Translation from Transactions of IPSJ

Generation of the Hichart Program Diagrams

KOICHI OGURA*, NOBUYOSHI GO**, MIKI KISHIMOTO***
YOUZOU MIYADERA****, NAOYUKI OKADA*****, KENSEI TSUCHIDA******
HIROSHI UNNO**** and TAKEO YAKU****

This paper concerns methods of generating tree-structured program diagrams.

The first part introduces an implementation of a system for generating **Hichart** [1, 7] diagrams. The drawing of the diagrams is then discussed. Several eumorphous conditions are formalized for tree-structured diagrams that consist of non-uniform sized cells. Some are conversions of conditions for trees [2-4], and some are extensions of recent results [12]. It is shown that some eumorphous conditions correspond to $O(n)$ and $O(n^4)$ time complexity algorithms and to an NP-hard problem, analogously to conditions for trees [6]. Finally, several methods are introduced that provide diagrams satisfying eumorphous conditions introduced here.

It is noted that this system for generating **Hichart** program diagrams is the first that employs formalized eumorphous conditions and corresponding formalized methods, and that the results given in this paper can be applied to all tree-structured diagrams, including **PAD**, **SPD**, and **TSF** program diagrams.

1. Introduction

Since the 1970's, a number of program diagram languages have been proposed as tools for visual representation of programs, in order to reduce the complexity of program development and analysis. They include **NSD, Hichart, HCP, SPD, TSF, PAD, YAC II,** and **50SM** [9, 10]. A number of processing systems have been also implemented for these program diagram languages [8], namely, diagram generators, diagram editors, diagram compilers, and related systems. We will deal in this paper with methods and theory related to these processing systems for program diagram languages.

An important problem in the design of such processing systems is how to draw program diagrams tidily and efficiently. So far, however, there has been no systematic study of this drawing problem.

We will consider **Hichart** program diagrams, which are sets of *tree-flowcharts* [1]. The tree-flowchart introduced the iteration symbol  for the first time. Recently, this symbol has been commonly used in several program diagrams languages. The major characteristics of **Hichart** are (1) a tree-flowchart keeps the flow control lines of a Neumann program flowchart, (2) only two classes of symbols are added to the Neumann flowcharts to represent control structures, one for iteration and the other for selection, and (3) a program is represented by a tree-like structured graph, which is spread rightward by the use of these two classes of control symbols. Therefore, **Hichart** is different from many other diagram languages in the sense that it keeps the characteristics of a flow chart, and is able to describe simultaneously both the hierarchy and the flow of the control. The tree-flowchart discussed here is a diagram in which each rectangle (cell) is placed in a tree-like structure on the integral lattice. The tree-flowchart can be used to represent not only a control flow but also the modular structure and data structure of a program.

Our diagram-generating subsystem was implemented as a part of a system that generates program diagrams from **Pascal** source programs. The details of its development steps are as follows.

(1) First, we dealt with a generating system that manipulates a program diagram, which consists only of fixed-size (1×1) rectangles ("cells").

(2) Next, we developed a method for diagram layout optimization that employs the results of tree layout optimization problems [6]. We also developed a module that determines the size of cell symbols according to lexical analysis, and implemented a system for generating the diagrams, which consists of cells with variable heights and fixed width.

(3) Then, we extended the generating system to enable it to manipulate program diagrams in which

This is a translation of the paper that appeared originally in Japanese in Transaction of IPSJ, Vol. 31, No. 10 (1990), pp. 1463-1473.

*Academic Computing Center, Hokkaido Tokai University, Sapporo, Hokkaido, Japan.
**Academic Computing Center, Tokai University, Hiratsuka, Kanagawa 259-12, Japan.
***Fujitsu Ltd., Kawasaki, Kanagawa, Japan.
****Department of Information Sciences, Tokyo Denki University, Hatoyama, Saitama 350-03, Japan.
*****NEC Corp., Fuchu, Tokyo, Japan.
******Department of Industrial Engineering and Management, Kanagawa University, Yokohama, Kanagawa, Japan.

both the heights and widths of cells are variable [5].

The problem of drawing trees tidily is known to be similar to our problem of drawing tree-flowcharts tidily. The former problem has been studied frequently from the point of view of graph algorithm theory. Wetherell and Shannon formalized constraints for the tidy drawing of binary trees and proposed a linear time algorithm for the drawing [2]. Reingold and Tilford presented a linear time algorithm that gives narrower drawings of binary trees while satisfying the Wetherell-Shannon constraints [3]. Supowit and Reingold proved that the problem of determining the narrowest drawing of binary trees under the Wetherell-Shannon constraints has polynomial time complexity in the case of real number coordinates, because the problem can be converted to one of linear programming, while the problem is NP-complete in the case of integer coordinates [4].

Tsuchida introduced constraints for general n-ary trees after being motivated by the problem of drawing **Hichart** program diagrams. Furthermore, he showed that the problem of drawing the narrowest n-ary trees has a computational hierarchy such as $O(n)$, $O(n^2)$, $O(n^3)$, $O(n^4)$, and NP-hard, according to the sets of constraints [6].

Go and others modified the above constraints in n-ary tree drawings, and obtained the constraints in program diagram drawing [12]. The problem of drawing program diagrams has been also studied from the point of view of attribute graph grammars. Nishino constructed an attribute graph grammar corresponding to a set of constraints, and obtained several properties of the problem [11].

The motivation of our study in this paper is to implement a diagram processing system that employs a systematic theory including diagram drawing problems.

The methods newly obtained in this paper will be applied not only to **Hichart** and *PAD* diagrams but also to **SPD** and **YAC II** diagrams and to all the generating systems for displaying tree-structured data.

As a first step, we deal in this paper with the following two subjects. First, we introduce a method of implementing the generating system. Second, we formalize diagram drawing problems in a mathematical theory, and clarify the mathematical properties of these problems.

In Section 2, we will introduce the data structure and the procedure structure of a **Hichart** diagram generating system. We will attempt to represent a program diagram by means of a hierarchical graph. Accordingly, we can smoothly employ a graph theoretical approach in our system. In consequence, it is easy to improve our system and to add new functions to it.

In Section 3, we will modify the constraints [2, 4, 6] and introduce several constraints for tree-structured diagrams. First, we will deal with several constraints for tree-structured diagrams. The constraints for n-ary trees [6], which have been studied from the viewpoint of graph algorithm theory, are directly applied to program diagrams consisting only of uniform-sized cells. We therefore extend the constraints corresponding to tree-structured diagrams that consist of non-uniform-sized cells. Furthermore, the constraint in the former conditions [4, 6], in which every parent cell is centered over its children, can be applied only to a special situation of display (such as a fair copy) for a minority of program diagrams, such as **PAD** and **Hichart** diagrams. Therefore, there may exist some better sets of constraints for program diagrams in which a constraint causes the parent cell to be placed $k (k \geq 0)$ levels below its top child. We formalize this constraint. The resultant set of constraints is effective in displaying **SPD**, **YAC II** diagrams, and other diagrams, in addition to **PAD** and **Hichart** diagrams. It is known that for n-ary trees, the sizes of horizontal intersections between the subtrees affects the time complexity of a layout algorithm [6]. We will there-fore introduce two alternative constraints for tree-structured diagrams. One, $E'_*(j)$, permits intersection and the others, $E_0(j)$, does not [12]. The horizontal length of the interior cells is fixed at 1 in conditions $E_0(j)$ and $E'_*(j)$, in consideration of the application to program diagrams. We will formalize, in Section 3, conditions $F_0(j)$ and $F'_*(j)$ without this constraint.

In Section 4, we will deal with methods and their complexities corresponding to a set of the above constraints. We consider two methods corresponding to the conditions $F_0(j)$ and $F'_*(j)$. These methods are modified from the former algorithms [6] by the extension and modification of the constraints described in Section 3. It is noted that we obtain the relations among the constraints and the time complexity in the problems of drawing tree-structured diagrams.

2. System Structure of a System for Generating Program Diagrams

In this section, we describe the internal representation of program diagrams, which are used in our system for processing program diagrams, and the modular structure of the processing system. In subsequent sections, when we consider drawing problems, we assume the internal representation and the processing system described here.

To begin with, we explain the language specification of **Hichart**. A **Hichart** program diagram is a graph composed of control symbols, as shown in Fig. 1. A **Hichart** program diagram is also called a *tree flowchart*, and is illustrated in Figs. 2 and 3. Figure 2 shows an example of a **Hichart** program diagram. Figure 3 shows an explanatory diagram, which is a Neumann flowchart corresponding to Fig. 2. Next, we describe the internal representation in the processing system. In the **Hichart** processing system, **Hichart** program diagrams are represented by using a common internal representation, and are treated uniformly. This internal representation

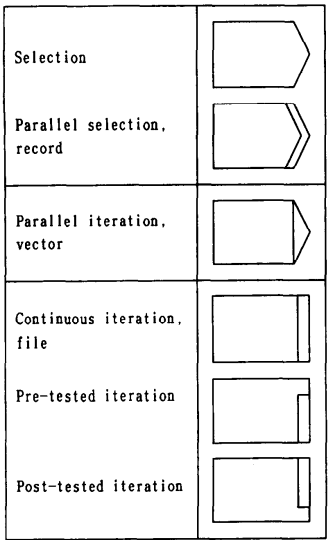

Fig. 1 **Hichart** control symbols.

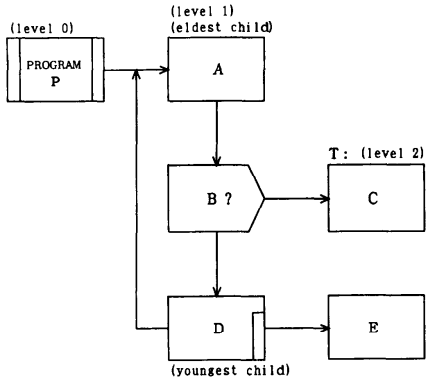

Fig. 2 Example of a **Hichart** program diagram.

is called H-*code*. Information on a cell in the H-code is represented by one record. A graph is usually expressed by vertices and edges, but information on a cell in the H-code also includes information on how it can be linked to other cells. Figure 4 shows the structure of the internal H-code representation. Figure 5 shows a **Hichart** program diagram represented in H-code. ETA_AIDE [9] is a software development environment in which programs written in **Hichart** can be manipulated. It consists of a system that manages specifications, program diagrams, source programs, and relational databases, and a filter between those objects. The core part of ETA_AIDE is Eta/H*, which is the processing system for the **Hichart** program diagrams. It consists of three tools: Eta/HED (a **Hichart** flowchart EDitor), Eta/HfromP (a Hichart-from-Pascal translator), and Eta/HtoP (a Hichart-to-Pascal translator). The struc-

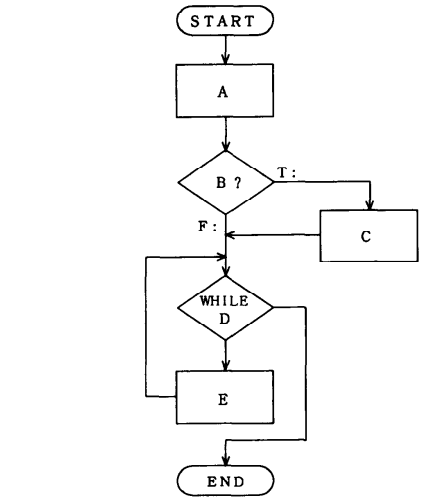

Fig. 3 Neumann flowchart corresponding to Fig. 2.

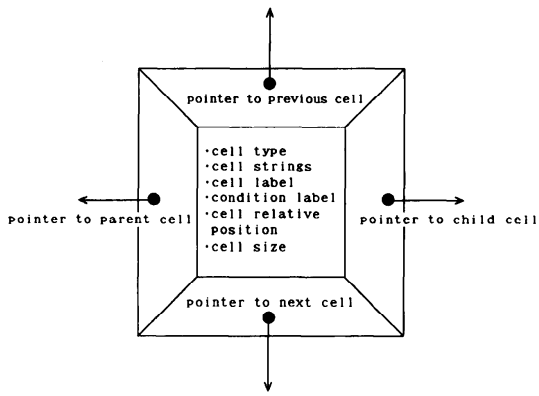

Fig. 4 Structure of H-code for internal representation.

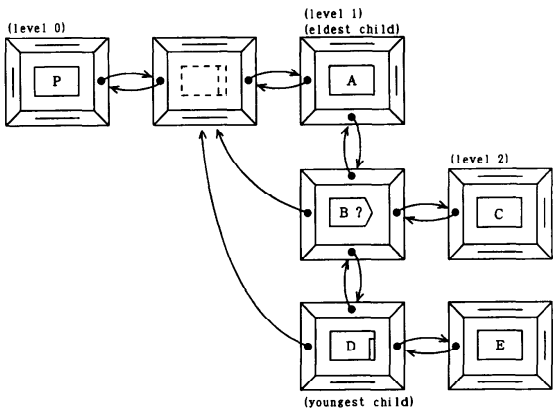

Fig. 5 Example of a **Hichart** program diagram represented by H-code for internal representation (an outline).

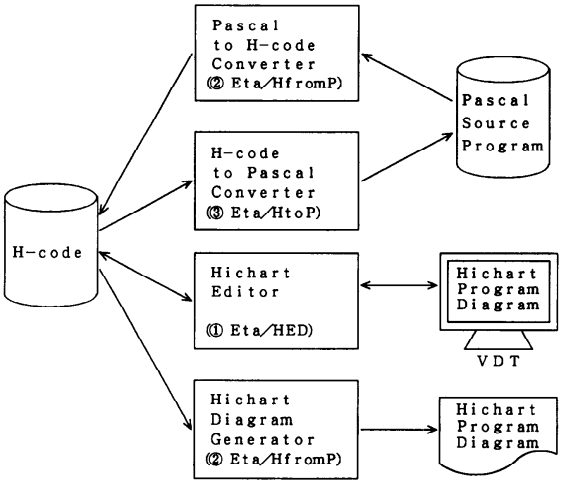

Fig. 6 Structure of the processing system of the **Hichart** program diagram.

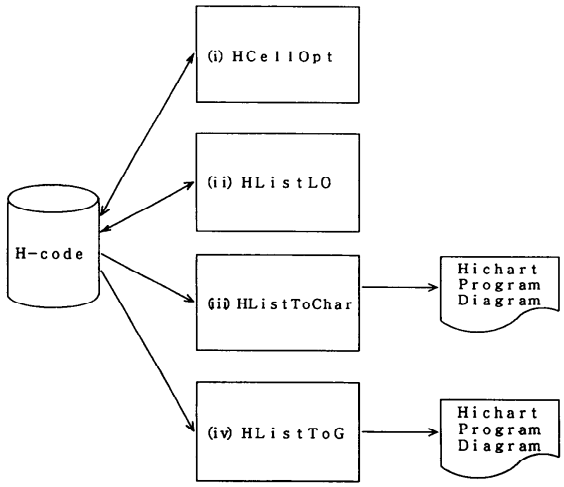

Fig. 7 System flowchart of the **Hichart** Diagram Generator.

ture of Eta/H* is illustrated in Fig. 6.

The **Hichart** Diagram Generator and drawing problems are related to the HED and the HfromP. We now describe the function and structure of the **Hichart** Diagram Generator. The **Hichart** Diagram Generator consists of the following four modules:

(i) HCellOpt (an H-code Cell size Optimizer). This adjusts the length of character strings written inside each cell, decides each cell size, and stores this information in H-code lists (see Fig. 4) (about 500 Pascal steps).

(ii) HListLO (an H-code List Layout Optimizer). This determines the placement of each cell in drawing a diagram, and stores this information in H-code lists (about 1000 Pascal steps).

(iii) HListToChar (an H-code List-to-Character format converter). This puts out the **Hichart** diagrams on printing devices from generated H-code lists (about 2000 Pascal steps).

(iv) HListToG (an H-code List-to-Graphics). It puts out the **Hichart** diagrams on graphic devices from H-code lists (about 2000 Pascal steps).

The system flowchart of the **Hichart** Diagram Generator is shown in Fig. 7. The method described in Sections 3 and 4 is used in HListLO.

3. Eumorphous Conditions for Program Diagrams

In this section, we consider tree-structured program diagrams of "cells" with variable sizes, and deal with "eumorphous" conditions concerning the drawing of the diagrams. Drawing methods for trees can be applied to the drawing of a tree-structured program diagrams of uniformly sized cells. First, we will define the following terms in order to apply the drawing problems of trees to the problems of drawing tree-structured program diagrams.

Definition 1. A tree structure is $T=(V, E, r, width, depth)$, where (V, E) is an *ordered tree* (a *tree* denotes an ordered tree), V is a set of cells, and E is a set of *edges*. $r \in V$ is the *root* cell. The map *width* $V \to Z$ is the *width function* of the cells. $width(p)$ represents the vertical length, which is called the *width*, of the cell p. The map *depth* $V \to Z$ is the *depth function* of the cells. $depth(p)$ represents the horizontal length, which is called the *depth*, of the cell p.

Definition 2. A *placement* of a tree structure T is defined by the following function π:

π: the *cells* $\to Z \times Z$. $\pi_x(p)$ and $\pi_y(p)$ denote the *x-coordinate* and the *y-coordinate* of p, respectively, where $\pi(p)=(x, y)$. We assume that the x-coordinate is directed from left to right and that the y-coordinate is directed downward, because of the application to program diagrams.

A pair (T, π) is called a *tree-structured diagram*. A tree structure is a *rooted tree* in which each tree is assigned two *attributes*, $width(p)$ and $depth(p)$. A tree-structured diagram is a rooted tree with vertices defined by four values, $width(p)$, $depth(p)$, $\pi_x(p)$, and $\pi_y(p)$. Each rectangle is represented by a cell, and a tree-structured diagram represents a diagram of rectangles placed in a tree structure on the integral lattice. In this paper, we call the length of a tree-structured diagram along the x-axis its *depth* (in the hierarchical order), and the length of a tree-structured diagram along the y-axis its *width*, in view of the application to program diagrams. In our discussion of ordered trees (ordered tree-structured diagrams), brothers (cells) are numbered in downward order. We define the location of a cell as follows. The *unit length* of x-coordinates (y-coordinates) is defined as the length of the vertical (horizontal) length of the unit cell (size 1×1). If the *location* of a cell p is (x, y), then we can put x unit cells between the cell p

and the y-coordinate, and y unit cells between the cell p and the x-coordinate. The *width* of a tree-structured diagram (T, π) is defined by:

$width(T, \pi) = \text{def} = \max\{\pi_y(p) + width(p) - \pi_y(q) - 1 \mid p \text{ and } q \text{ are cells in } T \text{ and } \pi_y(p) \geqq \pi_y(q)\}$. The *level* of a cell p is defined by the number of edges between the cell p and the root cell. For a cell p, the function *Index* is defined by

$Index(p) = \text{def} = 0$: if p is the root cell

i: if p is the *i-th* child of the parent of p

We introduce constraints concerning a placement of a tree structure. The following constraints B1, B2, B3, B4, B5, B5(j) and B¥(k) are introduced by Go. et al. [12].

Constraint B1 [2] (the lines among cells are not crossing). For a tree-structured diagram (T, π), if the level of a cell p is equal to the level of a cell q and $\pi_y(p) < \pi_y(q)$, then

$$\pi_y(\textit{the oldest child of } q) > \pi_y(\textit{the youngest child of } p) + width(\textit{the youngest child of } p)$$

Constraint B2 (the cells are mutually disjoint). For a tree-structured diagram (T, π) and a cell p, let $area(p, \pi) = \text{def} = \{(x, y) \mid \pi_x(p) \leq x \leq \pi_x(p) + depth(p) - 1,\ \pi_y(p) \leq y \leq \pi_y(p) + width(p) - 1\}$. Then,

$$are(p, \pi) \cap area(q, \pi) = \phi \text{ for all } p \text{ and } q (p \neq q).$$

Constraint B3 [4]. For a tree-structured diagram (T, π), if $T1$ and $T2$ are topologically isomorphic *sub*-tree structures, then $T1$ and $T2$ are placed in the same form with respect to a parallel movement.

The following **Constraint B4** is common in eumorphous conditions [2] of a tree. For a tree-structured diagram, the constraint is introduced in order to make the logically hierarchical level and the geometrically hierarchical level of a cell coincide [12]. The constraint is specific to tree-structured diagrams.

Constraint B4 [12]. In a tree-structured diagram (T, π), for any cell p with the level i, $\pi_x(p) = i$.

The following constraint is an extension of **Constraint B4.**

Constraint B4'. Let a cell p be in a tree-structured diagram (T, π), and let $p_0, p_1, \ldots, p_m$ be the cells on the path from the root cell $r = p_0$ of T to $p_m = p$. Then $\pi_x(p) = \Sigma_{(0 \leq i \leq m-1)} width(p_i)$.

The following constraint is common for binary trees [2]. This constraint is efficient for the attractive printing of several kinds of program diagram languages, including **PAD** and **Hichart** diagrams.

Constraint B5. For a tree-structured diagram (T, π), if a cell p has k children $q_1, \ldots, q_k$ ($Index(q_i) = i$, $1 \leq i \leq k$), the $\pi_y(p)$ satisfies the following condition:

$$\pi_y(p) + [width(p)/2] = \pi_y(q_1) + [g/2],$$

where $g = \pi_y(q_k) + width(q_k) - \pi_y(q_1) - 1$.

Constraint B5 is not generally suitable for displaying program diagrams. That is, if the diagram is displayed downward in a VDT, a child module may be displayed earlier than its parent module on the VDT. We therefore introduce the following constraint instead of **Constraint B5**.

Constraint B5(j). In a tree-structured diagram (T, π), if a cell p has k children $q_1, \ldots, q_k$ ($Index(q_i) = i$, $1 \leq i \leq k$), then $\pi_y(p) = \pi_y(q_1) + min\{j, \pi_y(q_k) - \pi_y(q_1)\}$

A function *Intersect* of the set of tree-structured diagrams to the integers is defined as follows:

$Intersect(T, \pi) = \text{def} = max\{\pi_y(p) - \pi_y(q) + 1$, where $T1$ and $T2$ are arbitrary sub-tree-structures for which the root cells are brother and $Index$(the root of $T2$) > $Index$(the root of $T1$). p and q are arbitrary cells in $T1$ and $T2$, respectively.$\}$

The next constraint concerns intersections between tree structures. It is known that the time complexities of the drawing problems for trees depend on the values of these intersections [6].

Constraint B¥(k). For $k \geq 0$, a placement π satisfies $Intersect(T, \pi) \leq k$.

Next, we combine the above constraints and introduce several "eumorphous conditions" for a tree-structured diagram. The first two conditions below are the same as the corresponding eumorphous conditions [2, 6] for trees, in which the constraint corresponding to **Constraint B5** in a tree is not changed. These conditions can be used to produce attractive drawings of program diagrams such as **PAD** and **Hichart**.

Notation 1 [12]. The eumorphous conditions E_0 and E'_* for a tree-structured diagram are combinations of the above constraints:

$$E_0 = B1 \wedge B2 \wedge B3 \wedge B4 \wedge B5 \wedge B¥ \text{ (0)}$$

$$E'_* = B1 \wedge B2 \wedge B4 \wedge B5$$

The next conditions are modifications of the above conditions obtained by replacing $B5$ with $B5(j)$ for more general drawing of tree-structured diagrams.

Notation 2 [11]. The eumorphous conditions $E_0(j)$ and $E'_*(j)$ denote the combinations

$$E_0(j) = B1 \wedge B2 \wedge B3 \wedge B4 \wedge B5(j) \wedge B¥ \text{ (0)}$$

$$E_*(j) = B1 \wedge B2 \wedge B4 \wedge B5(j)$$

The next conditions are modifications of the above conditions obtained by replacing $B4$ with $B4'$ for general drawing of tree-structured diagrams.

Notation 3. The eumorphous conditions F_0, F'_*, $F_0(j)$, and $F'_*(j)$ denote the combinations

$$F_0 = B1 \wedge B2 \wedge B3 \wedge B4' \wedge B5 \wedge B¥ \text{ (0)}$$

$$F'_* = B1 \wedge B2 \wedge B4' \wedge B5$$

$$F_0(j) = B1 \wedge B2 \wedge B3 \wedge B4' \wedge B5(j) \wedge B¥ \text{ (0)}$$

$$F'_*(j) = B1 \wedge B2 \wedge B4' \wedge B5(j)$$

These conditions are applicable not only to **PAD** and **Hichart** diagrams that are drawn in a fan style, but also to a large number of diagrams in which the hierarchical

level and the x-coordinate of the cell coincide, including **SPD** and **TSF** diagrams. Furthermore, they are partly applicable to the drawing of **HCP** and other diagrams. The **Hichart** program diagram shown in the following figure satisfies the eumorphous condition E'_*.

4. Drawing Methods

In this section, we consider informal methods of obtaining placements under the eumorphous conditions introduced in the last section. In problems of drawing trees tidily the drawing problem is NP-hard if the interval between the children of each vertex is uniform [6]. Accordingly, the following Theorem 1 holds for tree-structured diagrams.

Constraint $B\#$. For a tree-structured diagram (T, π), if a cell p has $k (k \geq 3)$ children $q_1, \ldots, q_k$ ($Index(q_i) = i$, $1 \leq i \leq k$), then $\pi_y(q_{j+2}) - \pi_y(q_{j+1}) = \pi_y(q_{j+1}) - \pi_y(q_j)$ $(1 \leq j \leq k-2)$

Let $E\# = B1 \wedge B2 \wedge B3 \wedge B4' \wedge B5 \wedge B\#$. Then we have

Theorem 1. *In a tree-structured diagram (T, π), it is NP-hard to determine whether the placement π satisfies the constraint $E_\#$ and has the narrowest width.*

For trees, the complexity is known when the following constraints are added. Cousequently, we have similar results for tree-structured diagrams. The prototype of the following condition was introduced to allow easy development of algorithms [6].

Constraint B$. For a tree-structured diagram (T, π), for all sub-tree structures T_i and T_j with the root cells r_i and r_j and with $Index(r_i) + 1 = Index(r_j)$, the maximum value of π_y (the lowest cell of T_i) + *width* (the lowest cell of T_i) $< \pi_y(r_j) + [width(r_j)/2]$, and

$$\pi_y(r_i) + [width(r_i)/2] < \pi_y \text{ (the highest cell of } T_j)$$

Notation 4 [6]. The eumorphous conditions $E_*\$$ and $F_*\$$ are defined by

$$E_*\$ = B1 \wedge B2 \wedge B3 \wedge B4 \wedge B5(j) \wedge B\$$$

$$F_*\$ = B1 \wedge B2 \wedge B3 \wedge B4' \wedge B5(j) \wedge B\$$$

Theorem 2 [6]. *Suppose that any cell v in a tree structure T satisfies the equation width(v)=*depth$(v)=1$. *Then, there is on $O(n^4)$ time algorithm that provides the narrowest placement satisfying the eumorphous constraint $E_*\$$.*

Next, we consider drawing methods, concerning the conditions $F_0(j)$ and $F'_*(j)$, $(j \geq 0)$, for the tree structures of cells with variable sizes. Making the narrowest placement π_{init} for a tree structure T under the eumorphous condition $F_0(j)$, $(j \geq 0)$ is called *the initialization* of the placement. The placement π_{init} is called the *initial* placement. The initialization procedure of the placement is easily constructed as follows:

Procedure *Init*: initialization of the placement

[Calling sequence]*Init* (T, j, π_{init})

[Input]$T = (V, E, r,$ width, depth$), j$

[Output]π_{init}: the narrowest width placement for T satisfying $F_0(j)$

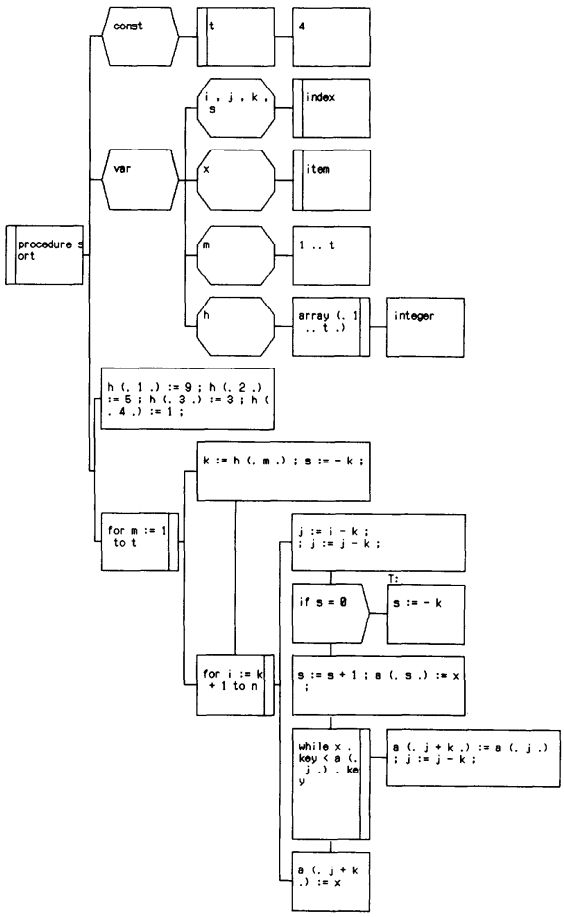

Fig. 8 Example of a Hichart program diagram satisfying condition E'_*.

[Method]place each cell in post order that satisfies the constraints $B1$, $B2$, $B3$, $B4'$, $B5(j)$.

Lemma 1. *π_{init} obtained by the procedure Init is the narrowest placement that satisfies $F_0(j)$. Init runs in $O(n)$ time.*

Proof Outline. The constraints $B1$, $B2$, $B4'$, and $B5(j)$ are obviously satisfied. $B3$ is also satisfied, because the placement works in post order. The width is clearly at a minimum.

A *placement removal* means changing the placement π_{init} that satisfies the eumorphous condition $F_0(j)$ $(j \geq 0)$ to a narrower placement that satisfies the condition $F'_*(j)$, $(j \geq 0)$. The following procedure *Layout* for a placement removal is a combination of the above *Init* and the next *subOpt*. Another procedure [6] is known for conditions F_0 and $F_*\$$, when the sizes of the cells are uniform. A eumorphous placement of a sub-tree structure T_q with the root cell q is determined as follows:

Procedure *SubOpt*: an upward movement of a subtree structure

[Calling sequence] *SubOpt*(T, j, p, π_{in}, π_{out}, m)
[input]$T=(V, E, r, width, depth)$: a tree structure
p: a cell of T
we assume that T_p is a sub-tree structure with a root p in T
π_{in}: a placement of T
j: j in $F'_*(j)$
m: a mark on the cells in T
[Output]π_{out}: A placement of T_p, the removed placement
[Method (Outline)]

```
begin
(*This procedure runs in post order and *)
(* determines the locations of the cells in *)
(* post order. *)
Initially π_out:=π_in;
Let p_1, . . . , p_k be the children of p;
for i=1 to k do
  begin
    if p_i is an unmarked leaf then
      begin
      move p_i upward satisfying
      B1, B2, B4', and B5(j) with respect
      to p;
      Mark "moved" p_i;
      π_in:=π_out
    end;
  if p_i is an unmarked interior cell then
    begin
      SubOpt(T, j, p, π_in, π_out, m);
      Mark p_i "moved";
      π_in :=π_out
    end
  end;
Evaluate the maximum distance d of the upward
movement of p satisfying B1, B2, B4', and
B5(j) with respect to T_p and P, where P is
the set of cells located above T_p;
Shift upward all cells in T_p by d;
π_in:=π_out
end.
```

Lemma 2. *The placement π_{out} obtained by the procedure SubOpt for T_p satisfies $F'_*(j)$, and the width of π_{out} is narrower than or equal to the width of the placement π_{in}.*

Proof Outline. Constraints $B1$, $B2$, $B4'$, and $B5(j)$ are clearly satisfied.

Accordingly, a eumorphous placement of a tree structure T is obtained by the following procedure *Layout*:

Procedure *Layout*: Removal of a placement of a tree structure.
[Calling sequence] *Layout*(T, j, π, *width_T_π*)
[Input] $T=(V, E, r,$ width, depth): a tree structure
j: j in $F'_*(j)$
[Output] π: A eumorphous placement of T satisfying $F'_*(j)$
width_T_π: *width*(T, π)
[Method]

```
begin
  Init(T, j, π_init);
  Initially, all cells in T are unmarked;
  SubOpt(T, j, r, π_init, π, Mark)
end.
```

Proposition 1. *The procedure LayOut provides a placement π with a width narrower than or equal to π_{init}, which satisfies the condition $F'_*(j)$.*

Proof Outline. We can verify that the procedure satisfies the constraints as follows. $B1$, $B2$, $B4'$, and $B5(j)$ are clearly satisfied.

The procedure *LayOut* runs in $O(n^3)$ time. For the condition F_*, we can consider similar procedures to that for a tree.

5. Conclusions

In our system for generating program diagrams, graph algorithm theory can be easily applied to tasks such as drawing problems, since program diagrams can be represented by hierarchical list structures, as shown in Section 2. This style of representing the inner structure of generators of structured program diagrams is effective for representing general structured diagrams in list structures. Such diagrams include **NSD** diagrams of modular structures in addition to hierarchical tree-structured program diagrams.

Traditional eumorphous conditions for n-ary trees are effective for tree-structured diagrams with uniformly sized cells. We formalized in Section 3 the eumorphous conditions for tree-structured diagrams with cells of variable sizes. We also introduced a new constraint, the locations of parent cells. The new eumorphous constraint is applicable not only to **Hichart** and **PAD** diagrams, in which the root cell is located in the center of the left-hand side, but also to **SPD**, **YAC II**, and **TSF** diagrams, in which the root cell is located at the top of the left-hand side.

In Section 4, we considered a drawing method corresponding to the eumorphous conditions introduced in Section 3. Our eumorphous conditions were compared with the eumorphous conditions of trees, and were shown to have time complexities that are $O(n)$, $O(n^4)$, and NP-hard according to the level of eumorphousness. In the development of practical programs, the $O(n)$ condition is appropriate for use in the middle stages of development, and the $O(n^4)$ condition is suitable for use in the documentation stage after development. The NP-hard condition seems to be the most eumorphous, but it is not feasible.

Finally, we introduced a method called *LayOut* corresponding to our eumorphous constraint, as the main result of this study.

We implemented drawing modules with procedures corresponding to the conditions F_0, F'_*, $F_0(j)$, $F'_*(j)$, and are using these modules in programming education and program development.

Our results are partially applicable to **HCP** diagrams, in which equally leveled cells may be placed from left to right, but not to the **NSD** and **Chapin** charts, which are formed in modular structures.

In the future, it is necessary to consider variations of the eumorphous conditions introduced in Section 3. Furthermore, it is necessary to investigate whether the *LayOut* method provides the placement with the narrowest width. If not, then it will be removed. In practical drawing modules, the width and the depth of a diagram are limited by page margins. As a result, diagrams have to be drawn separately in page margins. Accordingly, it is necessary to integrate the layout module and decision modules for the sizes of the cells. Accordingly, it is necessary to consider a module that works interactively between determination of the sizes of the cells and drawing modules introduced here, and to evaluate those complexities.

The *LayOut* method is shown to provide a diagram satisfying the eumorphous condition. It is an open question, however, whether the output diagrams of LayOut have the narrowest width.

Acknowledgement

This diagram generating system has been developed over a long period. The authors thank Mrs. Michiko Nakanishi of Tokai University, Mr. Yoshiichi Osada of Tokai University, Mr. Kazuhiro Takeuchi of Tokai University (presently in NEC Corp.), Mr. Masahiko Kondo of Waseda University (presently in IBM Japan Corp.), Mr. Hiroshi Banba of Tokai University (presently in Hokkaido Tokai University), and Mr. Kazuaki Imai of Tokai University (presently in CSK Corp.), for their assistance in the development of the system. They are also indebted to Dr. Kokichi Futatsugi of Electrotechnical Lab., Prof. Koushi Anzai of Kanto Gakuen University, Prof. Kimio Sugita of Tokai University, Prof. Esturo Moriya of Tokyo Women's Christian University, and Dr. Tetsuro Nishino of Tokyo Denki University (presently in Japan Advanced Institute of Science and Technology, Hokuriku).

The seventh and eighth authors were wartly supported by a Tokyo Denki University Research Institute Grant and by a grant from Galois Corp.

References

1. YAKU, T. and FUTATSUGI, K. Tree-Structured Flowcharts, Memoir of IECE, J. AL-78 (in Japanese with English abstract) (1978), 61-66.
2. WETHERELL, C. and SHANNON, A. Tidy Drawings of Trees, *IEEE Trans.*, **SE-5** (1979), 514-520.
3. REINGOLD, E. M. and TILFORD, J. S. Tidier Drawings of Trees, *IEEE Trans.*, **SE-7** (1981), 223-228.
4. SUPOWIT, K. J. and REINGOLD, E. M. The Complexity of Drawing Trees Nicely, *Acta Inf.*, **18** (1983), 377-392.
5. MIYADERA, Y., IMAI, K., KUWABARA, H., UNNO, H. and YAKU, T. ETA87-An Extension of a Hichart Flowchart Processing System, *Proc. 35th Annual Convention IPS Japan* (1987), 1201-1202.
6. TSUCHIDA, K. The Complexity of Tidy Drawings of Trees, *Topology and Computer Science* (S. Suzuki ed.), Kinokuniya, Tokyo (1987), 487-520.
7. YAKU, T., FUTATSUGI, K., ADACHI, A. and MORIYA, E. HICHART-A Hierarchical Flowchart Description Language-, *Proc. IEEE COMPSAC*, **11** (1987), 157-163.
8. SUGAI, M., UCHIYAMA, A., HAGIWARA, N., REKIMOTO, J., KOYAMADA, M. and SHIGO, O. A Coherent Software Development System with Interface Dictionaries, *Proc. IEEE COMPSAC*, **11** (1987), 428-432.
9. HARADA, K. (ed.), *Structured Editors*, Kyoritsu Shuppan (in Japanese, 1987), Tokyo.
10. TRIPP, L. L. A Survey of Graphical Notations for Program Design-An Update, ACM SIGSOFT SOFTWARE ENG. NOTES, **13** (1988), 39-44.
11. NISHINO, T. Attribute Graph Grammars with Applications to Hichart Editors, *Advances in Software Science and Technology*, **1** (1989), 426-433.
12. GO, N., KISHIMOTO, M., MIYADERA, Y., OKADA, N., TSUCHIDA, K. and YAKU, T. Generation of Hichart Program Diagrams, *Trans. IPSJ*, **31** (1990), 1463-1473 (in Japanese).