriptor>
</Device>
5 Custom data types

In TDTS several custom data types are defined. In this section each custom data type is presented.

5.1 Line Style

A line style element groups the parameters that define a new line style - the aspect of a line.

**Element name:** <LineStyle>
**Parent element:** N.A.

The line style value can be provided by:
- A predefined line style name as listed in Table 3;
- A line style name defined in the custom line styles section. See section 4.3.4 for details on defining custom line styles;
- Defined by a set of parameters.

The following parameters define a new line style:
- 1. Pattern
  - Dash array: an array of integers;
  - Phase: an integer.
- 2. Line Cap Style: type integer;
- 3. Line Join Style: type integer;
- 4. Miter Limit: type float;

The pattern is defined as a *dash array* and a *phase*. The *dash array* contains numbers that specify the lengths of alternating dashes and gaps. The *phase* specifies the starting point of the pattern.

When drawing a line with a line style defined by *dash array* and a phase the dash array is cycled adding dashes and gaps. When the length of the array equals the *phase* the drawing of the line begins.

For example:
- Dash array: [2 3]
- Phase: 11
  - First, the array is cycled: 2 dashes, 3 gaps, 2 dashes, 3 gaps, 1 dash = 11 (equals the phase).

Then the drawing starts: 1 dash, 3 gaps, 2 dashes, 3 gaps, 2 dashes, …etc. See Figure 5.
A few line styles templates can be defined as macros. For example *dashed*, *dotted*, etc.

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>StyleName</strong></td>
<td>A string that uniquely identifies the line style within the set of line styles.</td>
<td>String(50)</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>DashArray</strong></td>
<td>An array with two integer values separated by a space that specifies the lengths of alternating dashes and gaps.</td>
<td>String</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>Phase</strong></td>
<td>Specifies the starting point of the pattern</td>
<td>Byte</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td><strong>LineCap</strong></td>
<td>Specifies the decoration applied at the ends of a dashed or unclosed line.</td>
<td>LineCap</td>
<td>Yes</td>
<td>BUTT_CAP</td>
</tr>
<tr>
<td><strong>LineJoin</strong></td>
<td>Specifies the shape that will be used when two consecutive segments of a path meet at an angle</td>
<td>LineJoin</td>
<td>Yes</td>
<td>MITER_JOIN</td>
</tr>
<tr>
<td><strong>MiterLimit</strong></td>
<td>Specifies a ratio between the miter length and the line width. See section 5.1.4 for more details.</td>
<td>Float</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>

**Exceptions:**

- EX_INVALID_LS_NAME
- EX_REQUIRED_LS_NAME
- EX_INVALID_LS_NAME_DEFAULT
Example:

```xml
<LineStyle>
  <StyleName>DashedLarge</StyleName>
  <DashArray>5 5</DashArray>
  <Phase>3</Phase>
</LineStyle>
```

Note: In this version LineCap, LineJoin and MiterLimit are ignored.

### 5.1.1 Predefined Line Styles

This is the set of predefined line styles:

<table>
<thead>
<tr>
<th>Line style name</th>
<th>Dash array</th>
<th>Phase</th>
</tr>
</thead>
<tbody>
<tr>
<td>solid</td>
<td>1,0</td>
<td>0</td>
</tr>
<tr>
<td>dotted</td>
<td>1,1</td>
<td>0</td>
</tr>
<tr>
<td>dashed</td>
<td>3,3</td>
<td>0</td>
</tr>
</tbody>
</table>

### 5.1.2 Line Cap Style

The line cap style specifies the decoration applied at the ends of a dashed or unclosed line. The line cap style is defined as in the Adobe PDF Specifications [2]

Name: LineCapStyle
Type: Byte

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Appearance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BUTT_CAP</td>
<td>0</td>
<td>![Butt Cap Image]</td>
<td>Butt cap</td>
</tr>
<tr>
<td>ROUND_CAP</td>
<td>1</td>
<td>![Round Cap Image]</td>
<td>Round cap</td>
</tr>
<tr>
<td>SQUARE_CAP</td>
<td>2</td>
<td>![Square Cap Image]</td>
<td>Square Cap</td>
</tr>
</tbody>
</table>
5.1.3 Line Join Style

The line join style specifies the shape the will be used when two consecutive segments of a path meet at an angle. The line join style is defined as in the Adobe PDF Specifications [2].

Name: LineJoinStyle
Type: Byte

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Appearance</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MITER_JOIN</td>
<td>0</td>
<td><img src="image" alt="Miter Join" /></td>
<td>Miter Join</td>
</tr>
<tr>
<td>ROUND_JOIN</td>
<td>1</td>
<td><img src="image" alt="Round Join" /></td>
<td>Round Join</td>
</tr>
<tr>
<td>BEVEL_JOIN</td>
<td>2</td>
<td><img src="image" alt="Bevel Join" /></td>
<td>Bevel Join</td>
</tr>
</tbody>
</table>

5.1.4 Miter Limit

The *miter limit* is a ratio between the miter length and the line width. When two segments that have a miter join style meet at a very sharp angle there are chances that the miter will be extended beyond the width of the line.
5.2 Colour

The Colour Type is a structure with 3 integer variables R, G, B with a range between 0 and 255. These variables represent the amount of Red, Blue and Green that a colour contains. Using these 3 values the user can specify up to 16,777,215 colours.

If the ColourName is specified it overwrites the values specified in the R, G, B elements.

Please see section 5.2.1 for a list of predefined colours.

Element name: <Colour>
Parent element: N.A.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>ColourName</td>
<td>A predefined colour constant as defined in section 5.2.1.</td>
<td>String</td>
<td>No</td>
<td>“BLACK”</td>
</tr>
<tr>
<td>R</td>
<td>An integer value with a range between 0 and 255 specifying the amount of red in the combination that forms a colour.</td>
<td>Byte</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>G</td>
<td>An integer value with a range between 0 and 255 specifying the amount of green in the combination that forms a colour.</td>
<td>Byte</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>An integer value with a range between 0 and 255 specifying the amount of blue in the combination that forms a colour.</td>
<td>Byte</td>
<td>No</td>
<td>0</td>
</tr>
</tbody>
</table>

Example:

```xml
<Colour>
  <ColourName>RED</ColourName>
  <R>125</R>
  <G>0</G>
  <B>0</B>
</Colour>
```
5.2.1 Predefined Colours

This is the set of predefined colour constants:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Values</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R</td>
</tr>
<tr>
<td>BLACK</td>
<td>0</td>
</tr>
<tr>
<td>LIGHTGRAY</td>
<td>250</td>
</tr>
<tr>
<td>RED</td>
<td>125</td>
</tr>
<tr>
<td>GREEN</td>
<td>0</td>
</tr>
<tr>
<td>BLUE</td>
<td>0</td>
</tr>
</tbody>
</table>

5.3 Numbering Style

The NumberingStyle is an enumerated data type with integer values that represent the style of the identifiers attached to a caption. The table below shows the values allowed for the numbering style.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>NUMERIC</td>
<td>0</td>
<td>Numeric numbering</td>
<td>1, 2, 3, 4,…</td>
</tr>
<tr>
<td>ROMAN</td>
<td>1</td>
<td>Roman numbering</td>
<td>I, II, III, IV,…</td>
</tr>
<tr>
<td>CHAR</td>
<td>2</td>
<td>Character numbering</td>
<td>A, B, C, D,…</td>
</tr>
<tr>
<td>LOW_I</td>
<td>3</td>
<td>“i” character numbering</td>
<td>i, ii, iii,…</td>
</tr>
</tbody>
</table>

Note: In this version TDT implements only the NUMERIC numbering style.

5.4 Position

The Position data type is an enumerated data type with integer values that defines the positions allowed for a graphical element relative to another element on a page.

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDDEN</td>
<td>0</td>
<td>Do not display this element</td>
</tr>
<tr>
<td>TOP</td>
<td>1</td>
<td>The top position</td>
</tr>
<tr>
<td>RIGHT</td>
<td>2</td>
<td>The right position</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>3</td>
<td>The bottom position</td>
</tr>
<tr>
<td>LEFT</td>
<td>4</td>
<td>The left position</td>
</tr>
</tbody>
</table>
5.5 Font

The font element encodes all the parameters that control the display of the text within a box element.

Element name:  <Font>
Parent element:  N.A.

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FontName</td>
<td>The name of the font used.</td>
<td>String</td>
<td>No</td>
<td>“Times New Roman”</td>
</tr>
<tr>
<td>FontSize</td>
<td>The size in points of the font.</td>
<td>Integer</td>
<td>No</td>
<td>10</td>
</tr>
<tr>
<td>FontStyle</td>
<td>The style of the font. E.g. Italic, Bold, Bold-Italic, etc. as defined in FontStyle.</td>
<td>FontStyle</td>
<td>No</td>
<td>NORMAL</td>
</tr>
</tbody>
</table>

If a font with the name provided in the element is not found then the default value applies.

If none of the elements are provided then the result is the default font.

Exceptions:

- EX_INVALID_INTEGER: when the value provided for the <FontSize> element cannot be evaluated to an integer;
- EX_INVALID_VALUE: if the value provided for the <FontStyle> cannot be evaluated.

Example:

```
<Font>
  <FontName>ARIAL</FontName>
  <FontSize>12</FontSize>
  <FontStyle>BOLD</FontStyle>
</Font>
```
5.6 Font Style

The FontStyle data type is an enumerated data type with integer values that defines the style applied to the text.

Name: FontStyle
Type: Byte

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NORMAL</td>
<td>0</td>
<td>Normal style</td>
</tr>
<tr>
<td>BOLD</td>
<td>1</td>
<td>Bold font style</td>
</tr>
<tr>
<td>ITALIC</td>
<td>2</td>
<td>Italic font style</td>
</tr>
<tr>
<td>BOLD_ITALIC</td>
<td>3</td>
<td>Bold Italic font style</td>
</tr>
</tbody>
</table>

When the limit is exceeded the join becomes a bevel join style. More details in [2 pages 185-188]

5.7 Align Vertical

The AlignVertical data type is an enumerated data type with float values between 0 and 1 that defines where on the vertical direction the content of a box should be placed when the height of the box is greater than the height of the content. The predefined values are proportional to the box’s vertical space.

Name: AlignVertical
Type: Float

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TOP</td>
<td>0.00</td>
<td>Align the content to the top margin of the box</td>
</tr>
<tr>
<td>MIDDLE</td>
<td>0.50</td>
<td>Align the content in the middle of the box</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>1.00</td>
<td>Align the content to the bottom margin of the box</td>
</tr>
</tbody>
</table>

Figure 7 shows the position of the content within a box. Examples for different vertical and horizontal alignment values are displayed.
5.8 Align Horizontal

The \textit{AlignHorizontal} data type is an enumerated data type with float values between 0 and 1 that defines where on the horizontal direction the content of a box should be placed when the width of the box is greater than the width of the content. The predefined values are proportional to the box’s horizontal space.

Name: \textit{AlignHorizontal}  
Type: \textit{Float}  

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LEFT</td>
<td>0</td>
<td>Align the content to the left margin of the box</td>
</tr>
<tr>
<td>CENTRE</td>
<td>0.5</td>
<td>Align the content in the centre of the box</td>
</tr>
<tr>
<td>RIGHT</td>
<td>1</td>
<td>Align the content to the right margin of the box</td>
</tr>
</tbody>
</table>

Figure 7 shows the position of the content within a box. Different values for vertical and horizontal alignment are given.

\begin{figure}[h]
\centering
\includegraphics[width=0.8\textwidth]{figure7.png}
\caption{Figure 7 - Vertical Alignment and Horizontal Alignment}
\end{figure}

5.9 Text Direction

The \textit{TextDirection} data type is an enumerated data type with integer values that defines the direction of the text within a box. The text can be displayed on the horizontal direction or on the vertical direction top to bottom or bottom to top.
Name: \textit{TextDirection}  
Type: \textit{Byte}  

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HORIZONTAL</td>
<td>0</td>
<td>Align the content to the left margin of the box</td>
</tr>
<tr>
<td>TOP TO BOTTOM</td>
<td>1</td>
<td>Align the content in the centre of the box</td>
</tr>
<tr>
<td>BOTTOM TO TOP</td>
<td>2</td>
<td>Align the content to the right margin of the box</td>
</tr>
</tbody>
</table>

Figure 8 below shows the effect of different text directions within a box.

\begin{figure}[h]
\centering
\includegraphics[width=0.5\textwidth]{figure8.png}
\caption{Text Direction: Horizontal, Top to Bottom, Bottom to Top}
\end{figure}
6 The XML structure

This section shows an example of XML layout control file.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<TDTS>
  <LayoutControl>
    <LayoutSettings>
      <CaptionFormat>
        <Label>Expression</Label>
        <NumberingStyle>NUMERIC</NumberingStyle>
        <Separator>-</Separator>
      </CaptionFormat>
      <CustomLineStyles>
        <LineStyle>
          <StyleName>Dashed1</StyleName>
          <DashArray>5 3</DashArray>
          <Phase>3</Phase>
        </LineStyle>
        <LineStyle>
          <StyleName>Dashed2</StyleName>
          <DashArray>5 5</DashArray>
          <Phase>3</Phase>
        </LineStyle>
      </CustomLineStyles>
    </LayoutSettings>
    <Expressions>
      <Expression ID="1">
        <Box>
          <X>50</X>
          <Y>500</Y>
          <Height>42</Height>
          <Colour>Red</Colour>
          <MarginLeft>1</MarginLeft>
          <MarginRight>1</MarginRight>
          <MarginTop>1</MarginTop>
          <MarginBottom>1</MarginBottom>
          <Border>
            <Width>0.2</Width>
            <Colour>Blue</Colour>
            <LineStyle>Dotted</LineStyle>
          </Border>
          <BorderTop>
            <Width>0.1</Width>
          </BorderTop>
          <BorderBottom>
            <Width>1</Width>
            <LineStyle>
              <DashArray>4 2</DashArray>
              <Phase>3</Phase>
            </LineStyle>
          </BorderBottom>
          <BorderRight>
            <Width>3</Width>
        </Box>
      </Expression>
    </Expressions>
  </LayoutControl>
</TDTS>
```
7 Layout class

A layout class is an abstract structure that contains data and functions used to represent the graphical structure of tabular expressions. This class implements functions that allow the reading of its own data from an XML file and functions to render the tabular expression on a device. A device is another abstract structure that implements functions that render graphical elements. For more information about the device class see Section 8.

This class also represents a “contract” that any new layout class should sign for. For example, if it is required to print a tabular expression using a new circular layout then a new class should be developed. This class must implement all the functions described in Table 4.

<table>
<thead>
<tr>
<th>Returns</th>
<th>Name</th>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Render</td>
<td>device</td>
<td>Device</td>
<td>Renders the content of the layout using the provided Device;</td>
</tr>
<tr>
<td>-</td>
<td>loadLayout</td>
<td>e</td>
<td>org.w3c.dom.Element</td>
<td>Loads the layout’s parameters from the provided XML element;</td>
</tr>
</tbody>
</table>

Implementation notes:
- Currently TDTS implement only the rectangular layout.
- TDTS has been developed in Java 1.6. In this implementation the new class must be placed in the /layouts folder of the tool.
8 Device class

A *device* is an abstract structure that implements functions that render graphical elements to a specified output device: file, screen, printer, etc.

This class also represents a “contract” that any new *device* class should sign for. For example, if it is required to render data on a new type of device (e.g. printer) then a new class should be developed. This class must implement all the functions described in Table 5.

<table>
<thead>
<tr>
<th>Returns</th>
<th>Name</th>
<th>Parameters</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-</td>
<td>openDevice</td>
<td>-</td>
<td>-</td>
<td>Open the device for rendering content; This function must be called before any other function.</td>
</tr>
<tr>
<td>-</td>
<td>closeDevice</td>
<td>-</td>
<td>-</td>
<td>Closes the device. When this function is called no other rendering functions can be called without calling the <em>openDevice</em> function.</td>
</tr>
<tr>
<td>boolean</td>
<td>addPage</td>
<td>page box</td>
<td>boolean</td>
<td>Adds a new page to the list of pages. Returns whether the operation succeeded or not.</td>
</tr>
<tr>
<td>boolean</td>
<td>deletePage</td>
<td>index integer</td>
<td>boolean</td>
<td>Removes the page with the provided index from the list of pages. Returns whether the operation succeeded or not.</td>
</tr>
<tr>
<td>-</td>
<td>setColour</td>
<td>r byte</td>
<td>byte</td>
<td>Sets the device’s current stroking colour by passing the red, green and blue components of a colour; r is the red value;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g byte</td>
<td>byte</td>
<td>The green value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b byte</td>
<td>byte</td>
<td>The blue value</td>
</tr>
<tr>
<td>-</td>
<td>setFillColour</td>
<td>r int</td>
<td>int</td>
<td>Sets the device’s fill colour;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>g int</td>
<td>int</td>
<td>The green value</td>
</tr>
<tr>
<td></td>
<td></td>
<td>b int</td>
<td>int</td>
<td>The blue value</td>
</tr>
<tr>
<td>-</td>
<td>setLineWidth</td>
<td>width float</td>
<td>float</td>
<td>Sets the current line with;</td>
</tr>
<tr>
<td>-</td>
<td>setLineStyle</td>
<td>array float[]</td>
<td>float</td>
<td>Sets the current line style. array - the array of dashes and gaps;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>phase float</td>
<td>float</td>
<td>The phase of the style;</td>
</tr>
<tr>
<td>-</td>
<td>setLineCap</td>
<td>style int</td>
<td>int</td>
<td>Sets the current line cap;</td>
</tr>
<tr>
<td>-</td>
<td>setFont</td>
<td>fontName string</td>
<td>string</td>
<td>Sets the current font name</td>
</tr>
<tr>
<td>-</td>
<td>setFontSize</td>
<td>fontSize int</td>
<td>int</td>
<td>Sets the current font size</td>
</tr>
<tr>
<td>-</td>
<td>setFontAndSize</td>
<td>fontName string</td>
<td>int</td>
<td>Sets both the current font name and</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fontSize int</td>
<td>int</td>
<td>Font size</td>
</tr>
<tr>
<td>-</td>
<td>setFontStyle</td>
<td>fontStyle int</td>
<td>int</td>
<td>Sets the font style</td>
</tr>
<tr>
<td>-</td>
<td>setAlignmentHorizontal</td>
<td>alignment float</td>
<td>float</td>
<td>Sets the current horizontal alignment</td>
</tr>
<tr>
<td>-</td>
<td>setAlignmentVertical</td>
<td>alignment float</td>
<td>float</td>
<td>Sets the current vertical alignment</td>
</tr>
<tr>
<td>-</td>
<td>setCharSpace</td>
<td>space float</td>
<td>float</td>
<td>Sets the current character space</td>
</tr>
<tr>
<td>-</td>
<td>setWordSpace</td>
<td>space float</td>
<td>float</td>
<td>Sets the current words space</td>
</tr>
<tr>
<td>-</td>
<td>setLineSpace</td>
<td>space float</td>
<td>float</td>
<td>Sets the current line space</td>
</tr>
<tr>
<td>-</td>
<td>drawLine</td>
<td>x1 float</td>
<td>float</td>
<td>Draws a line using the current line width, line style and colour from the point specified by the</td>
</tr>
<tr>
<td>Returns</td>
<td>Name</td>
<td>Parameters</td>
<td>Type</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------</td>
<td>------------</td>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>x1,y1 to the point x2,y2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y1 float</td>
<td>Initial point’s vertical coordinate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>x2 float</td>
<td>Final point’s horizontal coordinate</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>y2 float</td>
<td>Final point’s vertical coordinate</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>moveTo</td>
<td>x</td>
<td>float</td>
<td>Moves the current drawing point of the current path to the specified coordinates; x is horizontal coordinate;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>float</td>
<td>Vertical coordinate</td>
</tr>
<tr>
<td>-</td>
<td>lineTo</td>
<td>x</td>
<td>float</td>
<td>Appends to the current path a straight line segment from the current point. The new current point is (x, y). x is horizontal coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>float</td>
<td>Vertical coordinate</td>
</tr>
<tr>
<td>-</td>
<td>Stroke</td>
<td>-</td>
<td>-</td>
<td>Strokes the current path using the current line width, line style and current colour</td>
</tr>
<tr>
<td>-</td>
<td>drawRectangle</td>
<td>x</td>
<td>float</td>
<td>Draws a rectangle using the current line width, line style and colour from the point specified by the x1,y1 to the point x2,y2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y1 float</td>
<td>Initial point’s vertical coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x2 float</td>
<td>Final point’s horizontal coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>y2 float</td>
<td>Final point’s vertical coordinate</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>drawRectangleAndFill</td>
<td>x</td>
<td>float</td>
<td>Draws a rectangle using the current line width, line style and colour from the point specified by the x1,y1 to the point x2,y2. The rectangle is filled with the current fill colour;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y1 float</td>
<td>Initial point’s vertical coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>x2 float</td>
<td>Final point’s horizontal coordinate</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>y2 float</td>
<td>Final point’s vertical coordinate</td>
<td></td>
</tr>
<tr>
<td>-</td>
<td>drawText</td>
<td>text</td>
<td>String</td>
<td>Draws the specified String at the position specified by x,y in an area having the specified width and height. The text is rotated by the specified angle;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>x</td>
<td>float</td>
<td>Horizontal coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>y</td>
<td>float</td>
<td>Vertical coordinate</td>
</tr>
<tr>
<td></td>
<td></td>
<td>width</td>
<td>float</td>
<td>Text area width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>height</td>
<td>float</td>
<td>Text area height</td>
</tr>
<tr>
<td></td>
<td></td>
<td>angle</td>
<td>float</td>
<td>The angle to rotate the text</td>
</tr>
<tr>
<td>-</td>
<td>drawImage</td>
<td>imagePath</td>
<td>String</td>
<td>Adds the image located at the specified location to the current page</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>getImageDimension</td>
<td>imagePath</td>
<td>String</td>
<td>Returns the dimension in points of the image with the location given in the imagePath parameter;</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>loadConfiguration</td>
<td>e</td>
<td>org.w3c.dom.Element</td>
<td>Loads the device’s configuration parameters from the provided XML element;</td>
</tr>
</tbody>
</table>

**Implementation notes:**

- Currently TDTS implement only the \texttt{PdfDevice}. This class generates a file in PDF format;
- TDTS has been developed in Java 1.6. In this implementation the new class must be placed in the /devices folder of the tool.
9 Exceptions

The table below shows the list of exceptions that are thrown by different elements of TDTS.

Table 6 List of exceptions thrown by the TDTS

<table>
<thead>
<tr>
<th>Exception Identifier</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EX_REQUIRED_ELEMENT</td>
<td>A required value has not been provided.</td>
</tr>
<tr>
<td>EX_REQUIRED_LS_NAME</td>
<td>When defining a new line style, the style name must be provided.</td>
</tr>
<tr>
<td>EX_INVALID_LS_NAME</td>
<td>When defining a new line style, the style name has already been used to define another line style. Line style names must be unique</td>
</tr>
<tr>
<td>EX_INVALID_LS_NAME_DEFAULT</td>
<td>When defining a new line style, the style name is used already by a default line style.</td>
</tr>
<tr>
<td>EX_INVALID_INTEGER</td>
<td>The value provided cannot be converted to a value of type Integer.</td>
</tr>
<tr>
<td>EX_INVALID_FLOAT</td>
<td>The value provided cannot be converted to a value of type Float</td>
</tr>
<tr>
<td>EX_INVALID_BOOLEAN</td>
<td>The value provided cannot be converted to a value of type Boolean.</td>
</tr>
<tr>
<td>EX_INVALID_VALUE</td>
<td>The expression provided is not recognized as a valid value for a parameter.</td>
</tr>
<tr>
<td>EX_DUPLICATE_INDEX</td>
<td>The index provided has already been used. The indexes must be unique.</td>
</tr>
<tr>
<td>EX_INVALID_CLASS</td>
<td>The specified class could not be found or could not be loaded.</td>
</tr>
</tbody>
</table>

Implementation notes:

- TDTS has been developed in Java 1.6. It takes advantage of the resources management features. The exception messages have been defined in resources files (text files with ‘.properties’ extension). Messages can be amended by simply editing these text files.
10 PDF Device

The PdfDevice class is a particular type of device and therefore it implements all the functions specified in Section 8. It allows rendering data in a PDF file.

All the configuration parameters of this class must be specified as sub-elements of the `<DeviceDescriptor>` element defined in section 4.4.1.

**Element name:** `<DeviceDescriptor>`  
**Parent element:** `<Device>`

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>The name of the output file.</td>
<td>String</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>Summary</td>
<td>Groups the metadata information for the document such as the author, the document title, etc.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>InitialSettings</td>
<td>Groups the parameters that encode the display options when a file is first opened such as the page that will be displayed when the document is opened and its zoom value.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>Pages</td>
<td>The set of the pages with their layout control parameters.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

**Exceptions:**
- **EX_REQUIRED_ELEMENT** if the `<FileName>` element is missing;

**Example:**

```
<DeviceDescriptor>
  <FileName>Output.pdf</FileName>
  <Summary></Summary>
  <InitialSettings></InitialSettings>
  <Pages></Pages>
</DeviceDescriptor>
```

10.1 Summary

The Summary element groups the metadata information for the document.
Element name: <Summary>
Parent element: <DeviceDescriptor>

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>The title of the document</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Author</td>
<td>The document author’s name</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Subject</td>
<td>The subject of the document</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Keywords</td>
<td>Keyword applicable to the document</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
<tr>
<td>Creator</td>
<td>The name of who created the document</td>
<td>String</td>
<td>No</td>
<td>&quot;&quot;</td>
</tr>
</tbody>
</table>

Exceptions:

none

Example:

```
<Summary>
  <Title>Table Drawing Tool Standard - Requirements Document</Title>
  <Author>Mihai Bilauca</Author>
  <Subject>Tabular Expressions</Subject>
  <Keywords>Expressions, SQRL, quality, software</Keywords>
  <Creator>Mihai Bilauca</Creator>
</Summary>
```

10.2 Initial Settings

The initial settings contain information about the page that will be displayed when the user opens the document.

Element name: <InitialSettings>
Parent element: <DeviceDescriptor>

Parameters:
<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>InitialPageIndex</td>
<td>The index of the page that will be displayed to the user when the document is opened</td>
<td>Integer</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td>InitialZoom</td>
<td>The zoom of the first page that is opened</td>
<td>Float</td>
<td>No</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_INVALID_INTEGER** : when the value provided for the `<InitialPageIndex>` element cannot be evaluated to an integer;
- **EX_INVALID_FLOAT**; when the value provided for the `<InitialZoom>` element cannot be evaluated to a float;

Example:

```xml
<InitialSettings>
  <InitialPageIndex>3</InitialPageIndex>
  <InitialZoom>0.5</InitialZoom>
</InitialSettings>
```

### 10.3 Page

The `page` element groups all the layout control parameters for an individual page.

**Element name:** `<Page>`  
**Parent element:** `<Pages>`

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The index of the page.</td>
<td>Integer</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td>PageSize</td>
<td>A string constant representing a standard dimension page. If the <code>PageSize</code> parameter is provided the dimensions set in the Box element are replaced with the dimensions given by the <code>PageSize</code> parameter. See section 10.4 for details.</td>
<td><code>PageSize</code></td>
<td>No</td>
<td>“A4”</td>
</tr>
<tr>
<td>PageOrientation</td>
<td>An integer value representing whether the longer side of the page is on the horizontal direction (landscape) or on</td>
<td><code>PageOrientation</code></td>
<td>No</td>
<td>PORTRAIT</td>
</tr>
</tbody>
</table>
the vertical direction (portrait). See section 10.5 for details.

<table>
<thead>
<tr>
<th>Box</th>
<th>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details. If the Page Size parameter is specified it overwrites the dimension specified in the Box element</th>
<th>Element</th>
<th>No</th>
<th>N.A.</th>
</tr>
</thead>
</table>

Exceptions:

- **EX_INVALID_INTEGER**: when the value provided for the `<Index>` element cannot be evaluated to an integer;
- **EX_INVALID_VALUE**: if the value provided for the `<FontStyle>` or `<PageOrientation>` elements cannot be evaluated to any valid values or if the value for the `<Index>` element is outside the required range;

Example:

```xml
<Pages>
  <Page>
    <Index>2</Index>
    <PageSize>A3</PageSize>
    <PageOrientation>LANDSCAPE</PageOrientation>
    <Box></Box>
  </Page>
</Pages>
```

10.4 Page Size

The `PageSize` type is an enumerated type with string values representing standard page sizes.

Name: `PageSize`

Type: `String`

The `PageSize` type consists of the following string values:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>A0</td>
<td>“A0”</td>
<td>Metric ISO A0 format</td>
</tr>
<tr>
<td>A1</td>
<td>“A1”</td>
<td>Metric ISO A1 format</td>
</tr>
<tr>
<td>A2</td>
<td>“A2”</td>
<td>Metric ISO A2 format</td>
</tr>
<tr>
<td>A3</td>
<td>“A3”</td>
<td>Metric ISO A3 format</td>
</tr>
<tr>
<td>A4</td>
<td>“A4”</td>
<td>Metric ISO A4 format</td>
</tr>
<tr>
<td>A5</td>
<td>“A5”</td>
<td>Metric ISO A5 format</td>
</tr>
</tbody>
</table>
10.5 Page Orientation

The `PageOrientation` type is an enumerated type with integer values representing whether the longer side of a page is on the horizontal direction or on the vertical direction.

Name: `PageOrientation`  
Type: `Byte`

The `PageOrientation` type consists of the following integer values:

<table>
<thead>
<tr>
<th>Identifier</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PORTRAIT</td>
<td>0</td>
<td>The longer side of the page is on the vertical direction</td>
</tr>
<tr>
<td>LANDSCAPE</td>
<td>1</td>
<td>The longer side of the page is on the horizontal direction</td>
</tr>
</tbody>
</table>
11 Rectangular Layout

A rectangular layout is a class that encodes a layout with a rectangular shape. This class implements all the functions specified in Section 7 in order to qualify for a *layout* class.

A rectangular layout consists of maximum 5 grids and maximum of 4 corners. The corners are just placeholders for additional information. Figure 9 shows the rectangular layout structure.

<table>
<thead>
<tr>
<th>Grid index</th>
<th>Description</th>
<th>Position</th>
<th>Rows</th>
<th>Columns</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Main grid</td>
<td>Center</td>
<td>Limited by the available memory</td>
<td>Limited by the available memory</td>
<td>Yes</td>
</tr>
<tr>
<td>1</td>
<td>Left header</td>
<td>Left</td>
<td>Same as G0</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>2</td>
<td>Top header</td>
<td>Top</td>
<td>1</td>
<td>Same as G0</td>
<td>No</td>
</tr>
</tbody>
</table>

In Figure 9 the grids are noted with G and the corners with C. The grids have indexes 0 to 4. The grid index 0 is always the main grid that is placed at the center of the layout. The grids with indexes 1 to 4 are the headers of the main grid.

The grids must respect the following requirements:
- At least the main grid (Index=0) and two headers must be specified;
- Grids 1 and 3 must have the same number of rows as the G0 and at least one column. Grids 2 and 4 must have the same number of columns as G0 and at least one row. Corners are not required but could be used to display related information;
- No two grids can have the same index;

The grids that can be specified in a rectangular layout are displayed in Table 7 and the list of corners in Table 8.

![Figure 9 Rectangular layout structure](image-url)
<table>
<thead>
<tr>
<th>Grid index</th>
<th>Description</th>
<th>Position</th>
<th>Rows</th>
<th>Columns</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Right header</td>
<td>Right</td>
<td>Same as G0</td>
<td>1</td>
<td>No</td>
</tr>
<tr>
<td>4</td>
<td>Bottom header</td>
<td>Bottom</td>
<td>1</td>
<td>Same as G0</td>
<td>No</td>
</tr>
</tbody>
</table>

Table 8 The list of corners and their position in the rectangular layout

<table>
<thead>
<tr>
<th>Corner index</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Upper left</td>
</tr>
<tr>
<td>1</td>
<td>Upper right</td>
</tr>
<tr>
<td>2</td>
<td>Bottom right</td>
</tr>
<tr>
<td>3</td>
<td>Bottom left</td>
</tr>
</tbody>
</table>

**Element name:** «LayoutDescriptor»  
**Parent element:** «Layout»

**Parameters:**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>SplitCells</td>
<td>Whether the cells should be split when the layout of a tabular expression extends horizontally on more than one page. This option has been introduced for compatibility purposes only. In this version TDTS does not split cells when adding pages on the horizontal direction.</td>
<td>Boolean</td>
<td>No</td>
<td>FALSE</td>
</tr>
<tr>
<td>MergeCells</td>
<td>Whether adjacent cells that have the same content should be merged or not. Merging cells improves the clarity of the expression.</td>
<td>Boolean</td>
<td>No</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**Grids** Groups all the grids in the rectangular layout.  
**Corners** Groups all the corners that can be specified in a rectangular layout.

**Exceptions:**
- **EX_REQUIRED_ELEMENT** if the «Grids» elements is missing or no grids have been specified;
- **EX_INVALID_BOOLEAN**; when the value provided for the «SplitCells» or «MergeCells» elements cannot be evaluated to a boolean;
Example:

```
<Layout>
  <LayoutDescriptor>
    <SplitCells>FALSE</SplitCells>
    <MergeCells>TRUE</MergeCells>
    <Grids></Grids>
    <Corners></Corners>
  </LayoutDescriptor>
</Layout>
```

11.1 Corner

A corner is in fact a box structure that can be placed in one of the four corners of a rectangular layout. See section 4.3.8 for information on the box structure.

Element name: `<Corner>`  
Parent element: `<Corners>`

Attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Index</td>
<td>The corner’s index. In the list of corners no two corners can have the same index. Table 8 lists the position of the corners depending on their index</td>
<td>Integer range 0 to 3</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:
• **EX_REQUIRED_ELEMENT** if the `<Index>` element is missing;
• **EX_INVALID_VALUE**; if the value provided for the `<Index>` element is outside the required range;
• **EX_DUPLICATE_INDEX** - if a corner with the same index value has already been defined;

**Example:**

```xml
<Corners>
  <Corner Index="1">
    <Box>
      <Width>500</Width>
      <Height>200</Height>
    </Box>
  </Corner>
</Corners>
```

11.2 Grid

A grid is a structure that contains a set of cells arranged on rows and columns. Each cell is identified by two indexes: `RowIndex` and `ColumnIndex`. The layout of the grid is controlled by the parameters encoded in its `box` element.

**Element name:**   <Grid>
**Parent element:** <Grids>

**Attributes**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Index</strong></td>
<td>The grid’s index. In the list of grids of a rectangular layout no two grids can have the same index.</td>
<td>Integer</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>Rows</strong></td>
<td>The number of rows in the grid</td>
<td>Integer &gt;0</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>Columns</strong></td>
<td>The number of columns in the grid</td>
<td>Integer &gt;0</td>
<td>Yes</td>
<td>N.A.</td>
</tr>
<tr>
<td><strong>CellPadding</strong></td>
<td>The default padding for all cells</td>
<td>Float &gt;=0</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>CellMargins</strong></td>
<td>The default margins for all cells</td>
<td>Float &gt;=0</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>CellBorderWidth</strong></td>
<td>The default border width for all cells</td>
<td>Float &gt;=0</td>
<td>No</td>
<td>0</td>
</tr>
<tr>
<td><strong>CellBorderColour</strong></td>
<td>The default border width for all cells</td>
<td>Colour</td>
<td>No</td>
<td>Black</td>
</tr>
<tr>
<td><strong>CellBorderStyle</strong></td>
<td>The default border width for all cells</td>
<td>LineStyle</td>
<td>No</td>
<td>Solid</td>
</tr>
</tbody>
</table>
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cells</td>
<td>Contains the layout parameters for the set of cells that belong to the grid. See section for details on the cell element.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:
- **EX_REQUIRED_ELEMENT** if the `<Index>` element is missing;
- **EX_DUPLICATE_INDEX** - if a grid with the same index value has already been defined;
- **EX_INVALID_VALUE**; if the value provided for the `<Index>`, `<Rows>` or `<Columns>` elements is outside the required range;

Example:
```
<Grid Index="0" Rows="2" Columns="25">
  <Cells></Cells>
  <Box></Box>
</Grid>
```

11.3 Cell

A cell is the smallest unit of a grid and is the direct placeholder for the expressions of a tabular expression. Each cell is identified within the grid by its indexes: the row index and the column index.

**Element name:** `<Cell>`

**Parent element:** `<Grid>`

Attributes:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RowIndex</td>
<td>The index of the row to which this cell belongs to.</td>
<td>Integer</td>
<td>Yes</td>
<td>0</td>
</tr>
<tr>
<td>ColumnIndex</td>
<td>The index of the column to which this cell belongs to.</td>
<td>Integer</td>
<td>Yes</td>
<td>0</td>
</tr>
</tbody>
</table>
Parameters:

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Type</th>
<th>Required</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Box</td>
<td>Groups all the visual appearance parameters (dimensions, colours, text size, etc). See section 4.3.8 for details.</td>
<td>Element</td>
<td>No</td>
<td>N.A.</td>
</tr>
</tbody>
</table>

Exceptions:

- **EX_REQUIRED_ELEMENT** if the `<RowIndex>` or `<ColumnIndex>` elements are missing;
- **EX_DUPLICATE_INDEX** - if a cell with the same row index and column index values has already been defined;
- **EX_INVALID_VALUE**: if the value provided for the `<RowIndex>,<ColumnIndex>` elements is outside the required range;

Example:

```xml
<Cell RowIndex="0" ColumnIndex="1">
  <Box></Box>
</Cell>
```
12 Future work

The following could be implemented in future releases:

1. *CharacterSpace, WordSpace, LineSpace* properties are ignored. This is because iText, the library used to generate PDF content, miscalculates the width of a text area when these properties are set to values different than 0. In fact, the additional width induced by these properties is not taken into account when fitting text in a given area. Therefore the text is misplaced;

2. Caption management. Implement a system that counts all the expressions and their inner expressions. Each caption has an associated number and a numbering style. The number of expressions should be displayed in the caption using a numbering style.

3. Custom numbering styles.

4. Implement rounded boxes. In this version a *box* is a rectangle with straight corners.

5. Implement more line styles;
13 Bibliography


Appendix C

TDT Sample Input File
Listing C.1: Sample Input File

```xml
<?xml version="1.0" encoding="utf−8" ?>
<TDTS>
  <LayoutControl>
    <Expressions>
      <Expression ID="1">  
        <Caption>
          <Position>RIGHT</Position>
          <Text>Condition Table 4.3.1−c:AS Reference Point in Designated..</Text>
        </Caption>
        <Layout>
          <LayoutClass>RectangularLayout</LayoutClass>
          <LayoutDescriptor>
            <Grids>
              <Grid Index="0" Rows="6" Columns="6">
                <Box><Width>400</Width></Box>
                <Cells>
                  <Cell RowIndex="0" ColumnIndex="0">
                    <Text>*Nattack*
                    *Snattack*
                    *HUDdownl*
                    *SHUDdownl*</Text>
                  </Cell>
                  <Cell RowIndex="0" ColumnIndex="1">
                    <Text>NOT ! Desig!</Text>
                  </Cell>
                  <Cell RowIndex="0" ColumnIndex="2">
                    <Text>X</Text>
                  </Cell>
                  <Cell RowIndex="0" ColumnIndex="3">
                    <Text>X</Text>
                  </Cell>
                  <Cell RowIndex="0" ColumnIndex="4">
                    <Text>! Desig!</Text>
                  </Cell>
                  <Cell RowIndex="0" ColumnIndex="5">
                    <Text>X</Text>
                  </Cell>
                </Cells>
                <Cells>
                  <Cell RowIndex="1" ColumnIndex="0">
                    <Text>*Noffset*
                    *Snoffset*
                  </Cell>
                </Cells>
              </Grids>
            </LayoutDescriptor>
          </Layout>
        </Layout>
      </Expression>
    </Expressions>
  </LayoutControl>
</TDTS>
```
<table>
<thead>
<tr>
<th></th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOT ! Desig ! AND ! Before slewing!</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOT ! Desig ! AND ! After slewing!</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>! Desig !</td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>BOC</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SBOC</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Column 0</th>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>BOCoffset</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><em>SBOCoffset</em></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

263
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>X</td>
<td>X</td>
<td>! Desig!</td>
<td>X</td>
</tr>
<tr>
<td></td>
<td>X</td>
<td>NOT ! Desig!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>∗BOCFlytoO∗</td>
<td></td>
<td>NOT ! Desig! AND ! Before slewing!</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SBOCFlytoo</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>∗Walleve∗</td>
<td></td>
<td>Alwavs</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix D

Tables Generated with TDT

In this chapter we show a number of tables that have been generated with TDT. The minimum table heights have been computed using IBM CPLEX solver [34], OPL [83] and the TSC model for tables with spanning cells presented in Section 4.1. We show how TDT renders the layout of table presented in Figure 1.5 while taking into account various formatting parameters such as different font size, padding, border widths, etc.

We show the table from Figure 1.4 rendered using TDT to highlight how TDT handles fonts, background colours, spanning cells and border settings.

We also show how the minimum height can be computed for very large tables with 60 rows and 20 or 28 columns. The tables are generated for A0 pages and scaled down to A4 in order to be included in this thesis.
Condition Table 3.4-b
Transitions Between Alignment, Navigation, and Test Modes
While the Aircraft Is Not Airborne (/ACARIRB/=\$No$)

<table>
<thead>
<tr>
<th>Current Mode</th>
<th>New Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Lautocal</em></td>
<td><em>Landain</em></td>
</tr>
<tr>
<td></td>
<td><em>I</em></td>
</tr>
<tr>
<td></td>
<td><em>OLB</em></td>
</tr>
<tr>
<td><em>Sautocal</em></td>
<td><em>SINSain</em></td>
</tr>
<tr>
<td></td>
<td><em>I</em></td>
</tr>
<tr>
<td></td>
<td><em>OLB</em></td>
</tr>
<tr>
<td><em>Landain</em></td>
<td><em>Lautocal</em></td>
</tr>
<tr>
<td></td>
<td><em>Landain</em></td>
</tr>
<tr>
<td></td>
<td><em>HUDain</em></td>
</tr>
<tr>
<td></td>
<td><em>I</em></td>
</tr>
<tr>
<td></td>
<td><em>OLB</em></td>
</tr>
<tr>
<td></td>
<td><em>Mag sl</em></td>
</tr>
<tr>
<td></td>
<td><em>Grid</em></td>
</tr>
<tr>
<td></td>
<td><em>IMS fall</em></td>
</tr>
<tr>
<td></td>
<td><em>PolarI</em></td>
</tr>
<tr>
<td></td>
<td><em>Grtest</em></td>
</tr>
<tr>
<td></td>
<td><em>I</em></td>
</tr>
<tr>
<td></td>
<td><em>OLB</em></td>
</tr>
<tr>
<td></td>
<td><em>Mag sl</em></td>
</tr>
<tr>
<td></td>
<td><em>Grid</em></td>
</tr>
<tr>
<td></td>
<td><em>IMS fall</em></td>
</tr>
<tr>
<td></td>
<td><em>PolarI</em></td>
</tr>
<tr>
<td></td>
<td><em>Grtest</em></td>
</tr>
<tr>
<td></td>
<td><em>I</em></td>
</tr>
<tr>
<td></td>
<td><em>OLB</em></td>
</tr>
<tr>
<td></td>
<td><em>Mag sl</em></td>
</tr>
<tr>
<td></td>
<td><em>Grid</em></td>
</tr>
<tr>
<td></td>
<td><em>IMS fall</em></td>
</tr>
<tr>
<td></td>
<td><em>PolarI</em></td>
</tr>
<tr>
<td></td>
<td><em>Grtest</em></td>
</tr>
</tbody>
</table>

Legend:
- TT = Transition between modes
- f = Failure
- t = Temporary
- @T = Test
- *Grtest* = Ground test

Legend for Conditions:
- ACARIRB/=$Yes$
- IMSMODE/=$Gndal$
- /IMSMODE/=$Mag si$
- !SINS up!
- !SINS velocity
- P5
- *PolarI*
<table>
<thead>
<tr>
<th>MODES</th>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nattack*</td>
<td>NOT !Desig!</td>
</tr>
<tr>
<td>Snattack*</td>
<td>X</td>
</tr>
<tr>
<td>HUDdown1*</td>
<td>X</td>
</tr>
<tr>
<td>SHUDdown1*</td>
<td>!Desig! X</td>
</tr>
<tr>
<td>Notoffset*</td>
<td>NOT !Desig! AND !Before slewing!</td>
</tr>
<tr>
<td>Snoffset*</td>
<td>X</td>
</tr>
<tr>
<td>HUDdown2*</td>
<td>!Desig! AND !After slewing!</td>
</tr>
<tr>
<td>SHUDdown2*</td>
<td>!Desig! X</td>
</tr>
<tr>
<td>BOC*</td>
<td>X !Before slewing!</td>
</tr>
<tr>
<td>SBOC*</td>
<td>X !After slewing!</td>
</tr>
<tr>
<td>BOCflyto0*</td>
<td>!Desig! NOT !Desig!</td>
</tr>
<tr>
<td>SBOCFlyto0*</td>
<td>!Desig! NOT !Desig!</td>
</tr>
<tr>
<td>BOCoffset*</td>
<td>NOT !Desig! AND !Before slewing!</td>
</tr>
<tr>
<td>SBOCoffset*</td>
<td>NOT !Desig! AND !After slewing!</td>
</tr>
<tr>
<td>Wallieve*</td>
<td>Always X</td>
</tr>
<tr>
<td>WALLIEVE*</td>
<td>X</td>
</tr>
</tbody>
</table>

**REFERENCE POINT**

<table>
<thead>
<tr>
<th>HUD Aimsight (see below)</th>
<th>!Fly-to point! (ground stabilized)</th>
<th>IOAP! (ground stabilized)</th>
<th>!Target! (ground stabilized)</th>
<th>point on ground track! 8 nmi in front of A/C</th>
<th>X</th>
</tr>
</thead>
<tbody>
<tr>
<td>MODES</td>
<td>CONDITIONS</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-------</td>
<td>------------</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Nattack*&quot; <em>Snattack</em> &quot;HUDdown1* &quot;SHUDdown1*</td>
<td>NOT !Desig! X X !Desig! X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Noffset*&quot; &quot;Snoffset* &quot;HUDdown2* &quot;SHUDdown2*</td>
<td>NOT !Desig! AND !Before slewing! X NOT !Desig! AND !After slewing! !Desig! X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;BOC*&quot; &quot;SBOC*&quot;</td>
<td>X !Before slewing! X !After slewing! X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;BOCFlyto0*&quot; &quot;SBOCFlyto0*&quot;</td>
<td>X X X !Desig! NOT !Desig!</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;BOCoffset*&quot; &quot;SBOCoffset*&quot;</td>
<td>X NOT !Desig! AND !Before slewing! NOT !Desig! AND !After slewing! !Desig! X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>&quot;Walleve*&quot;</td>
<td>Always X X X X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

REFERENCE POINT
HUD Aimsight (see below) !Fly-to point! (ground stabilized) !OAP! (ground stabilized) !target! (ground stabilized) point on !ground track! 8 nmi in front of A/C
## Condition Table 4.3.1-c

**AS Reference Point in Designated Weapon Delivery Modes**

<table>
<thead>
<tr>
<th>MODES</th>
<th>CONDITIONS</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>Nattack</em></td>
<td>NOT !Desig!</td>
</tr>
<tr>
<td><em>Snattack</em></td>
<td>NOT !Desig!</td>
</tr>
<tr>
<td><em>HUDdown1</em></td>
<td>NOT !Desig! AND !Before slewing!</td>
</tr>
<tr>
<td><em>SHUDdown1</em></td>
<td>NOT !Desig! AND !After slewing!</td>
</tr>
<tr>
<td><em>Noffset</em></td>
<td>!Desig!</td>
</tr>
<tr>
<td><em>Snoffset</em></td>
<td>!Desig!</td>
</tr>
<tr>
<td><em>HUDdown2</em></td>
<td>!Desig! AND !Before slewing!</td>
</tr>
<tr>
<td><em>SHUDdown2</em></td>
<td>!Desig! AND !After slewing!</td>
</tr>
<tr>
<td><em>BOC</em></td>
<td>X</td>
</tr>
<tr>
<td><em>SBOC</em></td>
<td>X</td>
</tr>
<tr>
<td><em>BOCflyto0</em></td>
<td>NOT !Desig! AND !Before slewing!</td>
</tr>
<tr>
<td><em>SBOCflyto0</em></td>
<td>NOT !Desig! AND !After slewing!</td>
</tr>
<tr>
<td><em>BOCoffset</em></td>
<td>!Desig!</td>
</tr>
<tr>
<td><em>SBOCoffset</em></td>
<td>!Desig!</td>
</tr>
<tr>
<td><em>Walleve</em></td>
<td>Always</td>
</tr>
</tbody>
</table>

### REFERENCE POINT
- HUD Aimsight (see below)
- !Fly-to point! (ground stabilized)
- !OAP! (ground stabilized)
- !Target! (ground stabilized)
- point on ground track! 8 nmi in front of A/C
Table 60x20 with up to 6 cell words per cell. Page size is set to A0, Portrait

Top, left and right margins are set to 400 points. Bottom margin is set to 100 points

<table>
<thead>
<tr>
<th>Column 1</th>
<th>Column 2</th>
<th>Column 3</th>
<th>Column 4</th>
<th>Column 5</th>
<th>Column 6</th>
<th>Column 7</th>
<th>Column 8</th>
<th>Column 9</th>
<th>Column 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
<tr>
<td>Content 1</td>
<td>Content 2</td>
<td>Content 3</td>
<td>Content 4</td>
<td>Content 5</td>
<td>Content 6</td>
<td>Content 7</td>
<td>Content 8</td>
<td>Content 9</td>
<td>Content 10</td>
</tr>
</tbody>
</table>
This document is the second published release of the... system evolves into subsequent versions. The requirements document should be easy to change. (3)

Serve as [101x1142]the years. This requirements for either requires specification I SPECIFICATION increasingly first required specification change it as better specification refers, should indicate unresolved requirements implementations of the system evolves for hardware, meet... hardware, meet the system. All implementations... expected to build an... expected to build an... December 1978 to... obvious... acceptable yield the best description of the primary... acceptable yield the best description of the primary... laboratory. A... laboratory. A... specification... methodology, the first... methodology, the first... fundamental... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first. For instance, the... methodology, the first... methodology, the first... methodology, the first... methodology, the first. For instance, the... methodology, the first... methodology, the first. For instance, the... methodology, the first... methodology, the first. For instance, the... methodology, the first. For instance, the... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first. For instance, the... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... methodology, the first... method...
Appendix E

Sample Text Source

Listing E.1: Sample from the source text for the automatically generated test tables

1 This document is the second published release of the Software Requirements of the A−7E Aircraft. The first release, published in November 1978, introduced a new approach to specifying requirements for real−time embedded systems in the form of an engineering model. That document has been perhaps the most successful of the publications of NRL’s Software Cost Reduction project in terms of the interest generated and the number of copies I requested since its introduction.

2 In spite of its success (in a sense, because of it) the specification has changed in many details over the years. This is not the result of flaws in its design, but the fulfillment of its creators’ vision I that the requirements should be a "living document;" i.e., that it would serve as the primary reference document for system designers, as well as the authoritative "test to" document for program validation, and be useful throughout the system development process. Because the document has served these purposes as well, it has changed over the years as requirements became better understood. I Further, since the document is intended to serve as a model document, we have felt free to change it as better specifications techniques have been developed. This release represents the accumulation of those changes from the original publication in November 1978 to the end of the SCR project in December 1988.

3 In spite of many changes in its particulars, the reader will find the document remarkably unchanged in its overall structure and approach. One of the principles guiding the original design was I that because requirements change, the requirements specification should be easy to change. As a result, incremental changes and improvements have been easy to accommodate over the years without disrupting the essential document structure. [Chmu82]
Index

A-7E, 14 63
abstract object, 145
Adobe FrameMaker, 13 20
aesthetical arrangement, 140
aircrew scheduling, 46
alignment, 146
AMPL, 49 53
authoring, 25 29
axes, 143

Bézier curves, 146
background
  colour, 133 137
  pattern, 137
benchmark, 7 14 64 166
best practice, 115
bin packing, 8 119
boolean algebra, 46
border space, 139
borders, 133 139
Bouygues, 11 89
box, 139 147 150
Branch-And-Bound, 46
Branch-And-Cound, 46
broadside tables, 116 131

C, 46 156
C++, 46 48 156
Calc, 116
canvas, 144
caption, 146 150
  label, 147 149
  number, 147
text, 147
CBC, 46
cell, 153
Chicago Manual of Style, 115
CHIP, 46
CHOCO, 44
classification, 28 29
clique problem, 33
co-ordinate system, 143
collapsing margins, 143
colour, 155
column groups, 8 126 127
column headers, 115
column ordering, 120 125 126 130 131
column positioning, 8 118 121 122 125
column reordering, 117
combinatorial optimization, 7, 31, 39, 52, 131, 156
combinatorial problems, 41
COMET, 49
computeLayout, 138
confidence intervals, 102, 104
configurations, 52, 64, 71, 111, 113, 157
consistency techniques, 46
constraints
  aesthetic, 6
  designer, 32, 34, 62, 72
  global, 43, 47
  higher order, 53, 61
  linear, 31
  logical, 53
  optimization, 6, 9, 34
  propagation, 47
  reasoning, 59
  store, 41
  systems, 6, 32, 41, 46
container, 137
  child, 137
  floating, 140
  parent, 137
  root, 137
  siblings, 137
continuation header, 115, 116
convex hull, 33
coordinate space, 146
coordinate system, 139, 145
Corel Draw, 13
corners, 151
CP Optimizer, 63, 89
CPLEX, 63, 70, 86, 89, 95, 112, 118, 156, 157, 161
criteria, 131
CSS, 13, 19, 138, 139, 143, 161
custom data types, 154
custom style, 149
cutting-plane, 46
data items, 133
data meaning, 8, 115, 131
data mining, 39
decision variables, 54
design guidelines, 115
Design Science, 145
designer, 115
Desktop Publishing, 13, 140
device, 136, 138, 145, 148, 153
dimensions, 133
document object, 28, 137, 140
drawing model, 161
drawing units, 143, 145, 146
dummy cells, 81
dynamic rendering, 6

ebook, 8

equations, 20, 133, 145, 153

exceptions, 155

expression counter, 150

expression number, 150

facing pages, 116

flow control, 122

font, 133

  family, 145, 146

  size, 137, 145

  style, 145, 155

Fortran, 156

GAMS, 44, 49, 53

geost, 43

GIF, 145

glue, 78, 79, 84

graph theory, 46

grid, 29, 31, 34, 133, 151, 152

grid flow, 143

grid layout, 140, 143

guidelines, 115

hardware design, 39

heuristics, 6, 59, 101

  maximal regret, 61

hierarchical model, 29, 138

hierarchical propagation, 137

hierarchical structure, 133, 137

hierarchy, 137

HTML, 8, 35

IBM ILOG Concert Technology, 156, 157, 161

illustrations, 137

ILOG, 46, 48, 61, 63, 71

indexation, 30

inheritance, 140

inherited parameter, 137

initial page, 154

initial zoom factor, 154

inner expressions, 150

inner tables, 4, 14, 20, 36, 111, 133

interface, 153

iText, 146

iWorks Pages, 4, 13, 16, 20, 21

Java, 46, 155, 156, 161

JPEG, 145

Juno, 31

LaTeX, 13, 18, 20, 22, 24

layout, 28, 146, 150, 151

  circular, 146, 161

  custom, 161

  descriptor, 152

278
page count, 8 [125 128 131]
opimum, 119
page space, 120
pagination, 8, 9
paging, 146
paragraphing, 64, 72
partial sums, 117
PDF, 8, 64, 141, 146, 161
PDF device, 136, 146, 153
pixels, 145
placement, 39
PNG, 145
position, 155
programming
conic, 35
constraint, 10, 36, 39, 51, 59, 70, 157
constraint logic, 46
convex, 35
dynamic, 35
integer, 41, 44
linear, 33, 41, 44
local search, 39, 48, 89, 104, 157
logic, 46
logical, 41
mathematical, 41, 46, 53
mixed integer, 10, 36, 39, 44, 51, 119
multi-objective, 122
non-linear, 41, 44, 46
PROLOG, 46
SICStus, 44
QuarkXPress, 13
rectangle packing, 42, 44
rectangular layout, 136, 138
regret, 61
render, 28, 138
rendering, 145
repeated headers, 116
resource file, 155
resource management, 155
scheduling, 42, 45, 53
SCIP, 46
Sketchpad, 62
search, 53, 59, 61, 70
set, 30
simplex, 46
Sketchpad, 47
spanning cells, 133
splitting tables, 118, 122, 125, 131, 146
style manual, 115
sweep-point algorithm, 43
table
appearance, 30
borders, 18, 142
caption, 21, 115
caption style, 137
footers, 21
footnotes, 115
groups, 21
header, 117
headers, 21 116
identifier, 115
layout, 136
margins, 17 143
resize, 16
semantics, 30
summary, 22
templates, 21
Table Drawing Tool, 133 136 147 156 161
Table Input Tool, 136 148
Table Kernel Module, 136 148 150
table layout
  problem, 5 33
Table Tool System, 135 136 147
TABPRINT, 139
tabular expressions, 30 135 146 148 153 161
tabular formatting, 135 199
tabular layout, 114
Tbl, 140
TDT, 134 147 199
TeX, 13
text, 137
text colour, 133
text direction, 144
text rotation, 144
text spacing, 146
Thinglab, 47
Tioga, 28
topology, 26 29 30 33
totals, 117
transposition, 27 29
travelling salesman, 46
TTS, 135 147
typographical parameters, 137
typographical problems, 134
typography, 26 30 133
VDP, 6
Visual Basic, 156
WebEQ, 145
weighted-sum-optimization, 33
wide columns, 117
wide tables, 8 10 21 32 33 114 119 120 131
WYSIWYG, 4 8 16 28 133
XML, 7 136 148 153
XSLT, 7
Bibliography


285


287


[58] A. Makhorin. GLPK (GNU linear programming kit), 2006. 46


290


291


