FITTING A TABLE TO A PAGE USING NON-LINEAR OPTIMISATION

Nicholas Beaumont

Working Paper 46/03
July 2003

ISSN 1327–5216

Abstract

It is sometimes difficult to fit a large table comprising several rows and columns onto a page. The usual tactic is to manually adjust column widths, abbreviate some text and/or change some cells' font sizes until the table fits onto a page. We show that it is possible to express the problem of adjusting column widths so as to minimize the height of a table as an optimization problem with non-linear constraints. Five test problems were routinely solved using a free software package. We stress that the solutions are approximate because the model imperfectly simulates how many lines of a cell of a table will be required to contain a segment of text, but they appear to provide good approximations in difficult cases. The scant literature is summarized; the formulation and solution techniques outlined; examples are described; and differences between theoretical and actual answers explained. It would be possible to incorporate the calculations in word processing and typesetting packages such as Word and TeX.

FITTING A TABLE TO A PAGE USING NON-LINEAR OPTIMISATION

INTRODUCTION

Research papers often contain tables summarising or classifying data and/or results. When a large table contains several columns or rows with very different amounts of text in cells, it can be difficult to adjust manually the width of columns so that the table fits onto one page or column.

The usual approach to squeezing a table onto a page is trial and error; we adjust the column widths, reduce the font size and/or abbreviate text in large cells until the tables fits. Unless the publication is of very high quality, this approach is probably more appropriate than the mathematical overkill espoused here. However, the problem of adjusting column widths (subject to page width) to minimise the height of a table is expressible as a non-linear optimization problem that can be routinely solved.

The literature was unsuccessfully searched for signs that this problem had been studied. A keyword search of the Operations Research literature was fruitless. In conversation, a publisher stated that this problem had not been addressed. Computer packages such as Microsoft Word, Wordperfect and Netscape Composer allows the user to set the width of the columns of a table but do not offer any kind of optimization of column widths. Similarly, typesetting and desktop publishing systems such as TeX (Knuth, 1986) Adobe Pagemaker (Connally, 2002) and QuarkXpress (Bain, 2002) do not have facilities for optimizing column widths. The web was unsuccessfully searched for discussion of the column optimization problem.

The problem has the air of a packing or cutting stock problem. The most closely related problem is the two-dimensional cutting stock problem in which rectangular shapes are cut from a continuous, fixed-width sheet (of metal, paper or cloth) so as to minimise the length used. The cuts are guillotine and orthogonal (either parallel or perpendicular to the sides of the sheet or roll). The required shapes are fixed, a large number of each shape is usually required, and the objective is to arrange the shapes so as to minimise some measure of wastage.

In the table-fitting problem, the “shapes” have fixed spatial relationships, are rectangular and have (nearly) fixed areas. The cuts are guillotine and orthogonal. The only degree of freedom is setting the column widths (which implicitly determine the row heights).

The traditional cutting stock problem is essentially geometric combinatorial and NP complete (Dyckhoff, 1990, p 157). The table-fitting problem is relatively easy: it is not combinatorial but does include quadratic constraints. Two surveys of the cutting stock (Cheng, Feiring and Cheng, 1994; Dyckhoff, 1990) and pallet loading literature (Ram, 1992) were fruitlessly searched for traces of or analogues to the table-fitting prolem.

FORMULATION

The objective is to minimize the height of a rectangular table by varying the width of its columns. Optimization is constrained by the maximum page or column width and each cell having to be large enough to contain its contents (usually text).

A table is a grid of rows and columns, a cell is defined by the intersection of a row and column. We assume that rows and columns are of constant height and width respectively throughout the table but the formulation given can be trivially modified if a cell spans more than one row and/or column.

The column width is conveniently measured in terms of the mean width of a character; the height is conveniently measured in terms of the distance between the bases of two adjacent lines. Word processing packages allow the user to define cell margins: white space between the text in a cell and the horizontal or vertical edges of a cell. The parameter *col_tol* (defined below) would normally be twice the vertical margin but can be set higher to allow for loss of space caused for example by text not filling out lines completely. This point is discussed further below (see SIMULATING WORD 2000’S BEHAVIOUR).

Notation

M the set indexing rows

N the set indexing columns.

m, n the number of rows and columns respectively.

w the page or column width.

row_tol a row tolerance defining the whitespace appearing adjacent to horozontal grid lines.

col_tol a column tolerance defining the whitespace appearing adjacent to vertical grid lines Low

$lr(i), ur(i)$ $i \in M$ lower and upper bounds on row heights.

$lc(j), uc(j)$ $j \in N$ lower and upper bounds on column widths.

$\mathrm{A}(i, j)$ $i \in M, j \in N$ The amount of text in the cell indexed by i, j of the table.

er and upper bounds on the height and width of rows and columns may be dictated by aesthetics or the need to fit an icon or image into a cell. $A(i, j)$ is the minimal area required by a cell.

$x(i)$ the height of row $i \in M$.

$y(j)$ the width of column $j \in N$.

$z(i, j)$ the useable area of each cell $i \in M, j \in N$.

The useable area of a cell is the area not taken up by white space defined by *row_tol* and *col_tol* at the cell borders.

Problem Defnition

Bounds

$lr(i) \leq x(i) \leq ur(i) \quad i \in M.$

$lc(j) \leq y(j) \leq uc(j) \quad j \in N.$

$z(i, j) \geq A(i, j) \quad i \in M, j \in N.$

Objective Function and Constraints

$$\text{Min} \sum_{i \in M} x(i).$$

$$\sum_{j \in N} y(j) \leq w.$$

$(x(i) - col\_tol) * (y(j) - row\_tol) = z(i, j) \quad i \in M, j \in N.$ defines the useable cell area.

All variables are non-negative.

The model is small; there are ostensibly $mn + m + n$ variables, $m * n$ non-linear constraints and one linear constraint. However, AMPL (Fourer, GayandKernighan, 1998, pp 121-2, 247-248) will, in most circumstances, eliminate variables defined by equalities. In this problem the variables $z(i, j)$ $i \in M, j \in N$ (retained for narrative clarity) will be eliminated leaving $m + n$ variables.

EXAMPLES

Example 1

Example 1(Table 1) contains text formatted in 12-point courier font each character of which has width 0.1 inch. Because courier fonts have constant character width, a 12-point courier font and imperial measurements were used for clarity of exposition in all examples (the ligature *fi* takes less space than *mm* in most fonts). This table has three columns of equal width (2 in) and "height" 30 lines. This table is almost certainly not optimized: there is spare space in each of column 1's cells. The method espoused was used to minimise the height of this table. The data file (Figure 1) describes the table; matrix A stores the number of characters of text (including spaces) in each cell of the table.

The student edition of the AMPL package (Fourer, Gay *et al.*, 1998) (downloadable free from http://www.ampl.com/cm/cs/what/ampl/index.html) was used to solve this problem. AMPL is a matrix generation package that incorporates several alternative optimisers; the MINOS optimiser was chosen because it can handle non-linear constraints. The student edition can solve problems with up to 300 constraints and 300 non-integer variables. The model input to AMPL is given in Figure 2

MINOS failed to solve the problem when a default set of initial values (all variables at their lower bounds) was used. When initial values:

$$y(j) = w/n \quad j \in N,$$

$$x(i) = \max_{j \in N} \left\{ A(i,j) / \left[y(j) - row\_tol \right] \right\} + col\_tol \quad i \in M.$$

were prescribed, a solution was found without difficulty. Mindful of the possibility of having found a local but not global minimum, we experimented with a few other starting values but always obtained the same optimum.

The parameter *col_tol* that defines the total amount of horizontal white space in a cell strongly affects the "look" and height of the table. Microsoft's Word package does not exploit all the space in a cell; inter-word spaces at the end of a line are not suppressed and hyphenation is evidently suppressed within tables.

Some of the output from AMPL is given in Table 3 and Table 4. The optimal column widths were 13.1, 25.7 and 21.2 characters. These values were rounded to integers; the student edition of AMPL does not give integer solutions. The optimised table with "height" 26, is shown as Table 4. Perhaps the dual variables' values are the most interesting aspect. The *page_width* constraint has a dual variable value of -0.35, widening the table by 3 characters width would ostensibly reduce the height of the table by one line. Reducing the content of the row 3 column 1 or row 2 column 3 cells would slightly reduce the table height.

Although the table's appearance is improved, the reduction in height (from 30 to 26 lines) is not especially impressive for this small example especially as the adjustment could have been done by hand. Manual adjustment of large tables is more difficult. The method was applied to four larger tables whose details are given in the next section.

Examples 2 Through 5

The tables used in these examples are not reproduced. In all examples, a 12pt Courier New font with ten characters per inch was used and *row_tol* was set to zero.

Example 2 has 3 rows, 4 columns (is 3 by 4), *col_tol* = 0 and $w = 0$; the number of characters in each cell is given in Table 5;Table 6 summarises and compares the original and optimally formatted tables. The third row of Table 6 gives the optimal continuous solution. The fourth row was obtained by (a) rounding the row widths from the continuous solution to the nearest integer and (if necessary) permuting these answers so that they sum to the page width and (b) Reformatting the table with these row widths in Word 2000 and noting the "height" or total number of rows Word 2000 used to print the table. Exact agreement cannot be expected because Word wastes some space at the ends of rows.

Example 3 is 4 by 3, has *col_tol* = 0 and a page width of 60 characters; its cell contents are given in Table 7 and Table 8, like Table 6, summarises and compares the original and optimally formatted tables. The optimal column widths are not very different from the original and the saving in vertical space is not very great.

Examples 4 and 5 are large 3 by 4 tables with *col_tol* = 0 and $w = 60, 80$ respectively; the number of characters in each cell is given in Table 9. Table 10 and Table 11, similar to Table 6, summarise and compare the original and optimally formatted tables.

SIMULATING WORD 2000'S BEHAVIOUR

The optimisation problem assumes that all space within a cell is used. This is not so: Especially as Word 2000 does not always hyphenate words, space is wasted at the end of lines (If a column is narrower than a word's length, the word will be split between lines without a hyphen being used).

Word 2000 contains a parameter *Hyphenation Zone* that specifies the amount of space that may be left at the right-hand end of a line. If, after normal formatting, more space than *Hyphenation Zone* would be left, Word exploits it by hyphenating the first word of the next line. High values of *Hyphenation Zone* reduce the amount of hyphenation and waste space. Low values exploit space by using frequent hyphenation. Unfortunately and illogically, Word 2000 turns off hyphenation in tables.

The parameter *col_tol* that gives the mean amount of space wasted per line was varied: the optimal solutions for problems 2 and 5 for different values of *col_tol* are summarised in Table 12. In problem 2, the optimal column widths are insensitive to changes in *col_tol* probably because no column is especially narrow. Contrastingly, in problem 3; the width of the narrowest column is increased by one unit with each unit increase in *col_tol*. This small sample suggests that, to ape Word 2000's behaviour, *col_tol* should be set to 2.

CONCLUSION

We note that, by changing the objective function to $\text{Min} \sum_{i \in M} x(i) + \sum_{j \in N} y(j)$ we solve the problem of minimising the table's perimeter or internal and external "fencing" but no application of this variant comes to mind.

It is unlikely that, unless an individual author was preparing an especially high quality publication, he or she would use the analysis described. The method might be useful to a professional typesetter if it could be routinised. It might be economic to include the method in automated form in a widely used package such as Microsoft Word or TeX. The ability to automatically compute the length of a text string and the area required to print any block of text in a given column width would be advantageous.

This is a simple, perhaps pedagogically useful, application of optimisation with non-linear constraints. Perhaps the most striking finding was that software that can generate the matrix for and routinely solve this problem is readily available.

A large number of solvers for linear and non-linear problems are available on the web (http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html and http://www-unix.mcs.anl.gov/otc/Guide/faq/nonlinear-programming-faq.html and, remarkably, a service offering to solve a variety of optimisation problems is available at the Optimization Technology Center at http://www.ece.nwu.edu/OTC/.

In contrast to the variety of solvers available, there is a paucity of matrix generators. AMPL seems to be well designed: as well as generating the matrix, the user can include preliminary calculations and reasonableness checks; there are many useful functions; many useful ways of defining and using sets and very flexible output options.

REFERENCES

Bain, S.(2002), QuarkXpress 5: the complete reference, Osborne/McGraw-Hill, Berkeley, Calif.

Cheng, C.H., B.R. Feiring and T.C.E. Cheng(1994), The cutting stock problem--a survey, International Journal of Production Economics 36, 291-305.

Connally, C.M.(2002), PageMaker 7: the complete reference, Osborne/McGraw-Hill, Berkeley, Calif. ; London.

Dyckhoff, H.(1990), A typology of cutting and packing problems, European Journal of Operational Research 44, 145-159.

Fourer, R., D.M. Gay and B.W. Kernighan(1998), AMPL: A Modelling Language for Mathematical Programming, Boyd & Fraser, Massachusetts.

Knuth, D.E.(1986), The TeXbook, Addison-Wesley, Reading, Mass.

Ram, B.(1992), The pallet loading problem: A survey, International Journal of Production Economics 28, 217-225.

FIGURES

Figure 1: AMPL model file

```
param NR integer;  # number of rows
param NC integer;  # number of columns
param col_tol;  # the amount of blank space at the top
                   # and bottom of a cell
param col_tol;  # the amount of blank space at the left
                                        # and right of a cell
set rows := 1 .. NR;
set cols := 1 .. NC;
param page_width >0;
param A{rows, cols} >= 0;      # the required area of each
                                                      # cell of the table
param lr{rows} >=0; ;          # lower bound on row height
param ur{i in rows} >= lr[i];  # upper bound on row height
param lc{cols} >=0; ;          # lower bound on column width
param uc{i in cols} >= lc[i];  # upper bound on column
                                                      #width
#intial values for x and y
param init_x {i in rows}
:= max {j in cols} A[i,j]/(page_width/card{cols}- col_tol)+ col_tol;
param init_y {j in cols} := page_width/(card{cols} - col_tol);  # the initial values of x
var x{i in rows} integer >=lr[i], <= ur[i], := init_x[i];  # the height of each row
var y{j in cols} integer >=lc[j], <= uc[j], := init_y[j];  # the width of each column
var z{i in rows, j in cols} >= A[i,j];
# the area of each cell
minimize total_height:  sum{i in rows} x[i];
subject to page_w: sum {j in cols} y[j] <=  page_width;    # total width less than page width
subject to cell_area {i in rows, j in cols}:
# define available area of the cell
          (x[i]- col_tol)* (y[j]- col_tol) = z[i,j];
# end of table design model.
```

Figure 2: Data for the page problem

```
# data for the page width model
# Nick Beaumont 21 jan 02
param NR := 3;
param NC := 3;
param page_width := 60;
param col_tol := 0.0;
param col_tol := 6.0;
param:   lr   ur :=
   1     0    80
   2     0    80
   3     0    80 ;
param:   lc   uc :=
   1     0    60
   2     0    60
   3     0    60 ;
param A:
          1     2    3:=
   1      92   266   47
   2      70    45  157
   3      35    21   58 ;
```

TABLES

Example 1

Table 1: An example of a table to be optimised

Competitive pricing to insure that the company is placed at the leading edge of the market.	Increased focus on meeting KPI/SLA to ensure that the service offering meets the ongoing needs of the business. Flexibility of supply systems to capture not only financial data, but service activity levels and their ability to meet the KPIs. how far do they go?	On time as per the agreement and accurate data.
Cost per unit and fee structure. Purchase price. Whole of life cost.	Delivery according to key process indicators.	Systems and reporting mechanisms that support all the components of the service/product offering allowing the supply to manage its business more efficiently.
How the cost compares with the market (benchmarking).	Margin of error rate.	The feasibility of the population is well below sea level.

Table 2: Table 1 optimised

Competitive pricing to insure that the company is placed at the leading edge of the market.	Increased focus on meeting KPI/SLA to ensure that the service offering meets the ongoing needs of the business. Flexibility of supply systems to capture not only financial data, but service activity levels and their ability to meet the KPIs. How far do they go?	On time as per the agreement and accurate data.
Cost per unit and fee structure. Purchase price. Whole of life cost.	Delivery according to key process indicators.	Systems and reporting mechanisms that support all the components of the service/product offering allowing the supply to manage its business more efficiently.
How the cost compares with the market (benchmarking).	Margin of error rate.	The feasibility of the population is well below sea level.

Table 3: Optimal row and column dimensions

Row or column	Row height	Column width
1	13.5	13.1
2	10.4	25.7
3	4.9	21.2
Totals	28.8	60.0

Table 4 Example 1: Values of cell size, required cell size and reduced costs

Row	Column	Z (calculated cell size)	A (Required cell size)	Reduced costs
1	1	96.5	92	-
1	2	266.0	266	0.05
1	3	204.4	47	-
2	1	74.1	70	-
2	2	204.4	45	-
2	3	157.0	157	0.07
3	1	35.0	35	0.14
3	2	96.5	21	-
3	3	74.1	58	-

Example 2

The table has 3 rows, 4 columns and page width of 60.

Table 5 Example 2: Number of characters (including spaces) in each cell

	Column 1	Column 2	Column 3	Column 4
Row 1	531	265	265	131
Row 2	399	131	527	399
Row 3	265	399	0	531

Table 6 Example 2: Details of initial and optimal solutions

	Column widths	Total width	Row heights	Total number of rows
Original	15, 15, 15, 15	60	34, 53, 49	136
Optimal solution	18.07, 10.26, 18.01, 13.66	60	29.38, 29.27, 38.88	97.52
Optimal solution rounded and applied in Word 2000.	18, 10, 18, 14	60	27, 33, 40	100

Example 3

The table has 4 rows and 3 columns.

Table 7 Example 3: Number of characters (including spaces) in each cell

	Column 1	Column 2	Column 3
Row 1	677	548	218
Row 2	300	99	249
Row 3	399	300	656
Row 4	656	1082	300

Table 8 Example 3: Details of initial and optimal solutions

	Column widths	Total width	Row heights	Total number of rows
Original	20, 20, 20	60	41, 17, 39, 62	159
Optimal solution	20.77, 21.99, 17.24	60	32.59, 14.44, 38.05, 49.21	134.29
Optimal solution rounded and applied in Word 2000.	21, 22, 17	60	34, 16, 40, 52	142

Example 4

The table has 3 rows and 4 columns and width 60.

Table 9 Examples 4 & 5: Number of characters (including spaces) in each cell

	Column 1	Column 2	Column 3	Column 4
Row 1	1547	712	361	1026
Row 2	159	1662	337	671
Row 3	1622	637	364	1882

Table 10 Example 4: Details of initial and optimal solutions

	Column widths	Total width	Row heights	Total number of rows
Original	15, 15, 15, 15	60	125, 189, 209	426
Optimal solution	16.80, 18.58, 5.45, 19.17	60	104.54, 100.24, 109.61	314.40
Optimal solution rounded and applied in Word 2000.	17, 18, 6, 19	60	108, 114, 151,	373

Example 5 Large

The table has 3 rows and 4 columns and width 80.

Table 11 Example 5: Details of initial and optimal solutions

	Column widths	Total width	Row heights	Total number of rows
Original	20, 20, 20, 20	80	140, 53, 117	310
Optimal solution	22.77,25.51,5.31,26.41	80	67.95. 65.16, 71.25	204.36
Optimal solution rounded and applied in Word 2000.	23, 25,6,26	80	79, 85, 82	246

Table 12: The effect of the *col_tol* parameter on table size.

Test problem	*col_tol*	Optimal number of rows	Number of rows when optimal column widths (rounded) are used in word
Example 2 (4 by 3 $w = 60$)	0	134	142
	1	141	142
	2	149	142
Example 5 (3 by 4 $w = 80$)	0	204	246
	1	215	246
	2	227	236
	3	240	232