A Visualization Technique for Hierarchical Time Series Task Data

Yumiko Uchida Takayuki Itoh
Graduate School of Humanitics and Sciences, Ochanomizu University
E-mail {yumi-ko,itot}@itolab.is.ac.jp

Abstract

This paper presents an information visualization technique that represents both time series information and hierarchical structure onto a two-dimensional display space. It represents time series data as bar charts along the horizontal axis of a screen, and hierarchical structure as nested rectangular borders enclosing the bar charts. It packs the nested rectangular borders representing hierarchical structure so that it can represent a lot of information in a limited display space. We suppose there are various applications of the technique since we can obtain much information from large-scale data in one display.

This paper shows the visualization result of a task table used for business progress management and loads of a parallel computing environment as examples of results of the proposal technique.

1. Introduction

Information visualization is an active research area which helps understanding of abstract information applying computer graphics. An advantage of information visualization is that users can intuitively understand data than reading them expressed only by characters or numerical values.

Shneiderman has advocated seven types of data structure: one dimension, two-dimension, three-dimension, n dimension (n>3), time series, hierarchical structure, and graph, as a classification of data used in information visualization [1]. There have been many techniques for visualizing each of above data structure. While on the other hand, many techniques for visualizing data compounded above seven types of data structures is actively proposed.

This paper focuses on hierarchical and time series data from the above seven data structure. Here, we define hierarchical structure data as the data forming a tree structure, while those each data element is hierarchically grouped. We also define time series data as the data representing the time-varying change of a certain phenomenon. Visualization of time series data is very valuable, because it is generally difficult to discover the temporal response of a certain phenomenon from character-based description of time series data.

This paper presents a technique for visualizing hierarchical time series task data. We define a task as the operating unit which has start time and end time. Here we treat each task as leaf node, and group of tasks as non-leaf nodes, so that we can form collection of tasks as hierarchical data.

This technique represents a time series along the horizontal axis on a screen, and represents each task using bar charts - horizontally stretched thin rectangles. Moreover, the technique represents the hierarchical structure of task groups as nested rectangular borders. This representation follows our hierarchical data visualization technique "HeiankyoView" [2]. The technique proposed in this paper can represent the whole large-scale data containing a lot of tasks in one display space, by packing the rectangle areas so that the amount of blanks are minimized in the display.

As an example of application of this technique, we consider the visualization of a task table used for business progress management of companies. Typical task tables record assignments, schedule, degree of progress, and so on. Although commercial production of the task management tool is progressing in recent years, most of them display a schedule and a degree of progress divided into two or more windows. Therefore, it is controversial to develop visualization technique which enables us to simultaneously get a lot of information such as task progress and schedule.

We also visualize the load of parallel computing environment as task data, as another example of application of the technique. We assume that each thread of parallel computing as a task, and the threads record start time, end time, load to a computer, and so on. We can form hierarchical time series task data by classifying these task groups according to their various attributes, such as types of threads or assigned computers. We expect visualizing these data will support to determine suitability of scheduling of hierarchical parallel computing environment. This paper tests an availability of proposal technique, as we visualize the processing time monitored in actual parallel computing environment and discussing its scheduling results.

2. Related Work

The technique proposed in this paper visualizes hierarchical data structure in one display space, by packing data items onto a display space while it attempts to minimize the display space usage. This approach is often called "space-filling approach", where HeiankyoView [2] is a typical space-filling hierarchical data visualization technique. The most famous space-filling hierarchical data visualization technique is TreeMaps [3] proposed by Johnson et al., which recursively divides a display space to form a nested band chart. There have been several variations of TreeMaps that improve its appearance and capability [4] [5]. The technique proposed in this paper is a kind of the combination of space-filling hierarchical data visualization technique with time-varying data visualization technique.

There have been several works which represent task data. Some of works extend Gantt Chart, a typical representation of task data. Some other works extend bar charts to represent hierarchical or network time series data [6]. There has been also a technique which represent time series and hierarchical structure by combining a bar graph, a line graph, and figures representing network on a display [7] [8]. Against these work, the proposed technique attempts to minimize the display space usage while it simultaneously represents both hierarchical and time-varying information.

3. HeiankyoView

HeiankyoView [2] is one of the techniques for visualizing hierarchical data, which represent leaf nodes as square icons, and non-leaf node as rectangular frames. Fig. 1 shows an example of visualization result of hierarchical data by using HeiankyoView. A feature of HeiankyoView is placing all leaf nodes over the whole hierarchical data in one display space. It efficiently uses display area by packing rectangles so that the blank spaces are minimized. HeiankyoView is appropriate for visualizing large-scale hierarchical data, since it can display many nodes in one display space, representing a hierarchical structure.

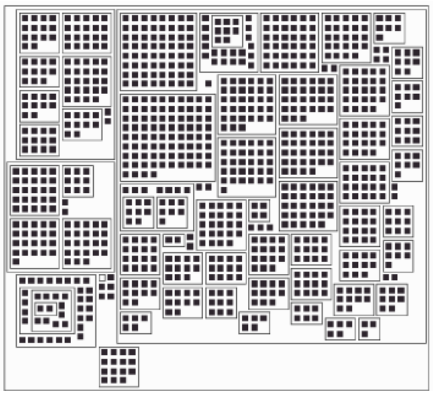


Fig.1 ： An example of visualization result of hierarchical data by HeiankyoView

Our technique proposed in this paper represents time series information using bar charts like typical existing techniques, and hierarchical structure as rectangular borders like HeiankyoView.

4. Proposal Technique

4.1 Data Structure

Table 1 shows an example of data for our technique, which forms hierarchy as

$$\text{Project} \supset \text{Function} \supset \text{Task} \supset \text{STask}.$$

Fig. 2 represents the data of Table 1 as hierarchical data. We treat lowest data elements (STask in the case of Table 1) of each task as leaf nodes, and the higher level unit (Project, Function, Task in the case of Table 1) of each task as non-leaf nodes. We assume that each leaf node contains start and end time, respectively, and treat them as time series information.

Table1: An example of task table.

Task	Schedule	
	Start Day	End Day
Project		
Function1		
Task1.1		
STask1.1.1	2006/04/01	2006/04/30
STask1.1.2	2006/04/01	2006/04/30
Task1.2		
STask1.2.1	2006/05/01	2006/05/31
STask1.2.2	2006/05/01	2006/05/31
STask1.2.3	2006/05/01	2006/05/31
STask1.2.4	2006/05/01	2006/05/31
Function2		
Task2.1		
: :	:	:
: :	:	:

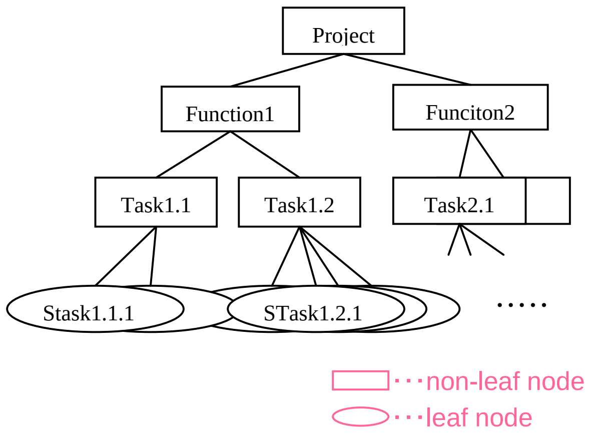


Fig.2 : Representation of the data in Table 1 as hierarchical data.

The technique proposed in this paper represents leaf nodes as bar charts, where those lengths are proportional to their duration (between start day and end day in the case of Table 1), and non-leaf nodes as nested rectangular borders which enclose bar charts.

4.2 Overview of this technique

Fig. 3 shows the overview of the rectangle packing algorithm used in this technique. The technique assumes the time-axis as a horizontal axis of a display, and set up a horizontal base point line. The technique places bar charts on a base point line so that they may not overlap with each other, and then encloses them by a rectangular border. It places these rectangles so that they may not overlap each other, and blank spaces are minimized. Repeating this process recursively from lowest to highest levels of hierarchy, it represents the whole data.

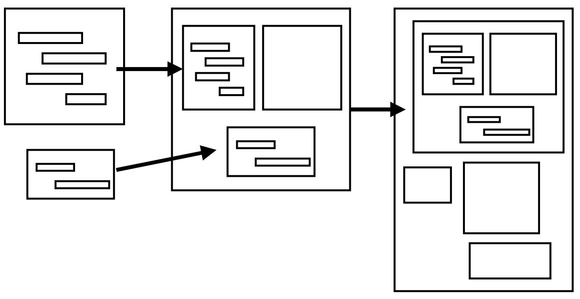


Fig.3 : The overview of the packing algorithm applied in this technique.

4.3 Algorithm for Packing Rectangles

Let positions of the vertices of an already placed rectangle R_j as shown in Fig. 4. The technique calculates the x-coordinate value of the vertices of the rectangle R_i according to the start time and the end time (start day and end day in the case of Table 1). If the calculated x-coordinate value of R_i is set to x_{ai}, x_{bi} ($x_{ai} < x_{bi}$) and satisfies equation (1), it may overlap with R_i (See Fig. 5).

$$x_{aj} < x_{bi} \cap x_{ai} < x_{bj} \ldots(1)$$

Referring to positions of R_j satisfying equation (1), the technique determines the position of R_i

Supposing there are k rectangles which satisfy equation (1). For these rectangles, the technique varies y_{cj} in ascending order, as $y'_{c1}, y'_{c2}, \ldots y'_{ck}$, and y_{dj} in ascending order, as $y'_{d1}, y'_{d2}, \ldots y'_{dk}$.

From $s=1$ to order

1) Place R_i temporarily where the y-coordinate of the lower edge of R_i as $y_{di} = y'_{ds}$.

2) Find R_j which satisfies $y'_{cj} < y_{di} \cap y_{ci} < y'_{dj}$.

・If there is at least one rectangle which satisfies the above condition, R_i overlaps to the already placed rectangle. Then let s increase by one and return to 1).

・If there are no rectangles which satisfy the above equation, we arrange R_i as $y_{di} = y'_{ds}$ (See Fig. 6).

Applying this process for all R_i, we determine positions of all bars and rectangles.

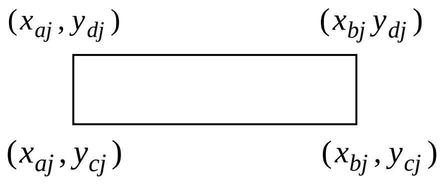
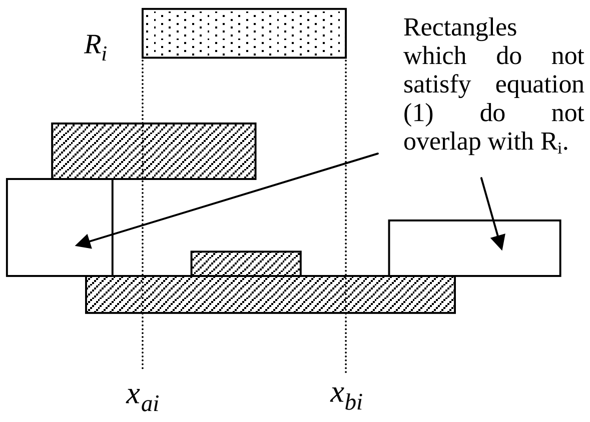


Fig.4：Positions of vertices of a rectangle.

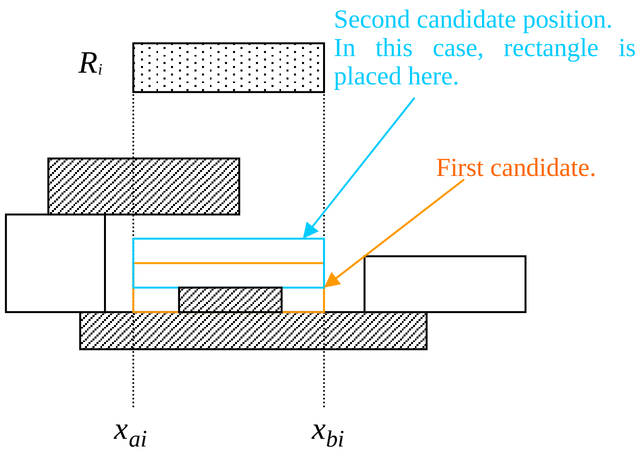


R_i
Rectangles which do not satisfy equation (1) do not overlap with R_i.
x_{ai}
x_{bi}

Fig. 5: Condition whether rectangles overlap to currently placing rectangle.

5. Results

We implemented the proposed technique on Java 1.5, and executed on HP dc5700 Small Form (CPU 2.8GHz, RAM 0.99GB) with Windows XP. The following results will be also published as color document files at http://itolab.is.ocha.ac.jp/ .



Second candidate position. In this case, rectangle is placed here.
R_i
First candidate.
x_{ai}
x_{bi}

Fig.6: Searching for the position in which the currently placing rectangle does not overlap to other rectangles.

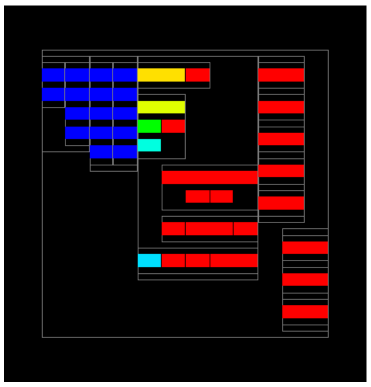


Fig. 7： A visualization result of a task table

Fig. 7 shows a task table visualized by our technique. Here, colors of bar charts denote degree of progress of task, where a redness bar chart represents a task with low degree of progress, and a blueness bar chart represents a task with high degree of progress. This technique helps us to understand the distribution of the section behind schedule.

This section also introduces another example of visualization of task data, constructed from load data of parallel computing environment, as shown in Fig. 8. Here, bars denote threads of parallel computing processes, and rectangular borders denote hierarchy of the threads. Color of bars denotes degree of their scales (such as CPU time or memory usage) so that redness bars represent large threads, and blueness bars represent small threads.

This experiment classifies threads and constructs hierarchy according to the following rules:

- There are same numbers of firstly executed threads and secondly executed threads. This section calls former threads "thread A", and latter threads "thread B".
- This experiment forms low-level groups gathering several thread A, and then forms a high-level group consists of all thread A, by collecting all the low-level groups. The experiment similarly forms two-levels of groups of thread B.
- Thread A relates to thread B one for one. A thread B can not start until the related thread A is completed.

One of the typical examples of processes forming such hierarchy is parallel processing of graphics hardware, where coordinate conversion process corresponds to thread A, and pixel process corresponds to thread B.

In Fig. 8, this technique displays whole thread A on the upper half of the visualization result, and whole thread B is displayed on the lower half.

Fig. 8 (upper) shows an example of visualizing parallel computing environment, where processing time of thread A is shorter than that of thread B. Here, parallel computing starts all threads of a group of thread B after all threads of the related group of thread A are completed. However, if threads of another group of thread B still run at this time, parallel computing must wait to complete all of them, and after that it can start the threads of another group of thread B. Fig. 8 (upper) shows a situation that two or more groups of thread B continue even if all thread of thread A is completed. Fig. 9 (left) is a zoom-up of the right end of Fig 8. (upper), which represents several threads of thread B still run after all threads of thread A is completed.

Fig. 8 (middle) also shows an example of visualization result of parallel computing environment, where processing time of thread A is shorter than that of thread B. Unlike the previous example shown in Fig. 8 (upper), parallel computing starts the threads of a group of thread B unconditionally, when threads of the related group of thread A is completed. This visualization result shows that parallel computing completes whole thread A and B almost simultaneous. Fig. 9 (middle) is a zoom-up of the right end of Fig. 8 (middle), which represents thread A and B are completed almost simultaneous. Moreover, this result also shows that a load of CPU becomes larger since processing of two or more group of thread B runs at certain time.

Fig. 8 (lower) shows an example of visualization result of parallel computing environment, where processing time of thread A is longer than that of thread B. Here, similar as Fig. 8 (upper), parallel computing starts threads of a group of thread B after threads of the related group of thread A and other groups of thread B are completed. This visualization result shows that a group of thread B has finished processing while a group of thread A is running. Moreover, this result also shows that idle time is arising in a part of processing of thread B. Fig. 9 (right) is a zoom-up of Fig. 8 (lower), which represents that thread B is often idle.

We expect that such visualization shown in Figures 8 and 9 can contribute to configure parallel computing environment.

6. Conclusion and Future Work

This paper proposed a technique for visualizing hierarchical time series task data, and showed visualization result of a task table and parallel computing environment.

As a future work, we would like to develop a technique to place rectangles as seamless as possible when the period of a task is changed. Furthermore, we would like to consider of dependency among tasks for their placement. We are also planning to conduct a user test using task data containing certain problems, to proof the availability of this technique. Ultimate targets of this study include simulations of scheduling of tasks which assume various situations.

Acknowledgment

We appreciate Masato Oguchi, Professor of Graduate School of Humanitics and Sciences, Ochanomizu University, for his advices about

visualization of parallel computing environment.

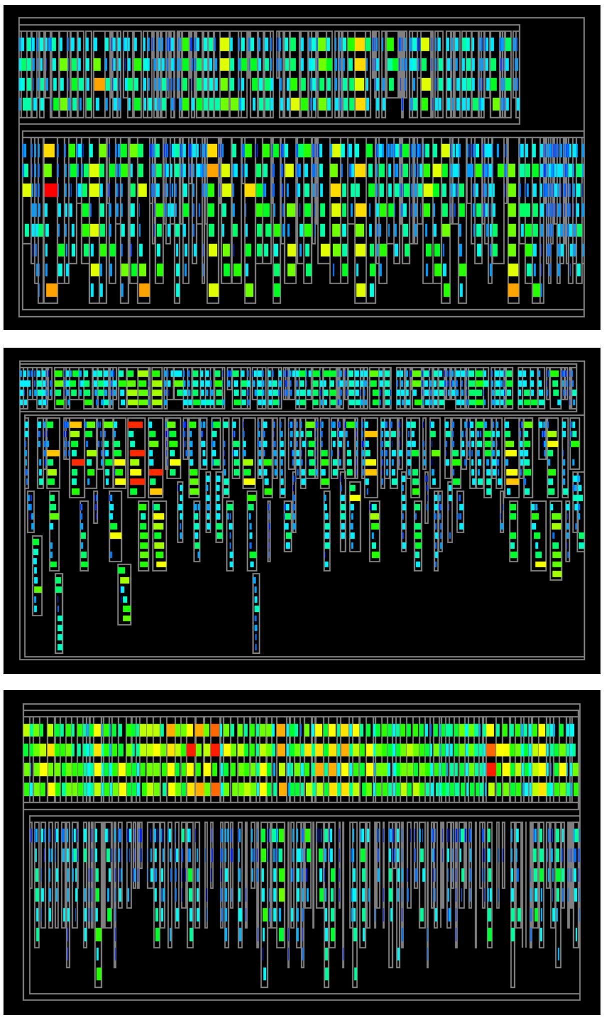


Fig. 8 ： Examples of visualization of tasks obtained from a parallel computing

References

[1] Shneiderman B., The Eyes Have It: A Task by Data Type Taxonomy for Information Visualization, IEEE Symposium on Visual Languages '96, pp. 336-343, 1996.

[2] Itoh T., Takakura H., Sawada A., Koyamada K., Hierarchical Visualization of Network Intrusion Detection Data in the IP Address Space, IEEE Computer Graphics and Applications, Vol. 26, No. 2, pp. 40-47, 2006.

[3] Johonson B., Scheneiderman B., Tree-Maps: A Space Filling Approach to the Visualization of Hierarchical Information Space, IEEE Visualization ’91, pp. 275-282, 1991.

[4] Bederson B., Scheneiderman B., Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies, ACM Transaction on Graphics, Vol. 21 No. 4, pp. 833-854, 2002.

[5] Bruls D. M., Huizing K., Wijk J. J., Squarified Treemaps, Proceedings of Data Visualization 2002, pp. 33-42, 2000.

[6] Herrmann J. W., A History of Decision-Making Tools for Production Scheduling, Multidisciplinary Conference on Scheduling: Theory and Applications, 2005.

[7] Russell, A. D., Udaipurwala, A., Construction Schedule Visualization, Proceedings of the International Workshop on Information Technology in Civil Engineering, 2-3 November 2002, pp. 167-178, 2002.

[8] Russell, A. D., Udaipurwala, A., Integrated Project Planning through Hierarchical Scheduling, Proceedings of Specialty Conference on Fully Integrated and Automated Project Processes, pp. 311-321, 2002.

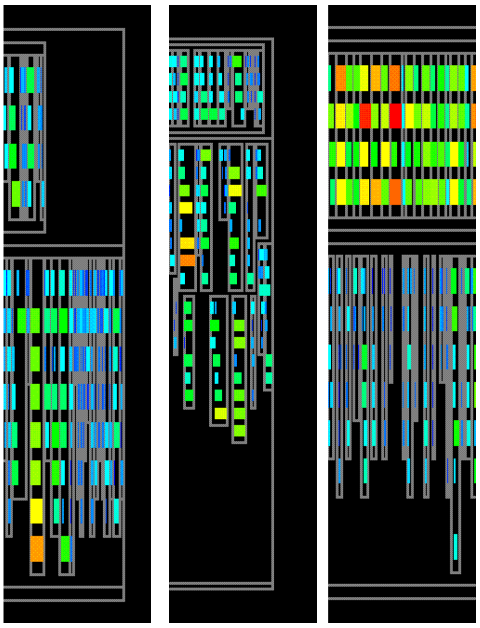


Fig. 9： Extended figure of distinct part of Fig. 8