An Octet Degree Graph Representation for the Rectangular Dissections

TOMOE MOTOHASHI
Kanto Gakuin University
TOMOKAZU ARITA
Obirin University
KENSEI TSUCHIDA
Toyo University
and
TAKEO YAKU
Nihon University

A considerable number of data structures have been introduced for models of rectangular dissections. However, editing operations such as column insertion have not been effectively formalized for known data structures (see Appendix).

In this paper, we propose an attribute graph as another type of data structure for the rectangular dissections with heterogeneous cells over the global meshes that performs editing and drawing. We call the graphs rectangular dissection graphs. We also give algorithms for basic operations in table editing. It is shown that our column insertion algorithm effectively executes an "expected " column insertion operation, and runs in $O(\sqrt{n})$ time, while known algorithm runs in $O(n)$ time for the n cell square rectangular dissections.

Several other algorithms are proposed, including one where the cell unifying algorithm runs in $O(1)$ time, while a known algorithm runs in $O(n)$ time for the n cell rectangular dissections.

Categories and Subject Descriptors: E.1 [**Data Structures**]: Graphs and Networks; I.7.2 [**Document and Text Processing**]: Document Preparation; J.6 [**Computer Applications**]: Computer-aided engineering

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Rectangular dissections, rectangular dissection graphs, table interface

1. INTRODUCTION

Rectangular dissections with heterogeneous cells over global meshes are commonly used in information processing such as tables in documentation and floor plans in operation researches. The importance of rectangular dissection processing in-

---

Authoe's address: T. Motohashi, 1-50-1, Mutsuura-Higashi, Kanazawa-ku, Yokohama, Kanagawa, 236-8501, Japan, e-mail tomoe@kanto-gakuin.ac.jp
T. Arita, 3758, Tokiwa-machi, Machida, Tokyo, 194-0294, Japan, e-mail arita@obirin.ac.jp
K. Tsuchida, 2100, Kujirai, Kawagoe, Saitama, 350-8585, Japan, e-mail kensei@eng.toyo.ac.jp
T. Yaku, 3-25-40, Sakurajosui, Setagaya-ku, Tokyo, 156-8550, Japan, e-mail yaku@cs.chs.nihon-u.ac.jp

creases in accordance with development of both document processing and operation researches.

Many authors have introduced various data structures in order to effectively formalize operations and to efficiently execute operations on rectangular dissections. Quad trees were introduced for data search [de Berg, van Kreveld, Overmats and Schwarzkoph 1997]. Rectangular dual graphs were introduced for floor layout [Brandenburg 1994],[Kozminski and Kinnen 1985]. Another model called wall representation has been introduced for a wall move (e.g. [Kundu 1998]).

However, those data structures could not effectively formalize several operations on rectangular dissections. For example, column insertion operation is not often effectively executed in word processors (see Appendix). That is, users often obtain unexpected results. And operations on those data structures may not be executed efficiently. That is, several operations seem to execute unnecessary operations and run in too large complexity.

We propose in this paper octed degree graphs called rectangular dissection graphs as a data structure of the rectangular dissections over global meshes. Several operations on rectangular dissections are effectively formalized such as column insertion on the data structure. Several operations are formalized in lower complexity on the data structure. Wall move and column insertion are both executed in $O(\sqrt{n})$ time, while the operations are executed in $O(n)$ time in known data structures [Kozminski and Kinnen 1985] for the n cell square rectangular dissections. Cell unifying is executed in $O(1)$ time, while the operation is executed in $O(log(n))$ time in quad tree model in [de Berg, van Kreveld, Overmats and Schwarzkoph 1997].

Accordingly, we claim that the rectangular dissection graphs may be a important model for rectangular dissection transformations.

We introduce the rectangular graphs for rectangular dissections. Our rectangular dissction graphs have two particular features. The first feature is that only the nodes with the same wall coordinates are connected by edges. The second feature is that the degree of nodes is bounded by eight.

By the first feature, transform operations such as column insertion are effectively formalized. By the second feature, efficient algorithms are constructed such as the $O(1)$ cell unifying algorithm and the $O(\sqrt{n})$ column insertion algorithm, as the known model drive $O(n)$ algorithms.

This model and algorithms are capable to be widely used for existing information processing systems. This model and algorithms increase effectiveness and efficiency of table processing systems.

Section 2 proposes a representation of tables by an attribute multi-edge graph. Several properties of the graphs are shown.

In Section 3, several algorithms that execute table editing based on the representation are shown. We provide algorithms for unifying cells, moving east wall, changing the column width and the insertion column.

Section 4 provides conclusions.

2. OCTET DEGREE GRAPHS FOR RECTANGULAR DISSECTIONS

We provide this section for several definitions concerning a selected table.

{(1,1)}	{(1,2)}	{(1,3)}
{(2,1)}	{(2,2)}	{(2,3)}

Fig. 1. A Partition P_1

<table>
<tr><td rowspan="2">{(1,1),
(2,1)}</td><td>{(1,2)}</td><td>{(1,3)}</td></tr>
<tr><td colspan="2">{(2,2),(2,3)}</td></tr>
</table>

Fig. 2. A Partition P_2

Definition 2.1 An $(s, t)-$ *table* is a set $\{(i, j)|1 \leq i \leq s, 1 \leq j \leq t\}$ of integer pairs. A *table* is an $(s, t)-$ table for some s and t. A *partial table* is a subset S of an $(s, t)-$ table, where S is in the form of $\{(i, j)|u \leq i \leq v, x \leq j \leq y\}$ for integers $1 \leq u, v \leq s$, $1 \leq x, y \leq t$. A *partition* P over a table T is a pairwise disjoint collection $S_1, S_2, \ldots, S_N$ of partial tables, where $S_1 \cup S_2 \cup \ldots \cup S_N = T$, and each S_i is called a *cell*. We call s the *row number* and t the *column number* of T.

Example 2.1 Figure 1 illustrates a partition

$$P_1 = \{\{(1, 1)\}, \{(1, 2)\}, \{(1, 3)\}, \{(2, 1)\}, \{(2, 2)\}, \{(2, 3)\}\}$$

over the $(2, 3)-$table T.

Example 2.2 Figure 2 illustrates a partition

$$P_2 = \{\{(1, 1), (2, 1)\}, \{(1, 2)\}, \{(1, 3)\}, \{(2, 2), (2, 3)\}\}$$

over the $(2, 3)-$table T.

Definition 2.2 The *row grid* of an $(s, t)-$ table T is a map $g_{row} : \{0, 1, \ldots, s\} \rightarrow \mathbf{R}$ such that $g_{row}(i) \leq g_{row}(i + 1)$ for $0 \leq i \leq s - 1$. The *column grid* is a map $g_{column} : \{0, 1, \ldots, t\} \rightarrow \mathbf{R}$ such that $g_{column}(j) \leq g_{column}(j + 1)$ for $0 \leq j \leq t - 1$. A *grid* is a pair $g = (g_{row}, g_{column})$.

A *tabular diagram* is a triple $D = (T, P, g)$ of a table T, a partition P over T, and a grid g of T.

Terminology Let c be a cell $c = \{(i, j)|u \leq i \leq v, x \leq j \leq y\}$.

The *north wall* $nw(c)$ of c denotes $g_{row}(u-1)$. The *south wall* $sw(c)$ of c denotes $g_{row}(v)$. The *east wall* $ew(c)$ of c denotes $g_{column}(y)$. The *west wall* $ww(c)$ of c denotes $g_{column}(x - 1)$.

The *location* of c is a pair $(nw(c), ww(c))$. The $height(c)$ denotes $sw(c) - nw(c)$, and $width(c)$ denotes $ew(c) - ww(c)$. The *location* of an $(s, t)-$ table is a pair $(g_{row}(0), g_{column}(0))$.

We show figures of tabular diagrams.

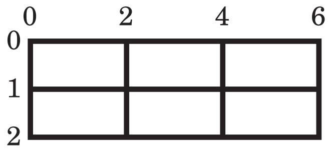


Fig. 3. A Tabular Diagram $D_1 = (T_1, P_1, g_1)$

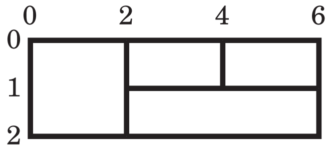


Fig. 4. A Tabular Diagram $D_2 = (T_2, P_2, g_2)$

Example 2.3 Figure 3 illustrates a tabular diagram $D_1 = (T_1, P_1, g_1)$, where $g_{1row}(0) = 0, g_{1row}(1) = 1$, $g_{1row}(2) = 2$, and $g_{1column}(0) = 0, g_{1column}(1) = 2$, $g_{1column}(2) = 4$, $g_{1column}(3) = 6$. The numbers represent the grid coordinates.

Example 2.4 Figure 4 illustrates a tabular diagram $D_2 = (T_2, P_2, g_2)$ corresponding to the partition P_2.

In the latter part of this paper, we consider a certain type of tabular diagrams in order to deal with effective algorithms for table editing.

Condition 2.1 Let $D = (T, P, g)$ be a tabular diagram. D satisfies the following conditions.
T is a $(s, t)-$ table, where $s, t \geq 3$.
P has $2s + 2t - 4$ *perimeter cells* each of which is in the form of $\{(i, j)\}$
satisfying one of the following conditions :
(1) $i = 1,\ 1 \leq j \leq t$, (2) $i = s,\ 1 \leq j \leq t$,
(3) $1 \leq i \leq s,\ j = 1$ or (4) $1 \leq i \leq s,\ j = t$.
For a perimeter cell $c = \{(i, j)\}$,
$width(c) = 0$ if $i = 1$ or s, and $height(c) = 0$ if $j = 1$ or t.

The following examples show tabular diagrams satisfying Condition 2.1. With respect to table drawing, they correspond to tabular diagrams without perimeter cells as in previous examples.

Example 2.5 Figure 5 illustrates a tabular diagram $D_{1p} = (T_{1p}, P_{1p}, g_{1p})$ with perimeter cells, where $g_{1p\ row}(0) = 0, g_{1p\ row}(1) = 0, g_{1p\ row}(2) = 1$, $g_{1p\ row}(3) = 2, g_{1p\ row}(4) = 2$ and $g_{1p\ column}(0) = 0, g_{1p\ column}(1) = 0, g_{1p\ column}(2) = 2$, $g_{1p\ column}(3) = 4, g_{1p\ column}(4) = 6, g_{1p\ column}(5) = 6$.
The numbers represent the grid coordinates. The tabular diagram D_{1p} corresponds to D_1 in Example 2.3.

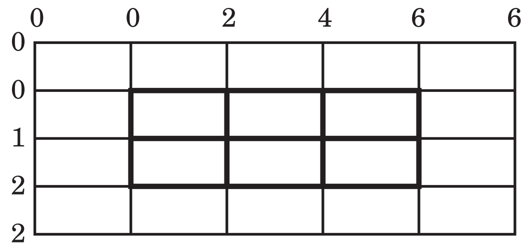


Fig. 5. A Tabular Diagram D_{1p} with Perimeter Cells

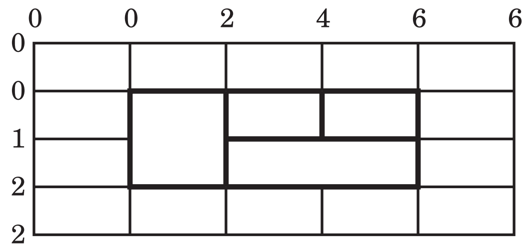


Fig. 6. A Tabular Diagram D_{2p} with Perimeter Cells

Example 2.6 Figure 6 illustrates tabular diagram D_{2p} with perimeter cells, which corresponds to D_2 in Example 2.4.

Editing and drawing are often effectively executed, when tabular diagrams are represented by graphs. Consequently, we term now to graph representation.

Now, we introduce an attribute graph. Then, we show how to represent a tabular diagram with an attribute graph.

Definition 2.3 An *attribute graph* is a 6−tuple $G = (V, E, L, \lambda, A, \alpha)$, where
(V, E) is a *multi-edge undirected graph*,
L is the set of *labels* for edges,
$\lambda : E \rightarrow L$ is the *label function*,
A is the set of *attributes*, and
$\alpha : V' \rightarrow A$ is the *attribute map*, where V' is a subset of V.

A tabular diagram $D = (T, P, g)$ is *represented* as an attribute graph $G_D = (V_D, E_D, L, \lambda_D, A, \alpha_D)$, where V_D is identified by a partition P (we denote a node corresponding to a cell c in P by v_c, we call v_c a *perimeter node* (resp. *inner node*) if c is a perimeter cell (resp. inner cell)), E_D is defined by Rules 1-4, $L = \{enw, esw, eew, eww\}$, $\lambda_D : E_D \rightarrow L$ is defined by Rules 1-4, $A = R^4$, and $\alpha_D : V_{1,*} \cup V_{*,1} \rightarrow R^4$ are defined by $\alpha_D(v_c) = (nw(c), sw(c), ew(c), ww(c))$ for $v_c \in V_{1,*} \cup V_{*,1}$, where $V_{1,*}$ is the set of perimeter nodes corresponding to the perimeter cells in the 1st row, and $V_{*,1}$ is the set of perimeter nodes corresponding to the perimeter cells in the 1st column.

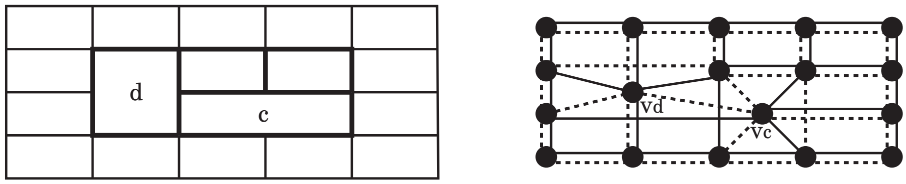


Fig. 7. Tabular Diagram and Its Corresponding Rectangular Dissection Graph

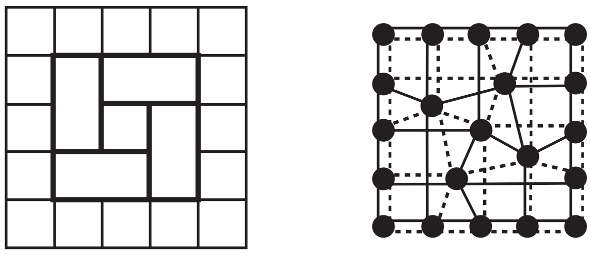


Fig. 8. Tabular Diagram and Its Corresponding Rectangular Dissection Graph

For simplicity, we also use $nw(v_c)$, $sw(v_c)$, $ew(v_c)$, and $ww(v_c)$ instead of $nw(c)$, $sw(c)$, $ew(c)$, and $ww(c)$, respectively.

Rule 1 If $nw(c) = nw(d)$ and there is no cell between c and d having an equal north wall, then $[v_c, v_d]$ is in E_D and $\lambda_D[v_c, v_d] = enw$. In this case, $[v_c, v_d]$ is called a *north wall edge*.

Rule 2 If $sw(c) = sw(d)$ and there is no cell between c and d having an equal south wall, then $[v_c, v_d]$ is in E_D and $\lambda_D[v_c, v_d] = esw$. In this case, $[v_c, v_d]$ is called a *south wall edge*.

Rule 3 If $ew(c) = ew(d)$ and there is no cell between c and d having an equal east wall, then $[v_c, v_d]$ is in E_D and $\lambda_D[v_c, v_d] = eew$. In this case, $[v_c, v_d]$ is called an *east wall edge*.

Rule 4 If $ww(c) = ww(d)$ and there is no cell between c and d having an equal west wall, then $[v_c, v_d]$ is in E_D and $\lambda_D[v_c, v_d] = eww$. In this case, $[v_c, v_d]$ is called a *west wall edge*.

An attribute graph G_D is called a *rectangular dissection graph* (a *tessellation graph* [Kirishima, Motohashi, Tsuchida and Yaku 2002]). Note that the degree of each node v in G_D is at most 8.

Example 2.7 Figure 7 shows a tabular diagram and its corresponding rectangular dissection graph. With the arrangement of the vertices, we represent south wall edges and west wall edges by dotted lines.

Example 2.8 Figure 8 shows a tabular diagram and its corresponding rectangular dissection graph.

Proposition 2.1 *Let G_D be a rectangular dissection graph for a tabular diagram D of the $(s, t)-$table. G_D is not generally a planar graph.*

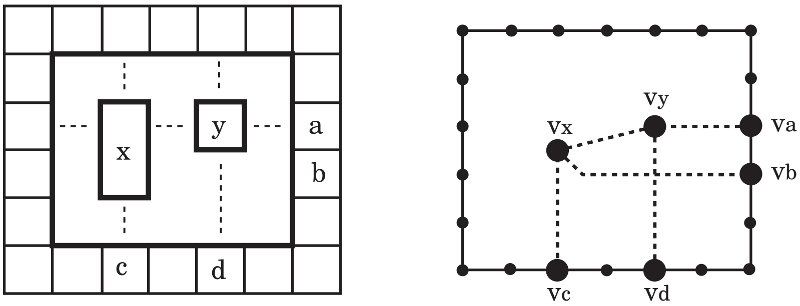


Fig. 9. Tabular Diagram and Subgraphs of Its Corresponding Rectangular Dissection Graph

Proof. Figure 9 represents a part of a tabular diagram. These two inner cells x and y have an equal north wall, but do not have equal south walls. Its corresponding rectangular dissection graph G contains a subdivision of a bipartite graph $K_{3,3}$. By the Kratowski's theorem, the rectangular dissection graph is not planar.

Q.E.D.

Note that we consider tabular diagrams with perimeter cells. Then,

Proposition 2.2 *Let G_D be a rectangular dissection graph for a tabular diagram D of the $(s,t)-$table. Let k be the number of inner cells in G_D.* For the number $\#E_D$ of edges in G_D, we have
$2\#E_D = 6(2s-4) + 6(2t-4) + 8k + 16.$

Proof. The degree of inner nodes in G_D is equal to 8. The degree of the perimeter nodes except the one in the corner is equal to 6. The degree of the nodes in the corner is equal to 4. Since $2\#E_D$ is equal to the sum of the degree of the nodes in V_D, the proposition is verified.

Q.E.D.

3. ALGORITHMS

We assume that the edges in rectangular dissection graphs are ordered, that is, a leftward equal north edge and a rightward equal north edge are identified, for example. This section provides algorithms for rectangular dissection graphs. The following algorithm unifies two adjacent inner cells in a tabular diagram.

ALGORITHM UnifyCells(G_D, v_x, v_y, G_E)

INPUT
$G_D = (V_D, E_D, L, \lambda_D, A, \alpha_D)$: a rectangular dissection graph for a tabular diagram D,
v_x : a node in G_D corresponding to an inner cell x,

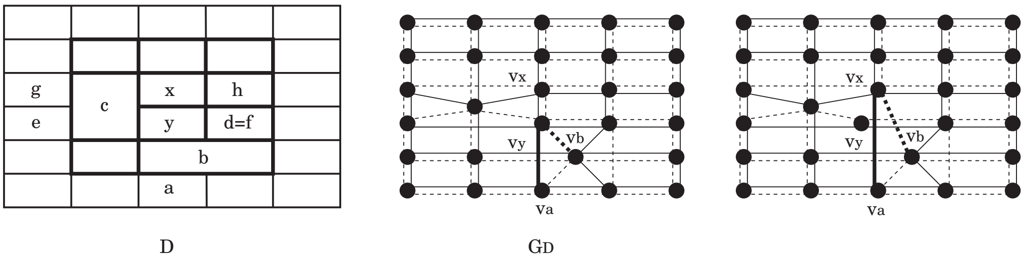


Fig. 10. A Change of Vertical Edges of v_y after PHASE 2 of Algorithm UnifyCells

v_y : a node in G_D corresponding to an inner cell y which is adjacent to the south side of x such that $ww(x) = ww(y), ew(x) = ew(y)$, and $sw(x) = nw(y)$.

OUTPUT
$G_E = (V_E, E_E, L, \lambda_E, A, \alpha_E)$: a rectangular dissection graph for a tabular diagram E, where E is obtained from D by unifying cells x and y into x.

METHOD
begin
/∗ PHASE 1 ∗/
Initially let $G_E \leftarrow G_D$;
let v_a, v_b be lower nodes linked to v_y by a vertical edge ;
let v_c be a westside node linked to v_y by a south wall edge ;
let v_d be an eastside node linked to v_y by a south wall edge ;
let v_e be a westside node linked to v_y by a north wall edge ;
let v_f be an eastside node linked to v_y by a north wall edge ;
let v_g be a westside node linked to v_x by a south wall edge ;
let v_h be an eastside node linked to v_x by a south wall edge ;
/∗ PHASE 2 ∗/
/∗ change of vertical edges concerning to v_y ∗/
delete two vertical edges $[v_y, v_a]$ and $[v_y, v_b]$ from E_E ;
add edges $[v_x, v_a]$ and $[v_x, v_b]$ to E_E ;
put $\lambda_E[v_x, v_a] \leftarrow \lambda_D[v_y, v_a]$, and $\lambda_E[v_x, v_b] \leftarrow \lambda_D[v_y, v_b]$;
delete two vertical edges between v_x and v_y from E_E ; (See Figure 10)
/∗ PHASE 3 ∗/
/∗ change of south wall edges concerning to v_y ∗/
delete south wall edges $[v_c, v_y]$ and $[v_d, v_y]$ from E_E;
add $[v_c, v_x]$ and $[v_x, v_d]$ to E_E ;
put $\lambda_E[v_c, v_x] \leftarrow esw, \lambda[v_x, v_d] \leftarrow esw$;
/∗ change of north wall edges concerning to v_y ∗/
delete north wall edges $[v_e, v_y]$ and $[v_y, v_f]$ from E_E ;
add an edge $[v_e, v_f]$ to E_E ;
put $\lambda_E[v_e, v_f] \leftarrow enw$;
/∗ change of south wall edges concerning to v_x ∗/
delete south wall edges $[v_g, v_x]$ and $[v_x, v_h]$ from E_E ;

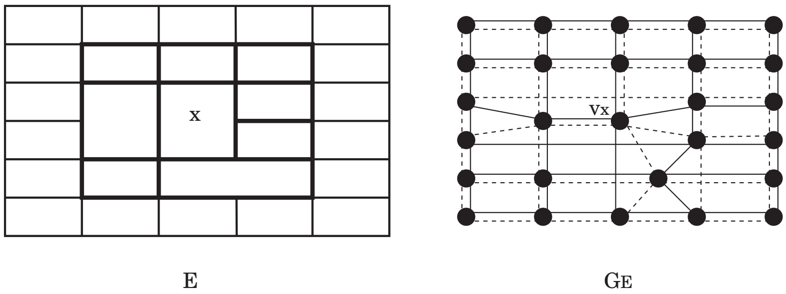


Fig. 11. Output Graph G_E in Algorithm UnifyCells

add an edge $[v_g, v_h]$ to E_E ;
put $\lambda_E[v_g, v_h] \leftarrow esw$;
/* delete of the node v_y */
delete the node v_y from G_E (See Figure 11)
end.

Theorem 3.1 *Let D be a tabular diagram of the (s, t)-table, and x be an inner cell in D. Suppose that there is an inner cell y adjacent to the south side of x such that $ew(x) = ew(y)$, $ww(x) = ww(y)$ and $sw(x) = nw(y)$. Let E be a tabular diagram obtained from D by the unifying cells x and y into x. Let G_D and G_E be the rectangular dissection graphs for D and E. Then G_E is obtained from G_D in constant time.*

Proof. The theorem is verified by Algorithm UnifyCells. G_E is obtained from G_D by the unifying nodes v_x and v_y into v_x, involving with changing edges, and the label of the edges. Since we look at rows containing the two nodes v_x and v_y, and the degree of nodes in rectangular dissection graphs is at most 8, then G_E is obtained from G_D in constant time.

Q.E.D.

The following algorithm executes a wall movement of a tabular diagram.

ALGORITHM MoveEastWall(G_D, v_x, δ, G_E)

INPUT
$G_D = (V_D, E_D, L, \lambda_D, A, \alpha_D)$: a rectangular dissection graph for a tabular diagram D,
v_x : a node in G_D corresponding to a cell x, where x is not in the 1-st column, the last column, and the second last column.
$\delta \geq 0$: a movement value.
Suppose $\Delta > \delta$, where $\Delta > 0$ is the width of a perimeter cell in the column adjacently located at the east-side of c.

OUTPUT
$G_E = (V_E, E_E, L, \lambda_E, A, \alpha_E)$: a rectangular dissection graph for a tabular diagram

E obtained from D by the east wall movement using δ of cells that have the equal east wall for x.

METHOD
begin
Initially, let $G_E \leftarrow G_D$;
let v_a be the northmost node linked to v_x by east wall edges ;
let v_b be a eastside node linked to v_a by a north wall edge ;
/$*$ change attribute for v_a $*$/
put $ew(v_a) \leftarrow ew(v_a) + \delta$;
/$*$ change attribute for v_b $*$/
put $ww(v_b) \leftarrow ww(v_b) + \delta$
end.

Theorem 3.2 *Let D be a tabular diagram of the (s,t)-table, and x be a cell in D, where x is not in the 1-st column, the last column, and the second to last column in D. Let $\delta \geq 0$.*

Suppose $\Delta > \delta$, where $\Delta > 0$ is the width of a perimeter cell in the column adjacently located at the east-side of c.

Let E be a tabular diagram obtained from D by the east wall movement using δ of cells that have the equal east wall for x. Let G_D and G_E be the rectangular dissection graphs for D and E, respectively. Then G_E is obtained from G_D in $O(s)$ time by the algorithm MoveEastWall.

Proof. The theorem is verified using Algorithm MoveEastWall.

Q.E.D.

The following algorithm executes a changing width of a column of a tabular diagram.

ALGORITHM ChangeColumnWidth (G_D, v_x, δ, G_E)

INPUT
$G_D = (V_D, E_D, L, \lambda_D, A, \alpha_D):$ a rectangular dissection graph for a tabular diagram D,
$v_x :$ a node in G_D corresponding to a cell x, where x is not in the first column and the last column,
$\delta :$ a movement value.
Suppose $\Delta + \delta > 0$, where Δ is the width of a perimeter cell in the column which has equal east wall for x.

OUTPUT
$G_E = (V_E, E_E, L, \lambda_E, A, \alpha_E):$ a rectangular dissection graph for a tabular diagram E obtained from D by the changing width using δ of cells that have an equal east wall for x.

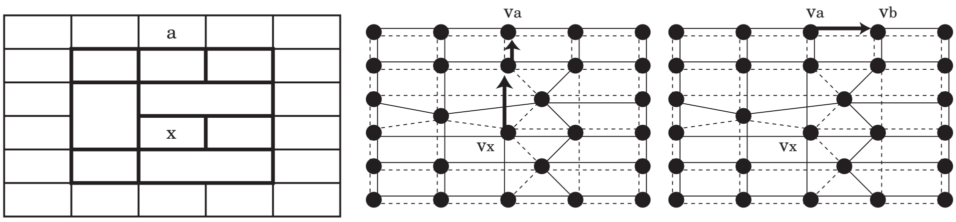


Fig. 12. Traversal of Nodes Upward through East Wall Edges from v_x

METHOD
begin
  Initially, let $G_E \leftarrow G_D$;
  let v_a be the northmost node linked to v_x by east wall edges (see Figure 12) ;
  /∗ change attribute ∗/
  put $ew(v_a) \leftarrow ew(v_a) + \delta$;
  let v_b be an east-side node linked to v_a by a north wall edge ;
  while v_b is not a node in the northeast corner **do**
    /∗ west wall ∗/
    put $ww(v_b) \leftarrow ww(v_b) + \delta$;
    /∗ east wall ∗/
    put $ew(v_b) \leftarrow ew(v_b) + \delta$;
    change v_b to the eastside node linked to v_b by a north wall edge
  end{while} ;
  /∗ for a node in the northeast corner ∗/
  put $ww(v_b) \leftarrow ww(v_b) + \delta$;
  put $ew(v_b) \leftarrow ew(v_b) + \delta$
end.

Theorem 3.3 *Let D be a tabular diagram of the (s,t)-table, and x be a cell in D, where x is not in the first column and the last column. Let δ be a movement value. Suppose $\Delta + \delta > 0$, where $\Delta > 0$ is the width of a perimeter cell in the column which has equal east wall to x. Let E be a tabular diagram obtained from D by the changing width using δ of cells that have an equal east wall for x. Let G_D and G_E be the rectangular dissection graphs for D and E, respectively. Then G_E is obtained from G_D in $O(s+t)$ time by the algorithm* ChangeColumnWidth.

Proof. Algorithm ChangeColumnWidth runs in $O(s+t)$ time since there are at most t nodes at the east side of x. G_E is obtained from G_D by changing the attribute of the nodes. Since the degree of nodes in rectangular dissection graphs is at most 8, then G_E is obtained from G_D in $O(s+t)$ time.

Q.E.D.

Corollary 3.1 *Let D be a tabular diagram of the $(\sqrt{n}, \sqrt{n})$-table, and x be a cell in D, where x is not in the first column and the last column. Let δ be a movement value. Suppose $\Delta + \delta > 0$, where $\Delta > 0$ is the width of a perimeter cell in the*

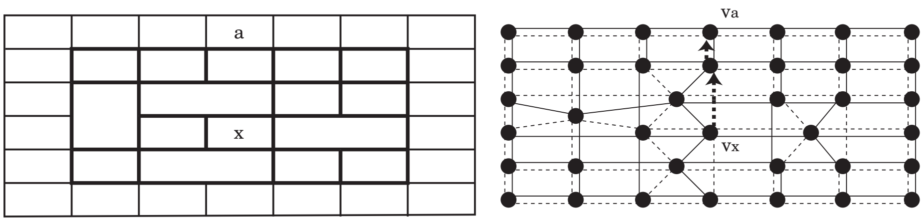


Fig. 13. Traversal of Nodes Upward through West Wall Edges from v_x

column which has equal east wall to x. Let E be a tabular diagram obtained from D by the changing width using δ of cells that have an equal east wall for x. Let G_D and G_E be the rectangular dissection graphs for D and E, respectively. Then G_E is obtained from G_D in $O(\sqrt{n})$ time by the algorithm ChangeColumnWidth, *where n is the maximal number of cells in a $\sqrt{n} \times \sqrt{n}$ square rectangular dissection D.*

The following algorithm executes insertion of a column at the west side of a focused cell into the tabular diagram.

ALGORITHM InsertColumn(G_D, v_x, G_E)

INPUT
G_D : a rectangular dissection graph for a tabular diagram $D = (T, P, g)$,
v_x : a node in G_D corresponding to a cell x, where x is not in the first column.

OUTPUT
G_E : a rectangular dissection graph for E, where E is a tabular diagram obtained from D by the insertion of a column with width δ at the west side of x, where δ is the width of a perimeter cell in the column including x.

METHOD
begin
  Initially, put $G_E \leftarrow G_D$;
  let v_a be the northmost node linked to v_x by east wall edges (see Figure 13) ;
  let δ be the width of the cell corresponding to v_a ;
  put $v_0 \leftarrow v_a$;
  add a node u_0 ;
  put $i \leftarrow 0$;
  /∗ insert a column ∗/
  while a node v_i is not the lowermost node **do**
    let w_i be an adjacently westside node linked to v_i by a north wall edge ;
    delete a north wall edge $[w_i, v_i]$;
    add $[w_i, u_i]$ to E_E ;
    put $\lambda_E[w_i, u_i] \leftarrow enw$;
    deform G_E similarly for a south wall edge ;
    add a north wall edge and south wall edge between u_i and v_i ;
    let v_{i+1} be a lower node linked to v_i by a west wall edge ;

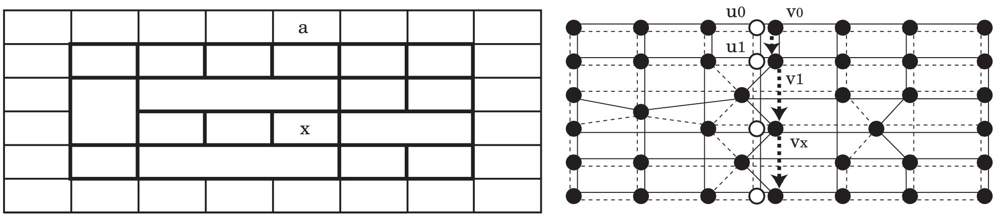


Fig. 14. Insertion of the Column at West-Side of x

add a node u_{i+1} ;
add a west wall edge and east wall edge between u_i and u_{i+1} ;
$i \leftarrow i+1$
end ; (see Figure 14)
/∗ for the lowermost node ∗/
let w_i be an adjacently westside node linked to v_i by a north wall edge ;
delete a north wall edge $[w_i, v_i]$;
add $[w_i, u_i]$ to E_E ;
put $\lambda_E[w_i, u_i] \leftarrow enw$;
deform G_E similarly for a south wall edge ;
add a north wall edge and south wall edge between u_i and v_i ;
/∗ the existing column shifts to the east ∗/
let u_0 be the uppermost node in u_i's ;
put $G_{E_0} \leftarrow G_E$;
ChangeColumnWidth$(G_{E_0}, u_0, \delta, G_E)$;
end.

Theorem 3.4 *Let D be a tabular diagram of the (s,t)-table, and x be a cell in D, where x is not in the first column. Suppose that E is the tabular diagram obtained from D by the insertion of a column with width δ at the west side of the column including x, where δ is the width of a perimeter cell of the column including x. Let G_D and G_E be the rectangular dissection graphs for D and E, respectively. Then G_E is obtained from G_D in $O(s+t)$ time.*

Proof. The theorem is verified from Algorithm InsertColumn.

Q.E.D.

The following algorithm executes horizontal splitting of an inner cell in a tabular diagram.

ALGORITHM HSplitCell(G_D, v_x, G_E)

INPUT
$G_D = (V_D, E_D, L, \lambda_D, A, \alpha_D)$: a rectangular dissection graph for a tabular diagram D,
v_x : a node in G_D corresponding to an inner cell x.

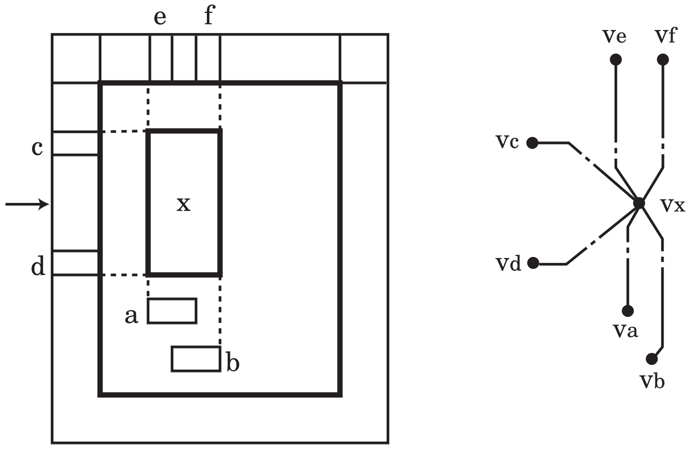


Fig. 15. Nodes with respect to v_x

OUTPUT
$G_E = (V_E, E_E, L, \lambda_E, A, \alpha_E)$: a rectangular dissection graph for a tabular diagram E, where E is obtained from D by splitting cell x horizontal into x and y.

METHOD
begin
Initially let $G_E \leftarrow G_D$;
let v_a, v_b be lower nodes linked to v_x by a vertical edge ;
let v_c be the westmost node linked to v_x by north wall edges ;
let v_d be the westmost node linked to v_x by south wall edges ;
let v_e be the uppermost node linked to v_x by west wall edges ;
let v_f be the uppermost node linked to v_x by east wall edges ;
put a vertex v_y in V_E (See Figure 15) ;
/* change of vertical edges concerning to v_x and v_y */
delete two vertical edges $[v_x, v_a]$ and $[v_x, v_b]$ from E_E ;
add edges $[v_y, v_a]$ and $[v_y, v_b]$ to E_E ;
put $\lambda_E[v_y, v_a] \leftarrow \lambda_D[v_x, v_a]$, and $\lambda_E[v_y, v_b] \leftarrow \lambda_D[v_x, v_b]$;
add an east wall edge and a west wall edge between v_x and v_y to E_E ;
/* change of south wall edges concerning to v_x */
put $\delta \leftarrow nw(c) + \frac{sw(d) - nw(c)}{2}$;
put $i \leftarrow 1$;
put $v_i \leftarrow v_c$;
while $sw(v_i) < \delta$ **do**
  $i \leftarrow i + 1$;
  let v_i be a lower node linked to v_{i-1} by a west wall edge ;
end{while} ;
/* the first case */
if $sw(v_i) > \delta$, **then**
  let w_{i-1} be the eastmost node linked to v_{i-1} by north wall edges ;

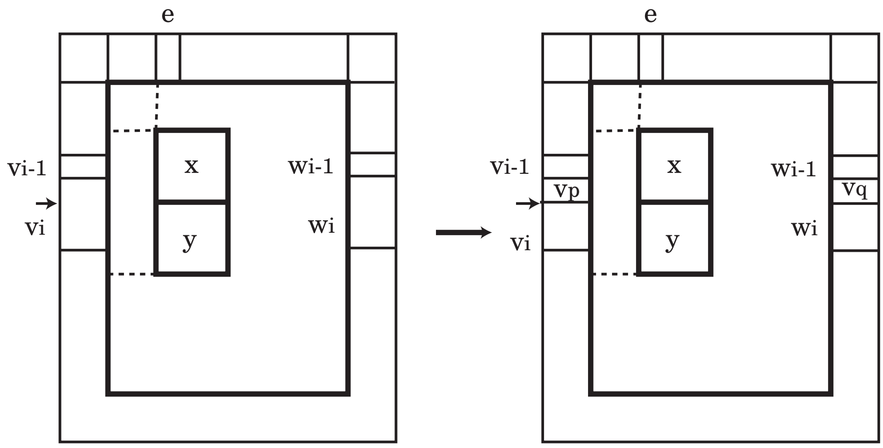


Fig. 16. Illustration for the first example

let w_i be a lower node linked to w_{i-1} by a west wall edge ;
let v_g be an eastside node linked to v_i by a north wall edge ;
let v_h be a westside node linked to w_i by a north wall edge ;
/* new perimeter node v_p and v_q */
add two nodes, v_p and v_q to V_E (See Figure 16) ;
add an east wall edge and a west wall edge between v_{i-1} and v_p to E_E ;
add an east wall edge and a west wall edge between v_p and v_i to E_E ;
put $nw(p) \leftarrow sw(v_{i-1})$;
put $sw(p) \leftarrow \delta$;
put $nw(v_i) \leftarrow \delta$;
delete vertical two edges between v_{i-1} and v_i from E_E ;
delete a north wall edge $[v_i, v_g]$ from E_E ;
add a north wall edge $[v_p, v_g]$ to E_E ;
add an east wall edge and a west wall edge between w_{i-1} and v_q in E_E ;
add an east wall edge and a west wall edge between v_q and w_i in E_E ;
delete vertical two edges between w_{i-1} and w_i from E_E ;
delete a north wall edge $[v_h, w_i]$ from E_E ;
add a north wall edge $[v_h, v_q]$ to E_E ;
add south wall edges $[v_p, v_x]$ and $[v_x, v_q]$ in E_E ;
add north wall edges $[v_i, v_y]$ and $[v_y, w_i]$ in E_E ;
/* the second case */
else ($sw(v_i)$ is equal to δ)
let v_{NW} be the node in the northwest corner ;
/* for south wall edges concerning to v_x */
put $j \leftarrow 1$;
put $w_j \leftarrow v_{NW}$;
put $x_j \leftarrow v_i$;
while $ww(w_j) < ww(e)$ **do**
$j \leftarrow j + 1$;

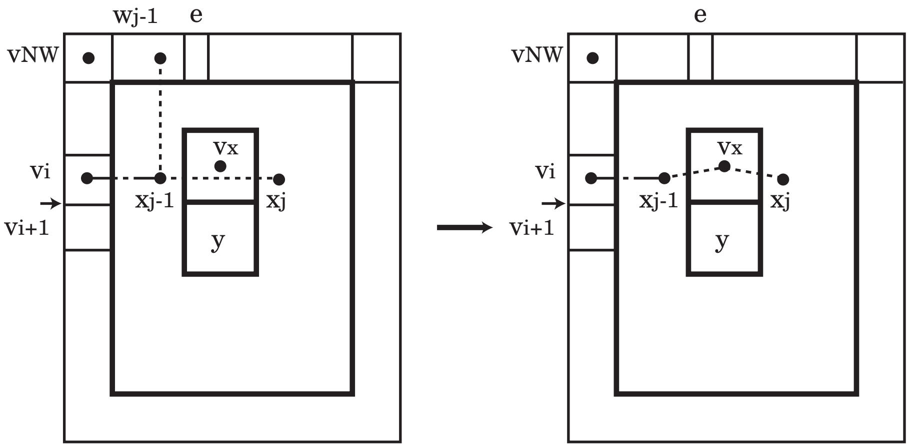


Fig. 17. Illustration for the second example

let x_j be a eastside node linked to x_{j-1} by a south wall edge ;
let w_j be the uppermost node linked to x_j by west wall edges ;
end{while} (See Figure 17) ;
add two south wall edges $[x_{j-1}, v_x]$ and $[v_x, x_j]$ in E_E;
delete a south wall edge $[x_{j-1}, x_j]$ from E_E ;
/* for north wall edges concerning to v_y */
let v_{i+1} be a lower node linked to v_i by a west wall edge ;
put $k \leftarrow 1$;
put $w_k \leftarrow v_g$;
put $y_k \leftarrow v_{i+1}$;
while $ww(w_k) < ww(e)$ **do**
$k \leftarrow k + 1$
let y_k be a eastside node linked to y_{k-1} by a north wall edge ;
let w_k be the uppermost node linked to y_k by west wall edges ;
end{while} ;
add two north wall edges $[y_{k-1}, v_y]$ and $[v_y, y_k]$ in E_E;
delete a north wall edge $[y_{k-1}, y_k]$ from E_E
end{if}
end.

Theorem 3.5 *Let D be a tabular diagram of (s,t)-table, and x be an inner cell in D. Let E be a tabular diagram obtained from D by the splitting cell x horizontal into x and y. Let G_D and G_E be the rectangular dissection graphs for D and E. Then G_E is obtained from G_D in $O(st)$ time.*

Proof. The theorem is verified from Algorithm HSplitCell. G_E is obtained from G_D by the splitting node horizontal into x and y, involving with changing edges, and the label of the edges. Since we look at nodes each of which is in the north west side of v_x, and the degree of nodes in rectangular dissection graphs is at most 8, then G_E is obtained from G_D in $O(st)$ time.

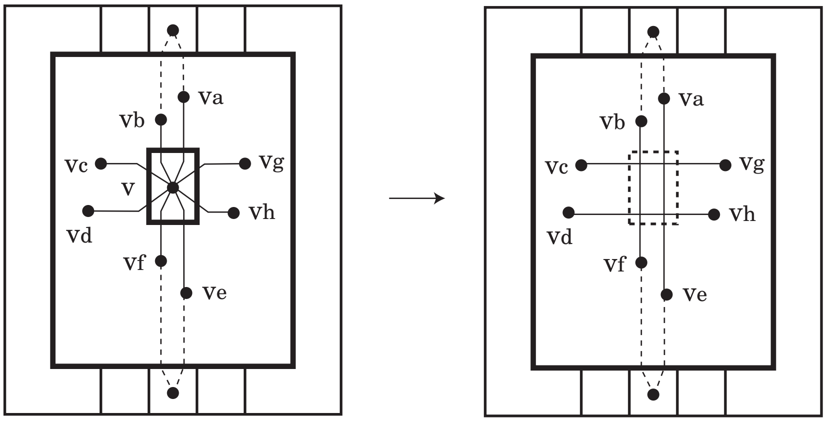


Fig. 18. Delete Isolated Cell

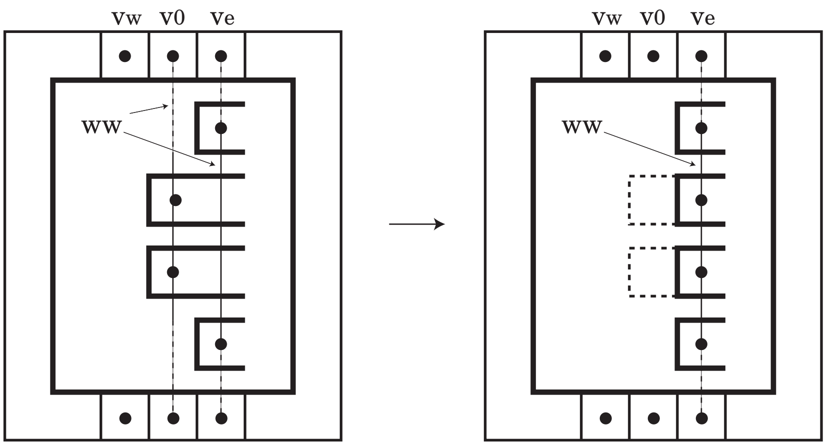


Fig. 19. West Wall Traversal

Q.E.D.

The following algorithm executes deletion of a column containing a focused cell from the tabular diagram.

ALGORITHM DeleteColumn(G_D, v_x, G_E)

INPUT
G_D : a rectangular dissection graph for a tabular diagram $D = (T, P, g)$,
v_x : a node in G_D corresponding to a cell x, where δ is the width of the leftmost perimeter cells in the column including x.

OUTPUT

G_E : a rectangular dissection graph for E, where E is a tabular diagram obtained from D by the deletion of a column containing x

METHOD
begin
  Initially, put $G_E \leftarrow G_D$;
  traverse upward through the west wall edges from v_x until a perimeter node v_0 ;
  put $\sigma_w \leftarrow ww(v_0)$;
  put $\sigma_e \leftarrow ew(v_0)$;
  let v_w be an adjacently west-side node linked to v_0 by a north wall edge ;
  let v_e be an adjacently east-side node linked to v_0 by a north wall edge ;
  /* delete a column */
  /* delete cells of unit width located southern side of v_0 */
  mark all vertices linked by west wall edges from v_0 "W" ;
  initialize the head to v_0 ;
  while the heads point inner cells **do**
    move the heads downward through east wall edge ;
    if the head a vertex v marked "W" v **then**
      let a be north-side node adjacently linked by east wall edge from v;
      let b be north-side node adjacently linked by west wall edge from v;
      let c be west-side node adjacently linked by north wall edge from v;
      let d be west-side node adjacently linked by south wall edge from v;
      let e be south-side node adjacently linked by east wall edge from v;
      let f be south-side node adjacently linked by west wall edge from v;
      let g be east-side node adjacently linked by north wall edge from v;
      let h be east-side node adjacently linked by south wall edge from v;
      delete the edges around v ;
      add $[a, e]$ to E_E ;
      add $[b, f]$ to E_E ;
      add $[c, g]$ to E_E ;
      add $[d, h]$ to E_E ;
      delete v from V ;
    end {if}
  end {while}
  /* west wall */
  let v_1 be an adjacently south-side node linked to v_0 by a west wall edge ;
  put $i \leftarrow 1$;
  while a node v_i is not the lowermost node **do**
    delete two west wall edges $[v_0, v_i]$ and $[v_i, v_{b'}]$ from E_E,
    where $v_{b'}$ is a north-side node of v_i and $v_{b'}$ is a south-side node of v_c ;
    add a west wall edge $[v_{a'}, v_{b'}]$ to E_E ;
    traverse westward through the north wall edges from v_i
      until a perimeter node v_k ;
    let $\sigma_N \leftarrow nw(k)$;
    put $j \leftarrow 1$;
    put $x_j \leftarrow v_e$;
    while $nw(x) < \sigma_N$ **do**

$j \leftarrow j + 1$
let x_j be a southside node linked to x_{j-1} by a west wall edge ;
end {while};
delete
add two west wall edges $[x_{j-1}, v_i]$ and $[v_i, x_j]$ in E_E ;
end {if} ;
delete $[x_{i-1}, s_j]$ from E_E;
let $i \leftarrow i + 1$;
let v_i be a south-side node adjacently linked to v_0 by a west wall edge ;
end {while};
/∗ east wall ∗/
let v'_1 be an adjacently south-side node linked to v_0 by a east wall edge ;
put $i \leftarrow 1$;
while a node v'_i is not the lowermost node **do**
traverse upward through the west wall edges from v'_i
until a perimeter node u'_i ;
delete two west wall edges $[v_0, v'_i]$ and $[v'_i, v_b]$ from E_E,
where v_b is a south-side node of v'_i ;
add an east wall edge $[v_0]$ to E_E ;
traverse westward through the north wall edges from v'_i
until a perimeter node v'_k ;
let $\sigma'_N \leftarrow nw(k')$;
put $j \leftarrow 1$;
put $x'_j \leftarrow v_W$;
while $nw(x'_j) < \sigma'_N$ **do**
$j \leftarrow j + 1$
let x'_j be a southside node linked to x_{j-1} by a east wall edge ;
end {while};
delete $[x_{i-1}, x_j]$ from E_E ;
add two east wall edges $[x'_{j-1}, v'_i]$ and $[v'_i, x'_j]$ in E_E ;
let $i \leftarrow i + 1$;
let v' a south side node adjacently linked to v_0 by an east wall edge ;
end {while};
delete the node v'_j from V_E ;
delete the node v_0 from V_E
end.

Theorem 3.6 *Let D be a tabular diagram, and c be a cell in D. Suppose that E is the tabular diagram obtained from D by the deletion of a column including c, where δ is the width of a perimeter cell of the column including c. Let G_D and G_E be the rectangular dissection graphs for D and E, respectively. Then G_E is obtained from G_D in $O(st)$ time, where s and t are the number of rows and columns in D, respectively.*

Proof. The theorem is verified from Algorithm DeleteColmn.

Q.E.D.

Consider a tabular diagram D of the (s, t)-table. We note here that the relation between the number n of the cells in D and the time complexity of certain algorithms.

The following proposition is verified from Algorithms.

Proposition 3.1 *Let D be a tabular diagram of the (s, t)-table, that is, with s rows and t columns.* ChangeColumnWidth *and* InsertColumn *run in $O(s+t)$ time.*

We obtain the following proposition.

Proposition 3.2 *Let D be a tabular diagram of the (s, t)-table with n cells.* ChangeColumnWidth *and* InsertColumn *run between in $O(\sqrt{n})$ time and $O(n)$ time.*

Proof. The numbers s, t, and n satisfies the following formula.

$$\sqrt{n} \leq \sqrt{st} \leq \frac{s+t}{2} \leq n$$

The second equation is obtained from the formula an arithmetic mean and a geometric mean. Since the cell number n is less than st, the first inequation holds. Since the number of the perimeter cells are greater than $s+t$, the third inequation holds. The time complexity of the algorithms are $O(s+t)$ by Proposition 3.1. We implies that the algorithms run between in $O(\sqrt{n})$ time and $O(n)$ time.

Q.E.D.

For practical computing, the computation time with respect to the number of inner cells may be more important than the computation time with respect to the number of whole cells in the input graph. So, we investigate as following the computation time with respect to the number of inner cells in the input graph.

Definition 3.1 A tabular diagram D is *reduced* if and only if there is no directly linked perimeter nodes in its corresponding rectangular dissection graph, that is, there is no diffuse grid.

The following algorithm reduces the tables.

ALGORITHM ReduceTable(G_D, G_E)
INPUT
G_D : a rectangular dissection graph for a tabular diagram $D = (T, P, g)$, with $s+2$ rows and $t+2$ columns $(s, t > 1)$, and with n' inner cells.

OUTPUT
G_E : a rectangular dissection graph for E, where E is a tabular diagram obtained from D without a diffuse grid

METHOD
begin
Initially, put $G_{E'} \leftarrow G_D$;
Let v_0 be the north west corner node of D;
Let $v \leftarrow v_0$;
while v is inner node **do**
move eastward by one east wall edge;
if the east wall edge of v is directly linked to the southern perimeter node
then delete the east wall edge of v from E;
delete the west wall edge of the node v'
adjacently located at the east side of v ;
let $ew(v) \leftarrow ew(v')$;
let $ew(u) \leftarrow ew(u')$, where u is the node directly linked by east wall edge
from v and u' is the node directly linked by west wall edge from v';
delete v' and u' from E;
end { if }:
end { while };
move to v_0 ;
while v is inner node **do**
move downward by one south wall edge ;
if the south wall edge of v is directly linked to the eastern perimeter node
then delete the south wall edge of v from E;
delete the north wall edge of the node v'
adjacently located at the south side of v ;
let $sw(v) \leftarrow sw(v')$;
let $sw(u) \leftarrow sw(u')$, where u is the node directly linked by south wall edge
from v and u' is the node directly linked by north wall edge from v';
delete v' and u' from E;
end { if }:
end { while };
end.

Proposition 3.3 *Let D be a reduced tabular diagram with $s + 2$ rows and $t + 2$ columns $(s, t > 1)$. Let n' be the number of inner cells in T. Then, $n' \geq max\{s, t\}$.*

Proof. Suppose that $t \geq s$. Let x be the sum of the numbers of vertical edges of north side perimeter nodes except the north east corner and the north west corner. We have $x = 2 \times t$. Let y be the sum of the numbers of northern vertical edges of inner nodes. We have $y = 2 \times n'$. Since $n' < t$, then there exists a northern perimeter node, which is not the north east and the north west corner, the southern edge of which directly linked to the southern perimeter nodes. Thus T is not reduced, a contradiction. Thus the Proposition is verified.

Q.E.D.

Proposition 3.4 *If E is a tabular diagram obtained by application of* UNIFYCELL *or* HSPLITCELL *on a reduced tabular diagram D, then E is reduced.*

Proposition 3.5 *Let D be a reduced tabular diagram with s rows and t columns.* ReduceGraph *and* InsertColumn *run in $O(s+t)$ time.*

Proposition 3.6 *Let D be a reduced square tabular diagram with n inner cells.* UnifyCell*,* HSplitCell*,* InsertColumn *and* DeleteColumn *run between in $O(n)$ time and $O(\sqrt{n})$ time.*

Next, we examine the rectangular dissection graphs extended from Definition 2.3, that is, the vertices of those graphs have location attributes. Similarly to the above algorithms, we can construct the following algorithms, i. e., (1) UnifyCells2 for the unifying of cells, (2) SplitCell2 for the splitting of a cell, (3) InsertColumn2 for the insertion of a column and (4) DeleteColumn2 for the deletion of a column. Suppose that T is a rectangular dissection graph and the vertices of T have the location attributes and s and t are the number of rows and columns in D. Then we have,

Proposition 3.7 UnifyCells2 runs in $O(1)$ and SplitCell runs in $O(s)$. InsertColumn2 runs in $O(st)$ and DeleteColumn2 runs in $O(st)$, since the changing of the width of a cell requires $O(st)$ time.

4. CONCLUSION

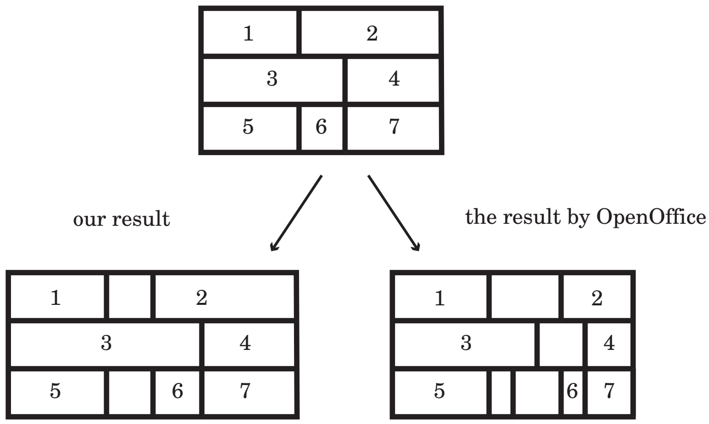
The column insertion algorithm runs on $O(\sqrt{n})$ time for $n = \sqrt{n} \times \sqrt{n}$ cell square diagrams, while known methods require $O(n)$ time. The following table illustrates the features of representation methods for n cell square tables with respect to the column insertion.

Model	Node degrees	Cell to node relation	Cell visits	Complexity
Quadtrees	at most 5	one 'block' to one node	at most n	$O(n)$
Rectangular dual graphs	at most $4n$	one cell to one node	at most n	$O(n)$
Rectangular dissection graphs	at most 8	one cell to one node	at most $2\sqrt{n}$	$O(\sqrt{n})$

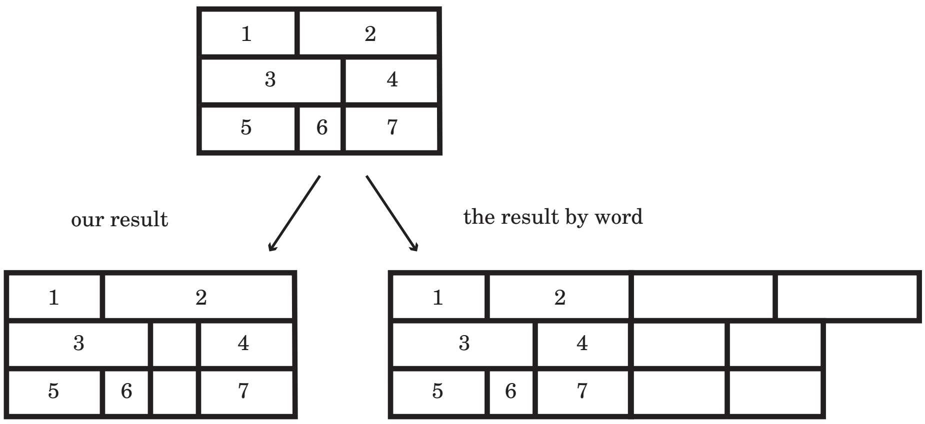
We introduced attribute graphs and algorithms for table editing. The necessary and sufficient condition, where an attribute graph represents a tabular diagram, has been determined by a graph grammar [Kirishima, Arita, Motohashi, Tsuchida and Yaku] . Future research focuses on designing a processing system for table editing based on other reasearch [Kirishima, Motohashi, Tsuchida and Yaku 2002].

APPENDIX

insert a column to the left side of the cell 2



insert a column to the right side of the cell 3



Acknowledgement

The authors thank to Prof. Kazuhito Tominaga of Tokyo University of Technology for valuable discussion with him.

REFERENCES

K. Kozminski, E. Kinnen, Rectangular Duals of Planar Graphs, *Networks 15* (1985) 145-157.

Franz J. Brandenburg, Designing Graph Drawings by Layout Graph Grammers, *Proc. Graph Drawing '94, LNCS 894* (1994) 416-427.

M. de Berg, M.van Kreveld, M. Overmats and O. Schwarzkoph , Computational Geometry - Algorithms and Applications, *Springer* (1997)

Sukhamay Kundu, The Equivalence of the Subregion Representation and the Wall Representation for a Certain Class of Rectangular Dissections, *Communications of the ACM 31* (1998), 752-763.

T. Arita, K. Tsuchida, T. Yaku et al. , Syntactic proccessing of diagrams by graph grammers, *Proc. IFIP World Computer Congress ICS 2000* (2000) 145-151.

A. Amano, N. Asada, T. Motoyama, T. Sumiyoshi and K. Suzuki, Table Form Document Systhesis by Grammar-Based Structure Analysis, *6-th Internal Conference on Document Analysis and Recognition (ICDAR)* (2001) 533-537

T. Motohashi, K. Tsuchida and T. Yaku, Attribute Graphs and Their Algorithms for Table Interface, *TECHNICAL REPORT OF IEICE SS2002-1* (2002).

T. Motohashi, K. Tsuchida and T. Yaku, Proc. Attribute Graphs for Tables and Their Algorithms, *Proc. Foundation of Software Engineering 2002* (K. Inoue Ed.), Kindaikagakusha, Tokyo, (2002) 183-186.

T. Kirishima, T. Motohashi, K. Tsuchida and T. Yaku, Attribute Graph for Table and Their Applications, *Proc. IASTED International Conference on Software Engineering and Applications SEA 2002* (2002), 317 - 322.

T. Kirishima, T. Arita, T. Motohashi, K. Tsuchida and T. Yaku, Syntax for Tables, *Proc. IASTED International Conference on Applied Informatics AI 2003* (2003), 1185 - 1190.

Received Month Year; revised Month Year; accepted Month Year