

Tabular formatting problems[‡]

Xinxin Wang* and Derick Wood[†]

Technical Report HKUST-CS96-28

June 1996

*NorTel

P.O. Box 3511, Station C
Ottawa, Ontario K1Y 4H7
Canada

[†]Department of Computer Science
Hong Kong University of Science & Technology
Clear Water Bay, Kowloon
Hong Kong

Abstract

Tabular formatting determines the physical dimensions of tables according to size constraints. Many factors contribute to the complexity of the formatting process so we analyze the computational complexity of tabular formatting with respect to different restrictions. We also present an algorithm for tabular formatting that we have implemented in a prototype system. It supports automatic line breaking and size constraints expressed as linear equalities or inequalities. This algorithm determines in polynomial time the physical dimensions for many tables although it takes exponential-time in the worst case. Indeed, we have shown elsewhere that the formatting problem it solves is NP-complete.

[‡]This research was supported by grants from the Natural Sciences and Engineering Research Council of Canada, from the Information Technology Research Centre of Ontario, and from the Research Grants Committee of Hong Kong.



Tabular formatting problems¹

Xinxin Wang²

Derick Wood³

Abstract

Tabular formatting determines the physical dimensions of tables according to size constraints. Many factors contribute to the complexity of the formatting process so we analyze the computational complexity of tabular formatting with respect to different restrictions. We also present an algorithm for tabular formatting that we have implemented in a prototype system. It supports automatic line breaking and size constraints expressed as linear equalities or inequalities. This algorithm determines in polynomial time the physical dimensions for many tables although it takes exponential-time in the worst case. Indeed, we have shown elsewhere that the formatting problem it solves is NP-complete.

1 Introduction

When we design a table, we first decide on the logical structure by taking into account the readers' requirements and convenience. Then, we specify the topology to arrange the items in two dimensions and select a presentational style so that the logical structure of the table is clearly seen. Given the logical structure of a table, a topological specification, and a style specification, we can generate a concrete table in two phases. First, in the *arrangement phase*, we generate a grid structure and a set of size constraints for the columns and rows in the grid structure. Then, in the *formatting phase*, we determine the physical dimensions of the columns and rows according to the size constraints.

Since a table contains different kinds of items, including text, graphics, images, mathematical equations, and even subtables, tabular formatting is inherently more complex than the formatting of text. There are three main factors that contribute to the complexity of tabular formatting:

1. The method of handling the line breaking of text within a tabular cell.

Current tabular composition systems adopt two approaches to handle line breaking: fixed and automatic. *Fixed line breaking*, adopted by many systems, requires users to indicate the line breaks in the table items. *Automatic line breaking*, adopted by TAFEL MUSIK [SKS94], requires the system to determine the line-break points based on the current dimensions of the columns.

2. The kinds of size constraints for columns and rows.

¹This work was supported under grants from the Natural Sciences and Engineering Research Council of Canada and from the Information Technology Research Centre of Ontario.

²NorTel, P.O. Box 3511, Station C, Ottawa, Ontario K1Y 4H7, Canada. E-mail: xinxin@nortel.ca.

³Department of Computer Science, Hong Kong University of Science & Technology, Clear Water Bay, Kowloon, Hong Kong. E-mail: dwood@cs.ust.hk.

Most tabular composition systems can specify only simple size constraints for tables, for example, the constraints that restrict row heights, column widths, and table width and height. Beach's system [Bea85] allows users to specify size constraints expressed as linear equalities or inequalities for rows and columns. For example, the sum of the widths of the first and second columns should equal the width of the third column. Since size constraints with nonlinear expressions are not often used and require a time-consuming constraint solver, no current system handles such kinds of size constraints.

3. The objective function that evaluates the quality of a tabular layout.

Most systems do not offer any ability to help users to format tables with optimal constraints, namely *objective functions*. For example, the diameter of a table should be minimum. Beach's system [Bea85] offers only minimal diameter. TAFEL MUSIK [SKS94] offers three different objective functions: minimal diameter, minimal area, and minimal white space.

As far as we know, Beach is the only person who has discussed the computational complexity of tabular formatting although Vanoirbeek [Van92] has investigated tabular composition within the framework of the Grif system. In his PhD thesis [Bea85], Beach identified a tabular formatting problem, RANDOM PACK that arranges a set of unordered table entries into minimum area, and proved that RANDOM PACK is NP-complete. Because of the random positioning of the table entries, RANDOM PACK does not produce pleasing and readable tables that clearly convey their logical structure. Beach also identified another problem, GRID PACK that formats a set of table entries assigned to lie between particular row and column grid coordinates within the table, and proved that GRID PACK is polynomial-time solvable. GRID PACK, however, assumes that the width and the height of the table entries are fixed; thus only fixed line breaks are allowed. Although Beach also allowed size constraints expressed as linear equalities or inequalities in his tabular model, he did not include size constraints in RANDOM PACK and GRID PACK. The designers of TAFEL MUSIK [SKS94] have designed an exponential-time algorithm for tabular formatting that provides automatic line breaking, allows size constraints expressed as linear equalities and inequalities, and considers objective functions.

XTABLE [Wan96, WW96] is an interactive tabular composition system that runs in a UNIX and X Windows environment. It is not only a tool for the design of high-quality presentations of tables, but also it is a tool for the exploration of tabular data from different viewpoints. XTABLE abstracts a table's multidimensional logical structure in a similar, yet different way, to the abstraction suggested by Vanoirbeek [Van92]. Abstract tables are mapped to different presentations according to user-defined topological and style specifications. The formatting process of XTABLE supports both fixed and automatic line-breaking methods and automatically determines the physical dimensions of a final layout according to user-defined size constraints expressed as linear equalities or inequalities.

In this paper, we analyze the computational complexity of tabular formatting with respect to different restrictions. Tabular formatting can be polynomial-time solvable or NP-complete, depending on the functionality it supports. We also present an algorithm, which is used by XTABLE, for an NP-complete formatting problem that supports automatic line-breaking and size constraints expressed as linear inequalities. This algorithm determines the physical dimensions for many tables in polynomial time.

Table 1: The complexity of tabular formatting.

Line breaks	Size constraints	Objective functions			
		None	Diameter	Area	White space
Fixed	None	P^1	P^3	P^3	P^3
	Linear equality or inequality	P^3	P^1	P^3	P^3
	Nonlinear expression	?	?	?	?
Automatic	None	P^3	?	?	?
	Linear equality or inequality	NPC^2	NPC^3	NPC^3	NPC^3
	Nonlinear expression	?	?	?	?

¹ Proved by Richard Beach [Bea85].² See Theorem 5.1 [Wan96].³ See the discussion in Chapter 7 of Wang’s thesis [Wan96].

2 Complexity analysis

Based on previous research and our work, we summarize the complexity of tabular formatting for different combinations of the restrictions in Table 1, where P denotes polynomial-time solvable and NPC denotes NP-complete.

Beach proved two of the complexity results [Bea85]. Based on his results, we obtain the complexity results for the remaining polynomial-time solvable problems. We [Wan96] have established NP-completeness when the formatting problem includes automatic line breaking and size constraints, but disregards objective functions. We use the abbreviation TFALN for this problem. Based on the complexity result for TFALN, we can also obtain NP-completeness results for the three problems that also include an objective function. We have not classified the complexities of the problems that include an objective function but do not handle any size constraints. The complexity results for all problems that handle size constraints with nonlinear expressions are also unknown.

Automatic line breaking is important and useful for tabular formatting. It is also important to allow users to control the selection of the dimensions of columns and rows for a table. In this paper, we focus on TFALN, the tabular formatting problem that supports only automatic line breaking and size constraints. We disregard objective functions to simplify tabular formatting for two reasons. First, a layout that is optimal with respect to an objective function does not always provide the most appropriate layout. An optimal solution may make one column too narrow and another too wide, or generate a table with an unacceptable aspect ratio. Second, users tend to care more about the sizes of tabular components, such as whether a table can be placed inside a region of a given width and height, whether the relative sizes of the components in a table are appropriate, and whether the relative sizes of

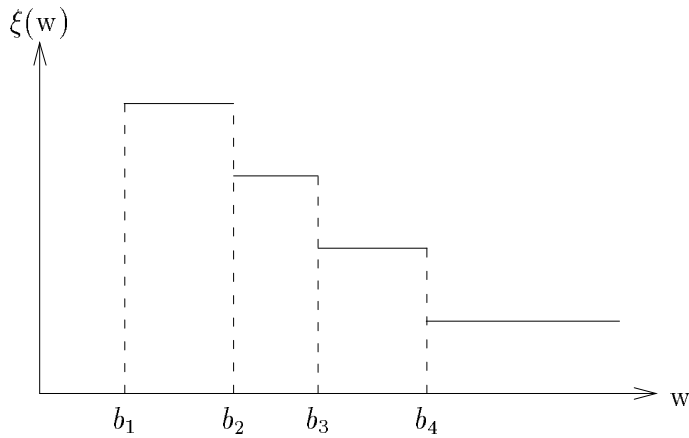


Figure 1: The characteristics of a size function.

a table and its surrounding objects are appropriate. Such requirements are specified by size constraints, rather than by objective functions.

3 Definition of TFALN

Before we formally define TFALN, we first define a *grid structure* that we use to model the topological arrangement of table items in two dimensions.

We inherited this concept from Beach’s system [Bea85] but make some changes to it. A grid structure consists of two components: a grid and a set of non-overlapping items that are placed on the grid.

An $m \times n$ *grid* is a planar integer lattice with m rows and n columns. The intersection of a row and a column is called a *cell* and the cell that is the intersection of the i th row and the j th column is identified by (i, j) . A *block* is a rectangular region that completely surrounds a set of cells, and it is identified by (t, l, b, r) , where (t, l) is its upper left cell and (b, r) is its lower right cell.

An *item* is an object that is placed in a block of a grid. The content of an item can be a string, a number, a textual object, a fixed-sized picture or image, or a table. (In the current formulation we do not allow uniform magnification of images.) The *size function* of an item is a decreasing step function that describes the line-breaking characteristics of the item for a particular output device. It takes a width as its argument and returns the height of the item when the item is typeset within the given width. We can assume that both the width and the height are integers. The characteristics of a size function for an item are shown in Fig. 1. We use a *step* to denote the range of widths in $[b_k, b_{k+1})$, where b_k and b_{k+1} are two adjacent break points or b_k is the maximal break point and b_{k+1} is $+\infty$. The lower bound of a step is called a *step head* and the upper bound of a step, which is $b_{k+1} - 1$ if the step is $[b_k, b_{k+1})$ or $+\infty$ if the step is $[b_k, +\infty)$, is called a *step tail*. A size function returns the same height for all the widths in a step. In Fig. 1, the size function consists of four steps $[b_1, b_2)$, $[b_2, b_3)$, $[b_3, b_4)$, and $[b_4, +\infty)$.

We can specify an item by a six-element tuple (t, l, b, r, ξ, ψ) , where (t, l, b, r) is the block in which the item is placed in the grid, ξ is its size function, and ψ is the set of step heads for ξ . If s is a step, we use $s.head$ to denote its head and $s.tail$ to denote its tail. If ψ is a set of step heads for a size function, we use $\psi[min]$ to denote the minimal step head and $\psi[max]$ to denote the maximal step head.

Table 2: The tournament schedule.

Activity	Final Entry Date	Starting Date, Location, Times
Men's & Women's squash	Monday, Jan. 23,	Prelim. Sat. Jan. 28, Finals Sun. Jan. 29, 11:00am-6:00pm, Court 1068-1073, PAC
Singles Tennis	1:00pm, PAC 2039	Prelim. Sun. Feb. 5, 10:00am-6:00pm, Finals Sun. Feb. 12, 10:00am-6:00pm, Waterloo Tennis Club
Mixed Volleyball	Friday, Mar. 3,	Prelim. Wed. Mar. 8, 8:00pm-11:30pm, Finals Mon. Mar. 13, 8:00pm-11:30pm, Main Gym, PAC
Men's & Co-Rec Broomball	1:00pm, PAC 2039	Prelim. Fri. Mar. 17, 12:00pm-5:00pm, Finals Sat. Mar. 18, 3:00pm-1:00am, Columbia Icefield

Now we can formally define TFALN as follows:

INSTANCE: An $m \times n$ grid, r non-overlapping items: $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$ ($1 \leq k \leq r$), and s size constraints: e_1, e_2, \dots, e_s .

QUESTION: Are there $n + m$ integers w_1, w_2, \dots, w_n and h_1, h_2, \dots, h_m such that

1. $W = w_1, w_2, \dots, w_n$ satisfy all width constraints among e_1, e_2, \dots, e_s ;
2. $H = h_1, h_2, \dots, h_m$ satisfy all height constraints among e_1, e_2, \dots, e_s ;
3. $\forall o_k (1 \leq k \leq r), \sum_{p=l_k}^{r_k} w_p \geq \psi_k[\min]$ and $\xi_k(\sum_{p=l_k}^{r_k} w_p) \leq \sum_{q=t_k}^{b_k} h_q$.

The $w_j (1 \leq j \leq n)$ are the column widths and the $h_i (1 \leq i \leq m)$ are the row heights of the grid. The first two conditions ensure that $w_j (1 \leq j \leq n)$ and $h_i (1 \leq i \leq m)$ satisfy all the size constraints. The third condition ensures that the width of the block for each item is at least the minimal width of the item and the height of the block should be sufficient to hold the item when it is typeset within the width of the block. If $w_j (1 \leq j \leq n)$ and $h_i (1 \leq i \leq m)$ satisfy all three conditions for an instance, we say that the instance has *solution* (W, H) . If they satisfy only the third condition for an instance, we say the instance has *layout* (W, H) . Suppose we have an instance that consists of a 5×3 grid and the 13 items shown in Table 2. The size constraints for this instance are:

$$\begin{aligned} 290pt &\leq w_1 + w_2 + w_3 \leq 380pt \\ h_1 + h_2 + h_3 + h_4 + h_5 &\leq 350pt \\ w_3 &\geq 120pt. \end{aligned}$$

One of the solutions for this instance is:

$$\begin{aligned} w_1 &= 65pt, \quad w_2 = 68pt, \quad w_3 = 230pt, \\ h_1 &= 31pt, \quad h_2 = 47pt, \quad h_3 = 45pt, \quad h_4 = 47pt, \quad h_5 = 45pt, \end{aligned}$$

which are the physical dimensions of Table 2.

4 A formatting algorithm

An NP-complete problem does not have polynomial-time algorithms unless $P = NP$, which is considered unlikely. With this assumption, we can provide only an exponential-time algorithm for TFALN that solves every instance. We first describe an exponential-time algorithm and then we describe a polynomial-time greedy algorithm that partially solves TFALN for many common instances. Finally, we combine these two algorithms to obtain an algorithm that is guaranteed to solve TFALN completely and correctly and takes only polynomial time for many instances.

4.1 An exponential-time algorithm

The simplest way to solve TFALN is to check all the possible combinations of row heights and column widths. The first combination that satisfies all three conditions of TFALN is selected as a solution. We can improve this method by solving the size constraints to obtain row heights for given column widths. Once the column widths are fixed, the heights and widths of the items are also fixed; thus, we can use Beach's approach to find the row heights in polynomial time. We can further improve the algorithm by taking advantage of the characteristics of size functions. Since the height of an item will be the same when it is typeset within the widths of a step, we need to test only one of the widths in a step. For each combination of steps, we can find the column widths and row heights by solving inequalities. Suppose that item o_j has K_j steps; then, the number of checked combinations can be reduced to $N = \prod_{j=1}^r K_j$. N still increases at an exponential rate when most of the items have more than one step. In many tables, however, most of the items contain only one step. N is not too large for these cases. Based on this approach, we have designed an exponential-time algorithm that completely solves TFALN [Wan96, WW96].

Based on a given step combination $C = \{s_1, s_2, \dots, s_r\}$ of all the items, where s_k is a step of item o_k , we attempt to find column widths $w_j (1 \leq j \leq n)$ such that:

1. $w_j (1 \leq j \leq n)$ satisfies all the width constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $s_k.head \leq \sum_{p=l_k}^{r_k} w_p \leq s_k.tail$.

Similarly, we attempt to find row heights $h_i (1 \leq i \leq m)$ such that:

1. $h_i (1 \leq i \leq n)$ satisfies all the height constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $\xi_k(s_k.head) \leq \sum_{q=t_k}^{b_k} h_q$.

We find the solutions by solving a set of linear equalities and inequalities. There is an algorithm for this problem based on the simplex method [Dan63] that runs in $O(t^3)$ time, where t is the number of equalities and inequalities. Moreover, the algorithm guarantees that the sum of the values of the variables in the equalities and inequalities is minimum. Therefore, we can find column widths and row heights in $O((r + s)^3)$ time, where r is the number of items and s is the number of size constraints. The total running time is then

$$O\left(\left(\prod_{j=1}^r K_j\right) \times (r + s)^3\right),$$

where K_j is the number of steps for item o_j .

We need to introduce some notation before we present the greedy algorithm for TFALN. Suppose C is a step combination for an instance of TFALN. We use $WIE(C)$ to denote the

set of equalities and inequalities generated when we find the column widths for C and we use $\text{HIE}(C)$ to denote the set of equalities and inequalities generated when we find the row heights for C . If $w_j(1 \leq j \leq n)$ satisfy only the inequalities for item sizes (Condition 2) in $\text{WIE}(C)$, then $w_j(1 \leq j \leq n)$ is called a *layout* of $\text{WIE}(C)$ and if $h_i(1 \leq i \leq m)$ satisfy only the inequalities for item sizes (Condition 2) in $\text{HIE}(C)$, then $h_i(1 \leq i \leq m)$ is called a *layout* of $\text{HIE}(C)$.

4.2 A polynomial-time greedy algorithm

The first algorithm takes exponential time, in most cases, to find a solution for TFALN. Most tables, however, usually have few size constraints. For many such cases, we are able to find a solution in polynomial time by taking advantage of the monotonicity property of size functions. Given an instance I of TFALN, the monotonicity property of size functions enables us to generate a list $L_I = C_1, C_2, \dots, C_z$ of step combinations, where $C_u = \{s_1^u, s_2^u, \dots, s_r^u\}(1 \leq u \leq z)$, that satisfies the following properties:

Property 1 For the first step combination C_1 , $\text{WIE}(C_1)$ must have at least one solution.

Property 2 For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, s_k^{u+1} is either the same as s_k^u or the successor of s_k^u ; thus, $\xi_k(s_k^{u+1}.head) \leq \xi_k(s_k^u.head)$.

Property 3 There is at least one item such that its step in C_{u+1} is larger than its step in C_u .

Property 4 In the last step combination C_z , for each k , $1 \leq k \leq r$, s_k^z is the largest step of item o_k .

Property 5 For each step combination $C_u(1 \leq u \leq z)$, there is a layout $w_j^u(1 \leq j \leq n)$ for $\text{WIE}(C_u)$ and a layout $h_i^u(1 \leq i \leq m)$ for $\text{HIE}(C_u)$.

Given an instance I of TFALN, we try to generate a list L_I of step combinations that satisfies Properties 1–5. While we are checking the step combinations in L_I , we have three possible results: yes, no, and uncertain. Based on this approach, we obtain a polynomial-time algorithm that partially solves TFALN. In this algorithm, we generate the first step combination C_1 that satisfies Property 1 and a layout (W^1, H^1) , where $W^1 = w_1^1, \dots, w_n^1$ and $H^1 = h_1^1, \dots, h_m^1$, in which all items are typeset within the corresponding steps in C_1 . To obtain the first step combination, we attempt to find the column widths $w_j^1(1 \leq j \leq n)$ such that

1. $w_j^1(1 \leq j \leq n)$ satisfy the width constraints.
2. For each item $o_k = (t_k, l_k, b_k, r_k, \xi_k, \psi_k)$, $\sum_{p=l_k}^{r_k} w_p^1 \geq \psi_k[\min]$.

Given a step combination C_u and its layout (W^u, H^u) , where $W^u = w_1^u, \dots, w_n^u$ and $H^u = h_1^u, \dots, h_m^u$, The polynomial-time algorithm finds a new step combination C_{u+1} , generates a new layout (W^{u+1}, H^{u+1}) , where $W^{u+1} = w_1^{u+1}, \dots, w_n^{u+1}$ and $H^{u+1} = h_1^{u+1}, \dots, h_m^{u+1}$, in which all items are typeset within the corresponding steps in C_{u+1} , and ensures that L_I satisfies Properties 2–5. To reduce the number of uncertain responses, we try to find a step combination that can generate a solution or lead us to a solution rapidly by selecting as few items as possible whose steps we increase, to avoid reaching the largest steps of the items as long as possible. Based on these ideas, we use the following heuristics to obtain C_{u+1} :

1. For each column $1 \leq k \leq n$, we increase its width w_k^u to a new width w_k^* such that w_k^* is the minimal width to cause at least one item to fall into the next step. Based on $w_1^u, \dots, w_{k-1}^u, w_k^*, w_{k+1}^u, \dots, w_n^u$, we generate a new step combination C'_k and a layout (W'_k, H'_k) , where $W'_k = w'_{k1}, \dots, w'_{kn}$ and $H'_k = h'_{k1}, \dots, h'_{km}$. The n step combinations C'_1, C'_2, \dots, C'_n are possible candidates for C_{u+1} .
2. During Step 1, if we find that all items have reached their largest steps, we return **End**.
3. If there is a C'_k such that both $\text{WIE}(C'_k)$ and $\text{HIE}(C'_k)$ have solutions, then C_{u+1} is chosen as this C'_k and (W^{u+1}, H^{u+1}) as (W'_k, H'_k) .
4. If we do not find a C_{u+1} in Step 3, we let C_{u+1} be a C'_k such that $\sum_{j=1}^n w'_{kj} + \sum_{i=1}^m h'_{ki}$ is a minimum. In this case, (W^{u+1}, H^{u+1}) is chosen as (W'_k, H'_k) .

Step 1 guarantees that each C'_k satisfies Properties 2 and 3. It also guarantees that each C'_k satisfies Property 5 because $w_1^u, \dots, w_{k-1}^u, w_k^*, w_{k+1}^u, \dots, w_n^u$ must be a layout for $\text{WIE}(C'_k)$ and $h_i^u (1 \leq i \leq m)$ must be a layout for $\text{HIE}(C'_k)$. Step 2 ensures that L_I satisfies Property 4. Steps 3 and 4 increase the likelihood that we find a solution. Step 4 is based on the observation that we usually specify size constraints for table width and height. If we make the table width and height as small as possible, we are more likely to find a solution in the succeeding search.

The running time for finding the first step combination is $O((r+s)^3)$ and the running time for finding the next step combination is $O(n(n+m+(r+s)^3))$. The number of the step combinations in the list is at most $\sum_{j=1}^r K_j$. Therefore, the total running time for the greedy algorithm is

$$O\left(\sum_{j=1}^r K_j n(n+m+(r+s)^3)\right).$$

The running time increases at a polynomial rate as n , m , r , and s increase.

4.3 An efficient algorithm

By combining the two algorithms, we obtain a more efficient algorithm that can completely and correctly solve TFALN. For each instance of TFALN, we first use the greedy algorithm to check a list of step combinations C_1, C_2, \dots, C_z that satisfy Properties 1–5. If it does not find a solution for the instance, then we use the first algorithm. We have established elsewhere the correctness of this combined algorithm [Wan96]. Although it is still an exponential-time algorithm in the worst case, it is more efficient than the first algorithm for many instances.

We can divide the instances of TFALN into two groups, G_e and G_p . G_e includes the instances for which greedy algorithm returns *Uncertain* and G_p includes the instances for which the first algorithm returns either *Yes* or *No*. Thus, the new algorithm takes polynomial time to solve the instances in G_p and takes exponential time to solve the instances in G_e . Given a rectangular region, text is usually typeset to fill a region that is as wide as possible. If the region is not wide enough, then the text is broken into lines to vertically fill the region. Thus, we usually specify only width constraints to control the layout of a table. In these cases, $\text{HIE}(C_1)$ must have solutions and we can decide whether there are solutions to the instances using the greedy algorithm. The height constraints may be necessary when a table is too tall to fit into a region and it is possible to shorten it by widening the table. Therefore, we believe that G_p contains many more common instances than G_e . For languages in which people are used to reading text from top to bottom (such as Chinese and Japanese), a similar observation holds when we interchange the roles of widths and heights in the algorithm.

5 Conclusions

We have proved that the tabular formatting problem is NP-complete with respect to two useful features: automatic line breaking and size constraints expressed as linear inequalities. This complexity result guided us in the design of a new formatting algorithm that allows automatic line-breaking and size constraints since they are important features that can help users to deal with table sizes and shapes. We can extend the combined algorithm to generate locally optimal solutions for an objective function among a set of layouts. In polynomial time, we can check all the step combinations and select an optimal solution from all the layouts we found, rather than terminating when we have found a layout that satisfies the size constraints. Whether we can design an algorithm to solve the tabular formatting problem when we include objective functions is a challenging issue that we leave as an interesting, future investigation.

XTABLE [Wan96] adopts the main ideas of the combined algorithm to determine the physical dimensions of a table. Although the combined algorithm supports any size constraints expressed as linear equalities or inequalities, we restrict the size constraints in XTABLE to simplify the user interface and decrease the execution time of the tabular formatting algorithm. XTABLE allows only two kinds of linear inequalities for the size constraints: $l \leq \sum_{j=p}^q w_j \leq u$ and $l \leq \sum_{i=p}^q h_i \leq u$. We believe that these two kinds of size constraints are sufficient to specify most size requirements for tables.

Since the allowable size constraints in XTABLE are simpler, we are able to reduce the running time of the algorithm by making two changes. First, we do not use the simplex method to solve the linear equalities and inequalities. We use a more efficient inequality solver. Second, we use a branch-and-bound strategy to generate only those step combinations that guarantee to give a layout for a table. Any step combination which will not give a layout is not considered. For example, suppose two items o_1 and o_2 are placed in the same column and o_1 has a step [20, 30) and o_2 has a step [50, 60); then there is no layout for a step combination that contains these two steps because they do not overlap. By omitting such step combinations, the number of step combinations that are checked is greatly reduced.

References

- [Bea85] R. J. Beach. *Setting Tables and Illustrations with Style*. PhD thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, May 1985. Also issued as Technical Report CSL-85-3, Xerox Palo Alto Research Center, Palo Alto, CA.
- [Dan63] G. Dantzig. *Linear Programming and Extensions*. Princeton University Press, 1963.
- [SKS94] K. Shin, K. Kobayashi, and A. Suzuki. TAFEL MUSIK, formatting algorithm of tables. In *Principles of Document Processing 94*, pages 1–25, Lufthansa Training Center, Seeheim, May 1994.
- [Van92] C. Vanoirbeek. Formatting structured tables. In C. Vanoirbeek and G. Coray, editors, *EP92 (Proceedings of Electronic Publishing, 1992)*, pages 291–309, Cambridge University Press, UK, 1992.
- [Wan96] Xinxin Wang. *Tabular Abstraction, Editing, and Formatting*. PhD thesis, Dept. of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, 1996. Also issued as Technical Report CS-96-09, University of Waterloo.

[WW96] X. Wang and D. Wood. XTABLE—A tabular editor and formatter. To appear, *EP96 (Proceedings of Electronic Publishing, 1996)*, 1996.