Journal of Computers Vol. 30 No. 5, 2019, pp. 31-45
doi:10.3966/199115992019103005003

The Ordered Treemap of Weight Divided Layout Algorithm

Bo Yang[1], Weiqun Cao[1*]

[1] School of Information Science & Technology, Beijing Forestry University, Beijing, China
yb2219@163.com, weiqun.cao@126.com

Received 25 October 2017; Revised 12 April 2018; Accepted 2 June 2018

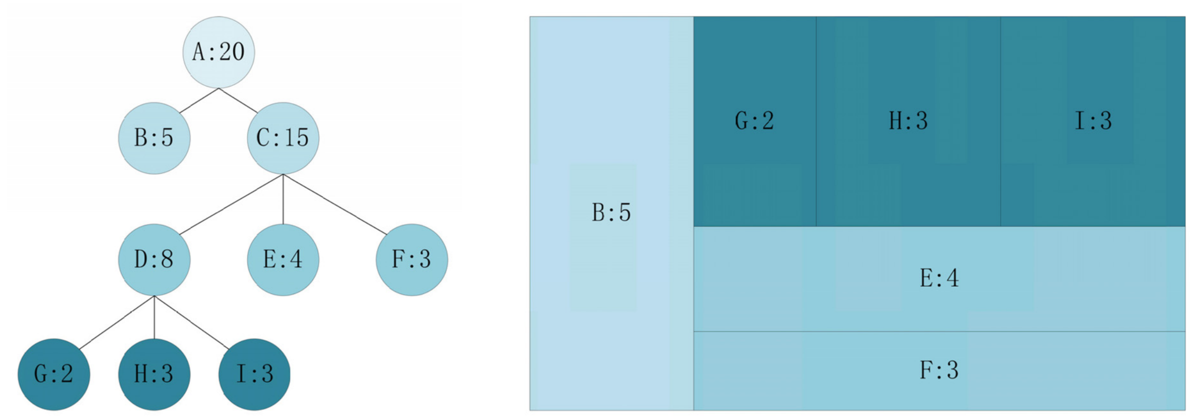
Abstract. Existing treemap layout algorithms are inefficient sequentially, while ordered treemap layout algorithms are incapable of squarifying ability and providing continuity. In this paper, we propose a type of ordered treemap of weight divided layout algorithm. The overall idea of the algorithm is to bundle multiple nodes into two nodes with similar weights. Then, the long edge segmentation principle is used to design the rectangular area of the treemap. Through the application of the "divide and conquer" rule for the new structure of the two power value nodes, the layouts of all the single nodes are accomplished by repeating the above process until completion. We used 50 randomly generated datasets and compared nine types of layout algorithms. The comparison and analysis are made with six indexes for our algorithm. The experimental results show that the algorithm has an advantage in terms of average aspect ratio, continuity, and distance correlation, and that it is suitable for a hierarchical data visualization display that has order requirements.

Keywords: hierarchical data, layout algorithm, ordered treemap, square treemap

1 Introduction

Information visualization is an interdisciplinary subject that emerged with the development of computer graphics, man-machine interaction, visual design and psychology. In today's big data era background, the rapid processing and display of massive amounts of data is one main problem to be solved. Hierarchical data represents an important data structure of big data that emphasizes hierarchical relationships between individuals. Hierarchical relationships are mainly divided into two categories: contains and subordinate. For example, China contains some provinces; each province contains some cities, and each city is divided into several counties; and several files and directories are in the computer system. The hierarchical data visualization includes two main types: node-link and space-filling [1]. The main research of this paper focuses on the most typical layout of the space filling method, i.e., the treemap. A Treemap is one of the representative works of early information visualization research and was first proposed by Johnson in 1991 [2]. A treemap is a visualization method of two-dimensional space filling, and it can very efficiently use area and color to represent the distribution of leaf nodes in a tree structure. Additionally, it has the obvious advantages of high space utilization and is suitable for displaying the hierarchical structure of the data collection on a large scale. As shown in Fig. 1, the tree structure of Fig. 1(a) can be represented as the Treemap of Fig. 1(b). The structure starts from the root node of the tree. The screen space is divided into multiple rectangles according to the data of the child nodes, and the area of the rectangle represents the numerical attributes of the node. The recursion obtains the final Treemap.
The core problem of treemap visualization is how to divide the screen space recursively. Since the inception of the treemap, various layout algorithms of the treemap have emerged. Many improvement algorithms have been developed on the indexes of the treemap layout algorithm, which includes the average aspect ratio [3], continuity [4], readability [5], average distance change [3], the variance of distance changes [3] and distance correlation [6]. Treemaps are mainly divided into rectangular and non-rectangular treemaps [7]. The rectangular treemap layout algorithm includes the following types: Slice-

* Corresponding Author



(a) Tree structure (b) Treemap

Fig. 1. Tree structure transforms into a treemap

and-Dice [2] which is the most original; the Squarified [8], which is focused on the square layout; the Strip [5] and the Spiral [4], which are focused on the sequential layout; and the Pivot [3], which is for achieving improved square and sequential layouts. Itoh [9] proposed the layout algorithm based on the Delaunay triangular grid, which effectively improved the performance of the squarifying ability. Jarke [10] proposed the cushion-treemap, which set different shading shadows for the partition area to enhance the hierarchical structure. The non-rectangular treemap layout algorithm includes the following: the polygon filling algorithm, represented by the Voronoi Treemap [11]; the triangle filling algorithm, represented by the Divide-and-Conquer Treemap [12]; and the circular filling algorithm, represented by the Circular Treemap [13]. Based on variations of the circular treemap layout algorithm, there are CropCircles [14], Circle-Packing [15], ClockMap [16], and an improved algorithm by Zhao et al. [17] for improving the spatial utilization of the circular treemap. There are hybrid treemap algorithms that use a variety of layout methods, i.e., the Hybrid Treemap [18]. Baudel and Broeksema [19] defined the design space of sequential, rectangular and space-filling layouts. This space is covered by five attributes, namely, the layout order, weight definition, the principle of filling, recursive position and direction of the filling. Through the existing layout algorithms, the user-defined rectangle treemap layout scheme is implemented. Xin and Yuan [20] summarized the basic layout algorithms, attribute design and interaction design of treemaps, analyzed the algorithms of a non-rectangular treemap from its scalability, summarized the effect evaluation method, the use in actual cases, and the development prospect of treemap research and its application. Chen et al. [7] analyzed and compared the standard treemap layout algorithms above and analyzed the various performances of treemaps through a variety of evaluation indexes. At the same time, they proposed that various algorithms can be combined efficiently and used flexibly according to the actual condition in the application so that the algorithms can learn from each other and achieve better performance indexes.

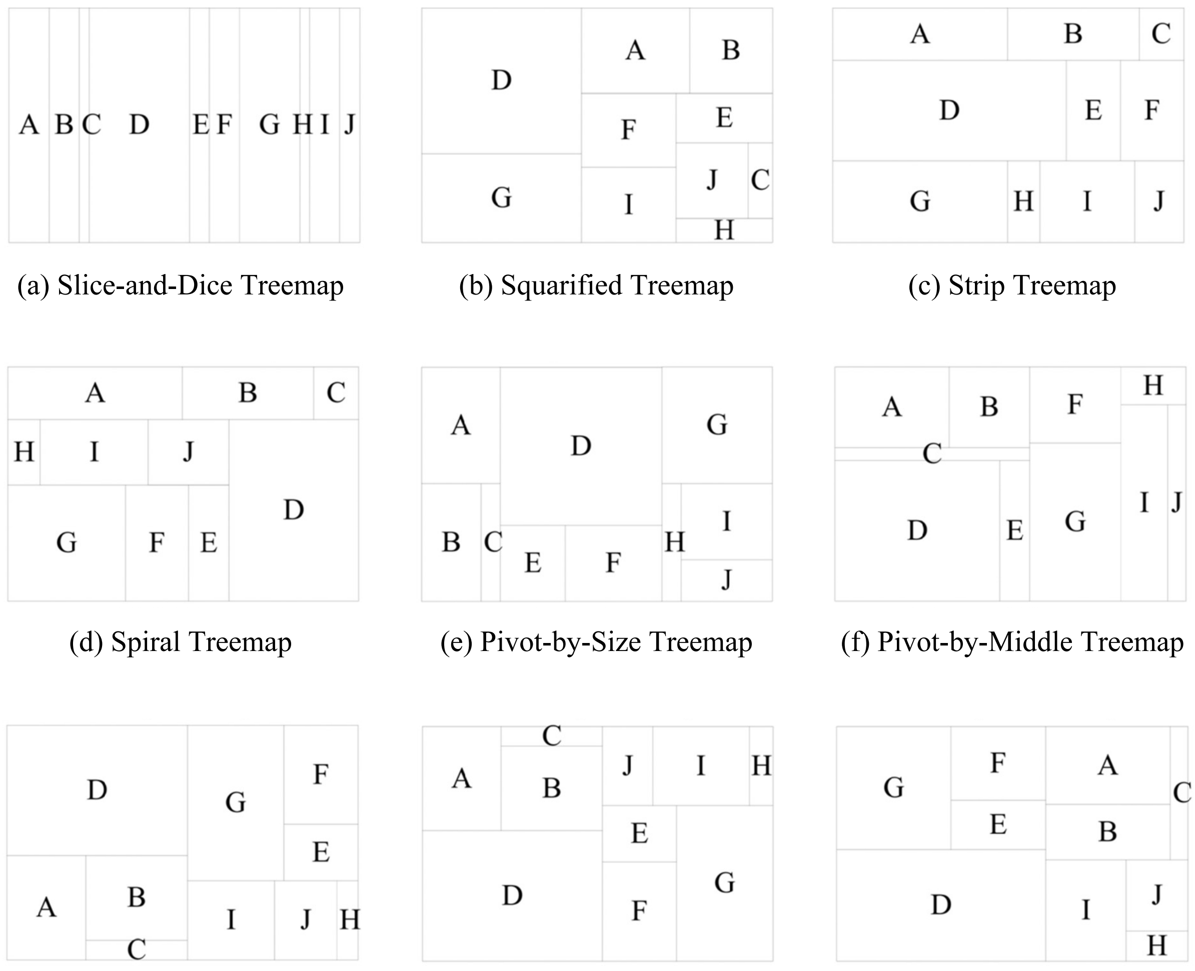
In the hierarchical data structure, we need to consider the order and continuity of some datasets. The existing treemap layout algorithms are inefficient sequentially, while ordered treemap layout algorithms are incapable of squarifying and providing continuity. We proposed the ordered treemap of weight divided layout algorithm (OTWDLA) for the type of data in this paper. The algorithm uses the idea of weight equalization and the principle of long edge division to divide the rectangular blocks. The structural nodes are divided by using the idea of “divide and conquer;” then, all treemap nodes are divided. We first describe and analyze the advantages and disadvantages of nine types of layout algorithms, which were introduced previously, and then, through the program design, implement these algorithms individually. Then, we compare and analyze the six evaluation indexes. This algorithm can quickly determine the dataset segmentation points, efficiently reduce the segmentation time, and effectively reduce the data uncertainty in the segmentation results to maintain a specific performance of the average aspect ratio, continuity and distance correlation of the layout, which is suitable for a visual representation of hierarchical data with sequential data requirements.

2 Treemap Layout Algorithm

The core idea of a rectangular treemap layout algorithm is how to divide the screen space for the rectangular nodes recursively into rectangular sections. In a given tree hierarchy structure, for the node weight and node sequence, the parent node is divided according to the weights of the child nodes, and then, the whole layout algorithm is defined. This paper introduces several rectangular treemap layout algorithms and analyzes their advantages and disadvantages.

2.1 The Slice-and-Dice Layout Algorithm

The Slice-and-Dice [2] layout algorithm was proposed by Johnson in 1991; the algorithm adopts the vertical and horizontal alternating method to segment the rectangle drawing area of the layout, from left to right or from top to bottom in turn to fill the rectangular area, only considering the order of the nodes and the proportion of weight in the process of filling. Fig. 2(a) shows the effect of this layout algorithm. This algorithm maintains the sequence and continuity of datasets, but its disadvantage is that it is easily generates long and thin rectangles that are not conducive to human eye recognition or the click interaction of the mouse.



(a) Slice-and-Dice Treemap (b) Squarified Treemap (c) Strip Treemap

(d) Spiral Treemap (e) Pivot-by-Size Treemap (f) Pivot-by-Middle Treemap

(g) Squarified-SP Treemap (h) SOTLA Treemap (i) Squrified-AMA Treemap

Fig. 2. Treemap layout algorithms

2.2 The Squarified Layout Algorithm

The Squarified layout algorithm, which was proposed by Bruls [8], is used to solve the problem of the Slice-and-Dice algorithm, i.e., that a large number of long and thin rectangles were formed that are difficult for the human eye to identify. The layout algorithm tries to make the rectangles as close as possible to squares. First, the algorithm sorts the original dataset from large to small according to weights. Second, the greedy strategy is used to allocate the power value of the nodes. The filling of each node is chosen such that the average aspect ratio is close to one, and the iterative loop continues until all nodes are assigned. As shown in Fig. 2(b), the algorithm enables the human eye to identify the nodes easily, but the original order of nodes is disturbed, and finding a particular node is difficult.

2.3 The Strip Layout Algorithm

The Strip layout algorithm, proposed by Bederson et al. [5], improves the readability and continuity of the treemap layout. The layout strategy of this algorithm works by processing the input rectangles in order and laying them out in horizontal strips of varying thicknesses, as shown in Fig. 2(c). The algorithm is efficient in that it can achieve simple, continuous, and stable structures with high readability.

2.4 The Spiral Layout Algorithm

The Spiral layout algorithm, proposed by Tu and Shen [4], is designed to improve the continuity of the data further. Its layout strategy is to choose a Spiral path layout, as shown in Fig. 2(d). The algorithm can highlight the change of layout and visual contrast.

2.5 The Pivot Layout Algorithm

The aim of the Pivot layout algorithm, proposed by Shneiderman and Wattenberg [3], is to solve the problem of the disorderly Squarified algorithm. It adopts the idea of partition incremental refinement to divide the filling area of the dataset into four parts, i.e., R_1, R_p, R_2 and R_3, to maintain the average aspect ratio and order of the treemap nodes. To lay out in the current selected rectangular area and ensure that it is allocated according to the order of R_1, R_p, R_2, and R_3, the node selection strategy of R_p includes two algorithms. One is the Pivot-by-Size algorithm, which is selected according to the maximum node weight value; the layout result is shown in Fig. 2(e). In addition, another is the Pivot-by-Middle algorithm, which is selected according to the intermediate node ordered by the node number; the layout result is shown in Fig. 2(f). If the number of nodes is greater than 1 in the area of R_1, R_2, and R_3, then they are respectively iterated until all the nodes are distributed. This algorithm guarantees the partial orderliness of the nodes while obtaining a better average aspect ratio, but it is not entirely ordinal.

2.6 The Squarified-SP Layout Algorithm

The aim of the Squarified-SP layout algorithm, proposed by Chen et al. [21] in 2013, is mainly to improve the order and stability of the Squarified algorithm. The core idea of the algorithm is according to the principle of weight dividing the dataset into a smaller subset of data and, simultaneously, according to the proportional relationship, cutting the rectangular area to obtain the rectangular sub-block. Then, the Squarified algorithm is used in the rectangular sub-block corresponding data subset to obtain the final effect of the layout. The algorithm introduces a parameter μ that represents the maximum number of rectangles allowed in each rectangular sub-block, which the user can customize according to their needs. The Squarified algorithm is performed when the number of blocks in the rectangle is less than μ, otherwise, it continues to be divided into smaller rectangles. The layout effect of this algorithm is shown in Fig. 2(g).

2.7 The SOTLA Layout Algorithm

The aim of the Squarified and ordered treemap layout algorithm (SOTLA), proposed by Hu et al. [22] in 2014, is mainly to solve the relatively poor order of the rectangular treemap and the poor performance of the orderly treemap algorithms. It combines the Squarified priority power value, the partition of the Pivot, and the layout path of the Strip and Spiral. The algorithm first bundles multiple nodes to combine the power value nodes. The construction principle is to choose the maximum of the dataset, calculate the sum of the items before the maximum value, item by item, and then compare it with the maximum value again. The summary data of the nearest maximum are divided into one group, and the remaining data are divided according to the principle. Calculate the sum of the items after the maximum value of the data, according to the principle above, so that the combined nodes are obtained. Then, the nodes are laid out according to the principle of the average aspect ratio closest to 1, and the path is filled in the shortest direction. The layout of each combination node is completed by dividing and repeating the process above. Finally, the layout of all individual nodes is completed. The layout effect is shown in Fig. 2(h).

2.8 The Squarified-AMA Layout Algorithm

The aim of the Squarified based on Approximate Maximum Average (Squarified-AMA) algorithm, proposed by Zhang et al. [23] in 2016, is mainly to improve the disorder and stability of the Squarified algorithm. The principle of this layout algorithm is first to restructure the original dataset similar to that of SOTLA to select the maximum of the data, calculate the sum of the items before the maximum value, item by item, and then compare it with the maximum value again. The summary data nearest the maximum are divided into one group, the remaining data are divided according to the principle, and the elements after the maximum are divided by the same strategy as the elements before the maximum value. The new combination data will be laid out according to the principle of the average aspect ratio and the shortest edge path. A combination of more than four elements in a combined dataset is divided according to the previous procedures until all element layouts are completed. The layout effect is shown in Fig. 2(i).

2.9 Comparative Analysis of Algorithms

Six treemap layout algorithm indexes were qualitatively or quantitatively analyzed with all the types of existing algorithms by using the data to evaluate the performance of the different treemap layout algorithms. We summarized the advantages and disadvantages of these nine different layout algorithms for all the evaluation indexes, as shown in Table 1. We see that the overall performance of the Slice-and-Dice algorithm is the best in the existing algorithms, but the average aspect ratio is very poor. The average aspect ratio of the Squarified algorithm is best, but other aspects of performance are not good. For the Strip and Spiral, the stability is poor, but the other performance aspects are good. The other five types of algorithms have poorer readability and stability, but the other related performance aspects each have advantages and disadvantages. Because of the difference of the layout algorithms, the user can choose treemap algorithms according to their need.

Table 1. Comparison of treemap algorithms

Evaluation Index	AAR	C	R	ADC	VDC	DC
Slice-and-Dice	☆	☆☆☆☆☆	☆☆☆☆☆	☆☆☆	☆☆☆☆☆	☆☆☆☆☆
Squarified	☆☆☆☆☆	☆	☆	☆	☆☆☆	☆☆☆
Strip	☆☆☆	☆☆☆☆	☆☆☆☆	☆	☆☆☆	☆☆☆☆
Spiral	☆☆☆	☆☆☆☆☆	☆☆☆	☆	☆☆☆	☆☆☆
Pivot-by-middle	☆☆☆	☆☆☆	☆	☆☆	☆☆☆	☆☆☆☆☆
Pivot-by-size	☆☆☆	☆☆☆	☆	☆	☆☆☆	☆☆☆☆☆
Squarified-SP	☆☆☆☆	☆☆☆	☆	☆☆	☆☆☆☆	☆☆☆☆☆
SOTLA	☆☆☆	☆☆☆	☆☆	☆	☆☆☆	☆☆☆☆
Squarified-AMA	☆☆☆	☆☆	☆	☆	☆☆☆	☆☆☆

Note. ☆☆☆☆☆ represents the best, ☆☆☆☆ represents good, ☆☆☆ represent average, ☆☆ represents general, ☆ represents poor.

3 OTWDLA

In this paper, according to the principle of equipartition of weights to divide a treemap, we introduce two types of algorithms to find the suitable division point: one is a sequential traversal dataset and the other is middle division. According to the principle of this segmentation, we can quickly determine the point of division at the same time in each partition. There are only two types of circumstances so that the uncertainty of data segmentation can be reduced. The main idea of the two algorithms is that dividing the dataset into a smaller subset of data by adopting the idea of the weights of a binary, iterative loop will be used to map the data, obtaining the final layout.

3.1 Algorithm Description

Before describing the algorithm of this paper, some basic definitions are provided. The inputs are a rectangle area R; its length is h and width is w, and the coordinate of the top left corner is (x, y). We define the original dataset as $O = \{a_1, a_2, ..., a_i, ..., a_n\}$, $i = 1, 2, ..., n$, where we set the number of data as n. O_1, O_2 are defined as two subsets of the original dataset after dividing ($O_1 \cup O_2 = O$). We define n_1 and n_2 as the number of data of the two subsets ($n_1 + n_2 = n$). The sum of the total weight of the original dataset is defined as s so that s_i represents the sum of the total weight of O_i. Thus, $s_1 + s_2 = s$. P is defined as the segmentation of the dataset and represents the sequence number of the end element of a subset O_1, $P \in 1, 2, \ldots, n$.

Weight Divided-Binary Search Algorithm, OTWDLAB, is described below:

Step 1. If $n > 1$, compute the collection partition position $p = \lfloor n/2 \rfloor$, and divide the dataset into two subsets, O_1 and O_2; then, calculate the difference between the weights of all elements of the two subsets $d_{old} = |s_1 - s_2|$; and then, jump to perform Step 2. If $n \le 1$, jump to perform Step 6.

Step 2. If $n_1 \ge 2$, p goes forward and obtains $p = p - 1$, repartition the dataset into two new subsets, O_1 and O_2; then, calculate the difference between the weights of all elements of the two subsets $d_{new} = |s_1 - s_2|$. Compare d_{old} and d_{new}; if $d_{new} < d_{old}$ so that $p = p - 1$, $d_{old} = d_{new}$, then jump to Step 2. If $d_{new} \ge d_{old}$ so that $p = p$, then jump to Step 3.

Step 3. If $n_2 \ge 2$, p returns to one and obtains $p = p + 1$, repartition the dataset to two new subsets, O_1 and O_2; then, calculate the difference between the weights of all elements of the two subsets $d_{new} = |s_1 - s_2|$. Compare d_{old} and d_{new}; if $d_{new} < d_{old}$, so that $p = p + 1$, $d_{old} = d_{new}$, then jump to Step 3. If $d_{new} \ge d_{old}$ so that $p = p$, then jump to Step 4.

Step 4. The final segmentation point p is searched through the above iteration loop. Divide the dataset into two subsets, O_1 and O_2. Based on the segmentation point, the sum weights of two subsets, s_1 and s_2, are each calculated. Based on the principle of long edge segmentation, if $w > h$, the width of the original rectangle R is divided according to the weights s_1/s and s_2/s of the O_1 and O_2, and the height remains unchanged. Obtain the corresponding segmentation rectangle R_1 of O_1, where the width is $w_1 = s_1/s \cdot w$, the height is $h_1 = h$, the upper left corner of the vertex coordinates is $(x_1, y_1) = (x, y)$. Then, obtain the corresponding segmentation rectangular R_2 of O_2, where the width is $w_2 = s_2/s \cdot w$, the height is $h_2 = h$, and the upper left corner of the vertex coordinates is $(x_2, y_2) = (x + x_1, y)$. If $w \le h$, the height of the original rectangle R is divided according to the weight of O_1 and O_2, and the width remains unchanged. Obtain the corresponding segmentation rectangle R_1 of O_1, where the width is $w_1 = w$, the height is $h_1 = s_1/s \cdot h$, and the upper left corner of the vertex coordinates is $(x_1, y_1) = (x, y)$. Then, obtain the corresponding segmentation rectangular R_2 of O_2, where the width is $w_2 = w$, the height is $h_2 = s_2/s \cdot h$, and the upper left corner of the vertex coordinates is $(x_2, y_2) = (x, y + y_1)$.

Step 5. Set the data subset O_1 and O_2 to O and the sub-rectangles, R_1, R_2, which are to be drawn are set respectively to the drawn rectangle R, and repeat Step 1 to Step 4 until all the data elements are separated completely.

Step 6. At this point, $n \le 1$. If $n = 1$, the height h, width w and the vertex coordinates of the upper left corner (x, y) of the rectangle R are to be drawn and are assigned to the corresponding drawn rectangle r_i of the sub-node a_i. Its height is $h_i = h$, the width is $w_i = w$, and the vertex coordinates of the upper left corner are $(x_i, y_i) = (x, y)$.

Step 7. The height, width, and coordinates of each sub-node are obtained through Step 1 to Step 6. Finally, the rectangle drawing area corresponding to each of the sub-nodes is drawn.

Next, we will introduce another algorithm, the **weight Divided-Sequential Search Algorithm, OTWDLAS,** which is described below:

Step 1. If $n > 1$, set the initial location of the set partition $p = 1$ and the initial size of the sum of weights d to positive infinity, namely, $d = +\infty$; then, jump to perform Step 2. If $n \leq 1$, jump to perform Step 6.

Step 2. Divide the original dataset into two new subsets O_1 and O_2 according to the location of p; then, calculate the difference between the weights of all elements of the two subsets $d_{cur} = |s_1\text{-}s_2|$. Compare d and d_{cur}; if $d_{cur} < d$ so that $p = p + 1$, then jump to Step 2. If $d_{cur} \geq d$ so that $p = p$, then jump to Step 3.

Step 3. This step is the same as the Step 4 of the OTWDLAB, so is not described here.

Step 4. Set the data subsets O_1 and O_2 to O; set the sub-rectangles R_1 and R_2 to the drawn rectangle R; and repeat Step 1 to Step 3 until all data elements are separated completely.

Step 5. At this point, $n \leq 1$. If $n = 1$, the height h, width w and the vertex coordinates of the upper left corner (x, y) of the rectangle R are to be drawn and are assigned to the corresponding drawn rectangle r_i of the sub-node a_i. The rectangle height is h_i=h, the width is w_i=w, and the vertex coordinates of the upper left corner are (x_i, y_i)= (x, y).

Step 6. The height, width, and coordinates of each sub-node are obtained through Step 1 to Step 5. Finally, the rectangle drawing area corresponding to each of the sub-nodes is drawn. Example Description

In this part, we will illustrate the proposed algorithm by an example. Suppose we have a rectangle with a width of 600 and height of 400, and furthermore, suppose that this rectangle must be subdivided into ten rectangles with areas of 4, 3, 1, 10, 2, 3, 6, 1, 3 and 2. Each element is represented by a rectangle whose area is the weight of the element. As shown in Fig. 3, according to the principle of equalization of power values, the elements 4, 3, 1 and 10 of the original dataset are combined into a new node 18. The elements 2, 3, 6, 1, 3 and 2 are combined into a new node 17 so that we can obtain new nodes 18 and 17 and then lay out the new node list. The rendering of the treemap layout algorithm in the second part of this paper is based on the drawing of this dataset.

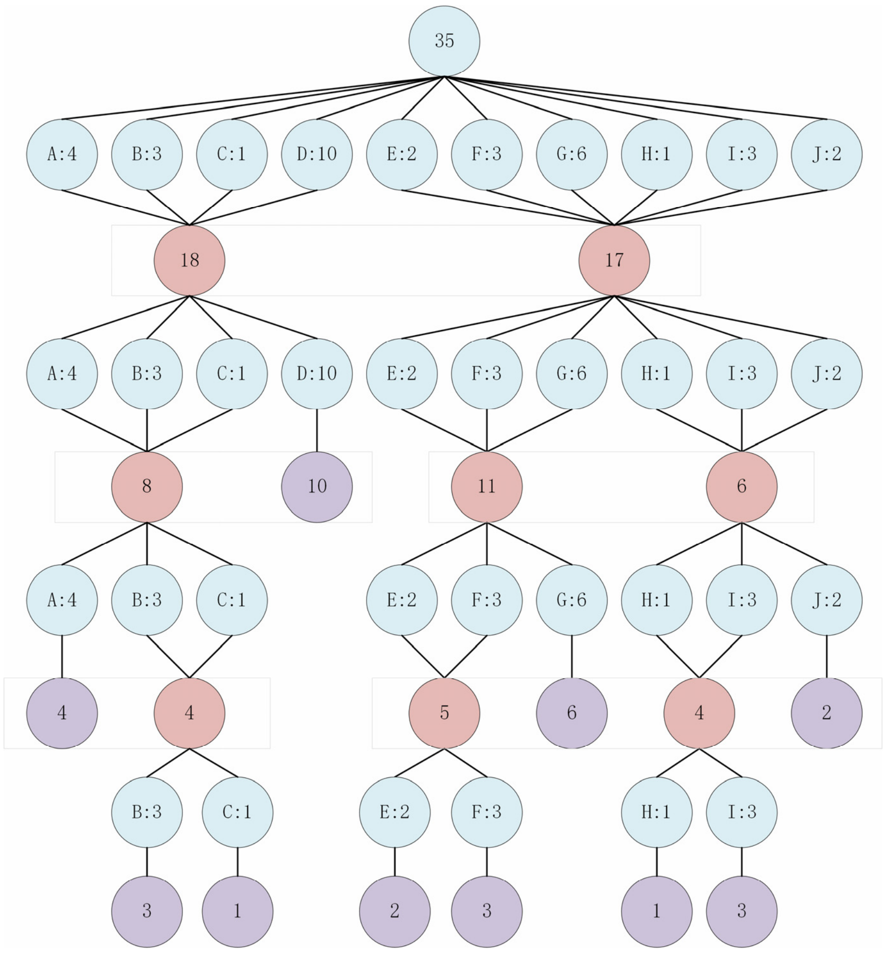


Fig. 3. Weights structure

The detailed drawing steps are shown below:

Step 1. By the principle of the weight structure in Fig. 3, in addition to the long edge segmentation principle, the original rectangle will be segmented into rectangular sub-blocks whose ratio is 18:17. At the same time, the width, length and top left corner vertex coordinates of the two rectangular blocks are obtained.

Step 2. To adopt the idea of "divide and conquer", we divide the rectangular sub-blocks with the weight of 18 and 17 into rectangular sub-blocks with ratios of 8:10 and 11:6, respectively.

Step 3. According to the above principle, all the elements are divided, the width, height and top left corner vertex coordinates of the ten rectangular sub-blocks are obtained, and then, all the sub-blocks are drawn to obtain the final effect.

The whole partition drawing process is shown in Fig. 4.

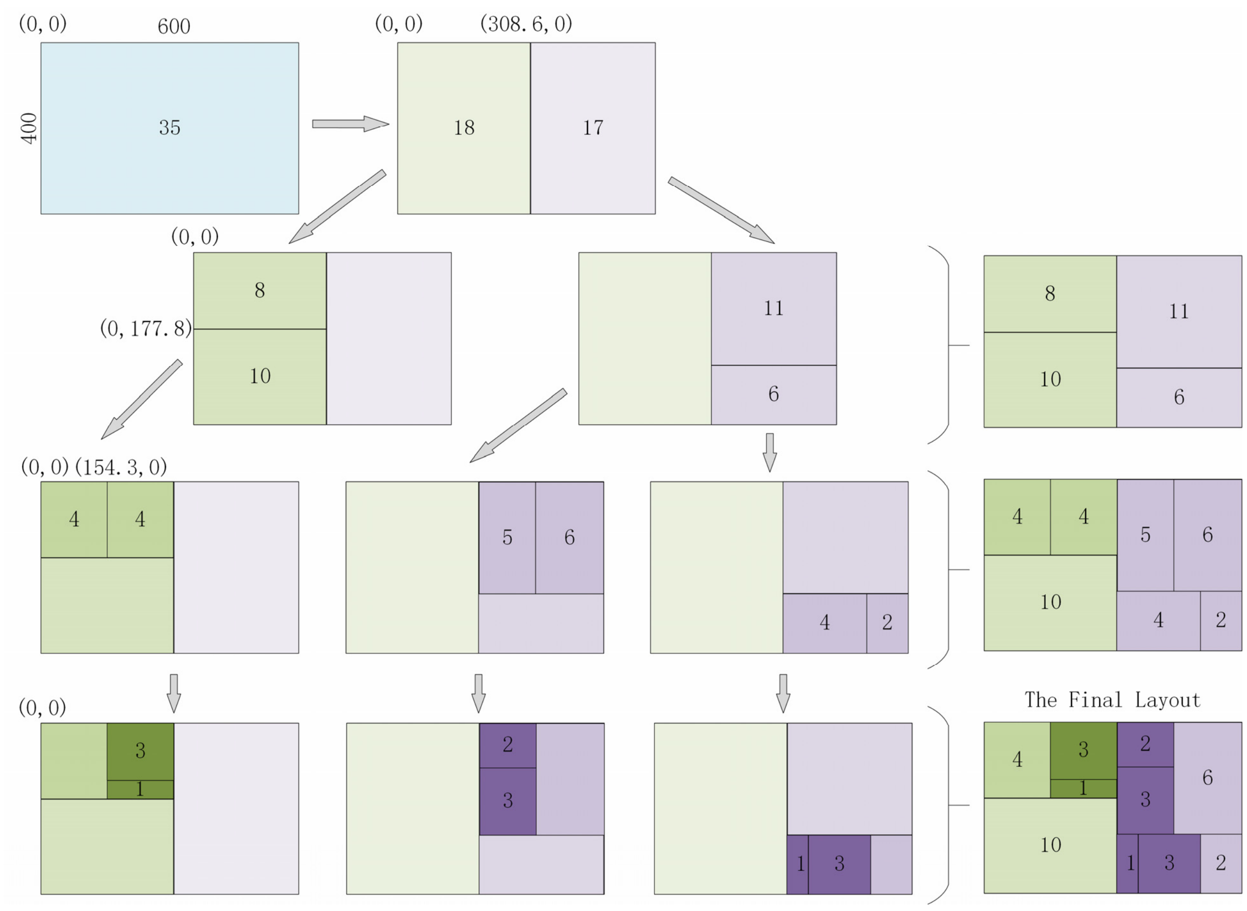


Fig. 4. Split drawing process

4 The Experimental Evaluation

4.1 Evaluation Indexes

This paper evaluates the performance of the layout algorithm for six commonly used evaluation indexes, including the average aspect ratio, continuity, readability, average distance change, the variance of distance changes and distance correlation [3-7, 13].

The average aspect ratio (AAR) [3] is used to describe the approximate degrees of the squares and rectangles of the treemap. The ratio is the unweighted arithmetic average of the aspect ratios of all leaf-node rectangles. The aspect ratio is defined as the maximum value of the ratio of length to width and the ratio of width to length. The aspect ratio of the *i* child is defined as follows:

$$AR_i = \max\left(h_i / w_i, w_i / h_i\right), \tag{1}$$

so that the average aspect ratio is defined as follows:

$$AAR = \frac{1}{n}\sum_{i=1}^{n} AR_i \cdot \tag{2}$$

By this definition, the aspect ratio of a rectangle is closer to 1. When the rectangle is closer to a square, the cognitive complexity for the user is lower.

Continuity (C) [4] is used to measure the search difficulty level of specific nodes in the treemap. Two adjacent nodes in a treemap are adjacent. If not, the user's search will be interrupted. Continuity can be described as the number of halts in the process of user search specific nodes. With the hierarchy of *n* nodes containing *n*-1 adjacent relationship, assuming the treemap visualization results contained the adjacent relation as *s*, the continuity can be defined as follows:

$$C = \frac{s}{n-1} \cdot \tag{3}$$

The continuous treemap is not interrupted during the search process, and the value is 1. Obviously, the greater the continuous is closer to 1, the lower the cognitive complexity is for the user.

Readability (R) [5] is used to measure the difficulty degree when using sequential search specific nodes in the treemap. It can be measured by the number of times the search direction of the line of sight changes for the user in the process of searching. Assuming that there are *n* nodes in a treemap, when traversing all the nodes, the line of sight can change *n*-2 times typically. In the specific calculation, first, calculate the center of the rectangle each node, and then, calculate the radian of the adjacent rectangular center wired in the sequential search. We defined that the line of sight direction of the user changes when the radian is greater than 0.1. For *s*, the number of the line of sight direction changes when the radian is greater than 0.1, so that readability can be defined as follows:

$$R = 1 - s/(n-2), n > 2 \cdot \tag{4}$$

Obviously, the higher the readability, the fewer times the line of sight jumps when a user searches the target node, and the lower the cognitive complexity is for the user. As shown in Fig. 5, the node sequence is 4, 3, 1, 10, 11, 1, 3 and 2; for sequence traversal, the radians of a_1, a_2, a_3, a_4 and a_5 are greater than 0.1. Therefore, the number of the line of sight changes of the user is 5, and the readability of this treemap can be calculated as follows:

$$R = 1 - 5/(8-2) \cdot \tag{5}$$

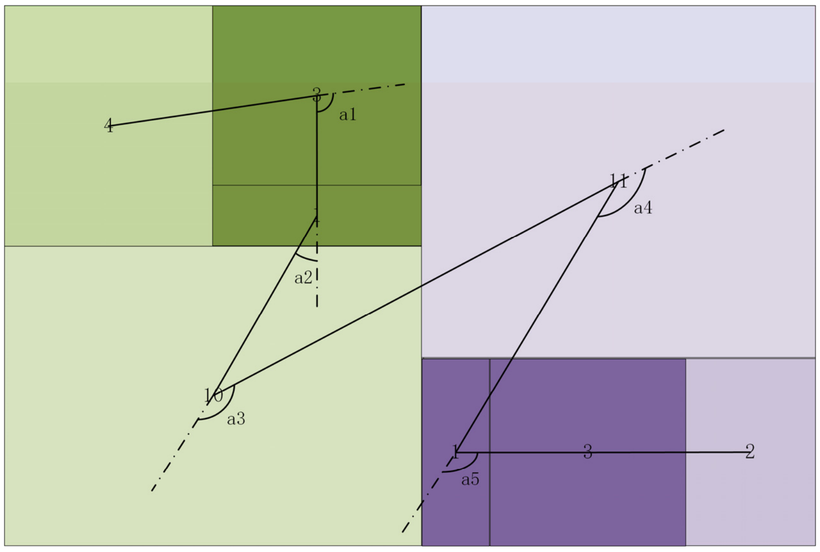


Fig. 5. Schematic diagram of readability for the line of sight change

Average distance change (ADC) [3] is used to describe the changes in the location and size of specific nodes when a dataset changes. Assume that rectangular node R_i is defined by a 4-tuple (x_i, y_i, w_i, h_i), where x and y are the coordinates of the upper left corner and the w and h are its width and height. After the data are updated, this node is given by R'_i (x'_i, y'_i, w'_i, h'_i). Then, the distance between R_i and R'_i is given by the following:

$$d_i = \sqrt{\left(x_i - x'_i\right)^2 + \left(y_i - y'_i\right)^2 + \left(w_i - w'_i\right)^2 + \left(h_i - h'_i\right)^2} \cdot \quad \textbf{(6)}$$

Therefore, the average distance change can be defined as follows:

$$ADC = 1 \Big/ \frac{1}{n} \sum_{i=1}^{n} d_i \cdot \quad \textbf{(7)}$$

Obviously, the greater the value of d_i is and the greater the change in the node is, the more unstable the visual effect is, and the higher the cognitive complexity is for the user. In this paper, we define ADC as the reciprocal of the mean value of the position change so that the more stability there is, the more stable the visualization is.

Variance of distance changes (VDC) [3] is used to measure the degree of equilibrium of stability of a treemap, and it can be represented as the variance of the rectangle position and shape change:

$$VDC = \sum_{i=1}^{n} \frac{\left(d_i - ADC\right)^2}{n} \cdot \quad \textbf{(8)}$$

The smaller the value of VDC is, the smaller the dispersion of the dataset is.

Distance correlation (DC) [6] is used to measure the linear correlation between the node positions and ordinal numbers in the treemap. The upper left coordinates of the rectangular nodes represent node coordinates, the upper left coordinates of the parent node are defined as (x, y), and the child is defined as (x_i, y_i). Thus, the Euclidean distance of the child node to the parent node can be defined as follows:

$$D_i = \sqrt{\left(x_i - x\right)^2 + \left(y_i - y\right)^2} \cdot \quad \textbf{(9)}$$

The distance correlation can be defined as follows:

$$DC = \frac{\sum_{i=1}^{n} i * D_i^2}{\sqrt{\sum_{i=1}^{n} i^2 * \sum_{i=1}^{n} \left(D_i^2\right)^2}} \cdot \quad \textbf{(10)}$$

DC has the range [0, 1]; the closer the value is to 1, the stronger the location of the rectangle node and its serial number is.

4.2 Experimental Description

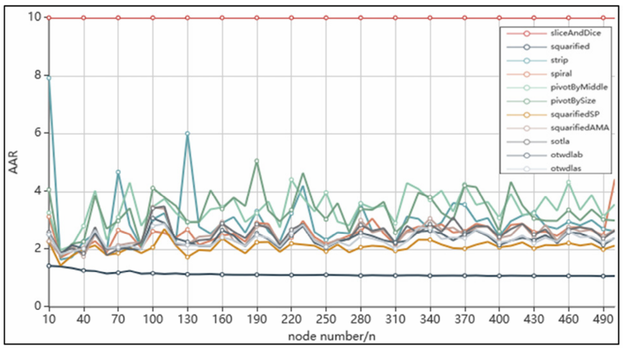
To verify the effectiveness of this algorithm, we realized the nine treemap layout algorithms of the previous algorithms and compared the experiment to evaluate the algorithm in this paper.

We selected datasets of different sizes to conduct the simulation experiments. The numbers of datasets were [10, 20, 30… 500], in a total 50 sets of sample data. Each datum in the dataset is generated by a random number generation function to give a value within the range [1, 100]. Two sets of data are required in the test of the ADC and VDC. This paper adopts the square root transformation method and takes up the entire original dataset with $O = \{a_1, a_2, ..., a_i, ..., a_n\}$, which will transform to the new dataset with $O' = \{a'_1, a'_2, ..., a'_i, ..., a'_n\}$. The transformation formula is $a'_i = \lceil \sqrt{a_i} \rceil$.

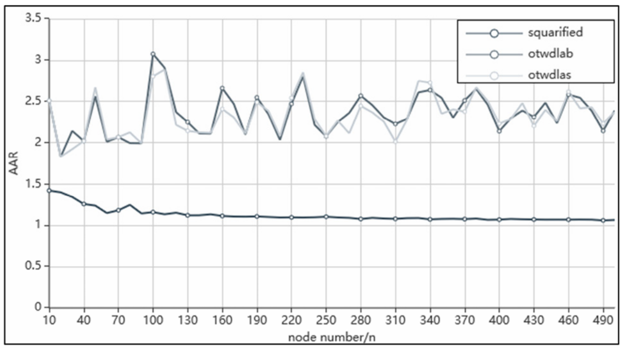
The experimental environment is a Windows 10 64-bit operating system, the processor Intel(R) Core(TM) i3 CPU M 380 @ 2.53 GHz, with 8.00 GB RAM.

4.3 Experimental Results and Analysis

We encoded the corresponding algorithm and performed a comparison experiment on six evaluation indexes. The results are shown in Fig. 6 to Fig. 11, the x-axis represents the count of data in the experimental dataset, and the y-axis represents the value of each evaluation index.



(a) The ARR of all algorithms

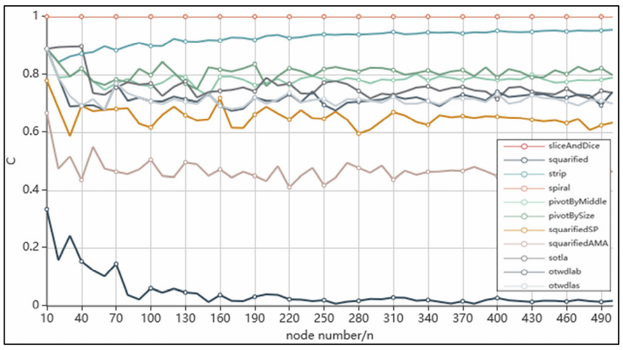


(b) The AAR of the OTWDLA and Squarifled algorithms

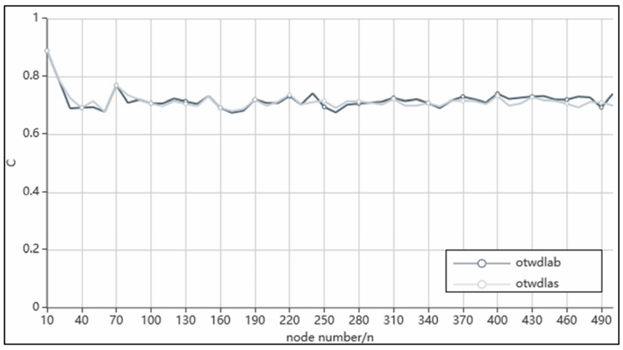
Fig. 6. The AAR evaluation index

As seen from Fig. 6(a), the algorithm with the best average aspect ratio is undoubtedly the Squarified algorithm, and the worst is the Slice-and-Dice algorithm; by setting the upper bounds of the aspect ratio as ten here, the algorithms can be compared, as shown in the diagram. The algorithm of this paper performed well on the average aspect ratio, second only to the Squarified-SP algorithm. Fig. 6(b) shows our algorithm and the Squarified algorithm. We can see that the AAR of our algorithm was maintained between 1.8 and 3.1 relatively well.

Regarding the continuity aspect, the closer the value is to 1, the better the performance is. As seen from Fig. 7(a), the Slice-and-Dice and Spiral are some of the best algorithms in this respect, where the continuity index is one. The Squarified algorithm is the worst in this aspect. The continuity of our algorithm was maintained between 0.7 and 0.9, and the performance is relatively good.



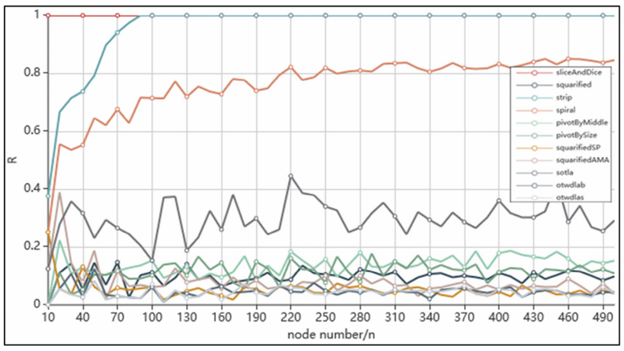
(a) The continuity all algorithms



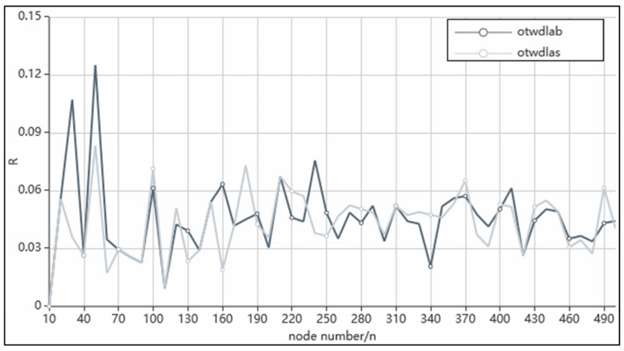
(b) The continuity of our OTWDLA

Fig. 7. The continuity evaluation index

Regarding readability, the closer the value is to 1, the better the performance is. As seen from Fig. 8(a), the readability of the Slice-and-Dice algorithm is the best, that of the Strip and Spiral is second, and the readability of the other algorithms is below 0.5. As seen from Fig. 8(b), regarding readability, the algorithm of this paper is not very good, with a value below 0.15.



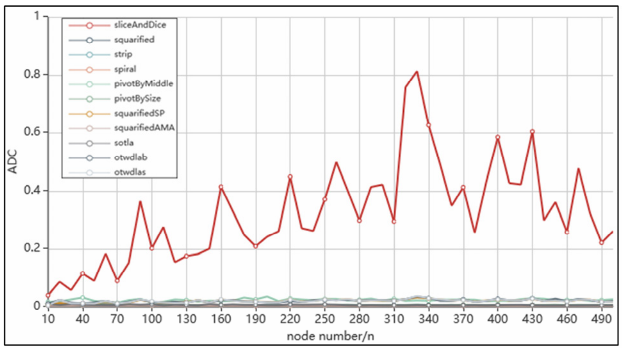
(a) The readability all algorithms



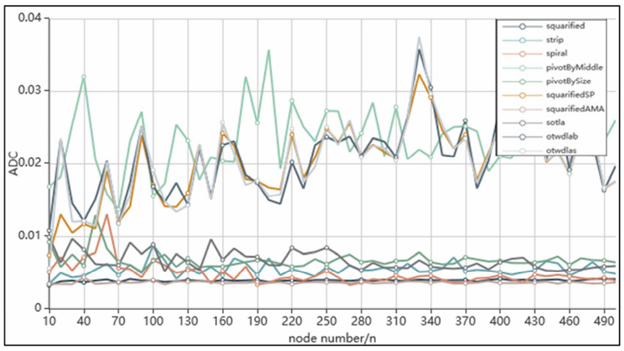
(b) The readability of OTWDLA

Fig. 8. The readability evaluation index

Regarding the average distance change, as seen from Fig. 9(a), the ADC of the Slice-and-Dice algorithm is best and is far better than that of the other algorithms. As shown in Fig. 9(b), other than the Slice-and-Dice algorithm, the algorithms are divided into two categories. Compared to the ADC of our algorithm, the pivot-by-middle and the Squarified-SP algorithm performances are relatively good. This result indicates that the average change is relatively small when the data are updated, which is more conducive to the long-term tracking of data.



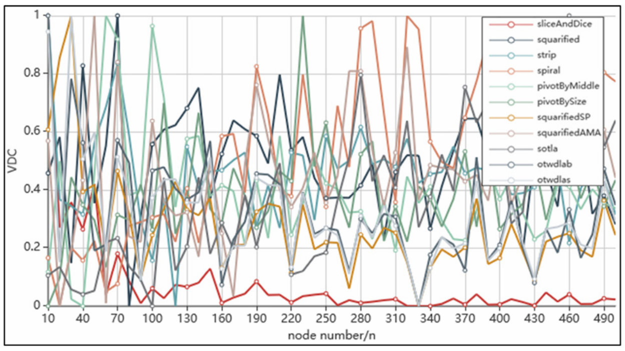
(a) The ADC of all algorithms



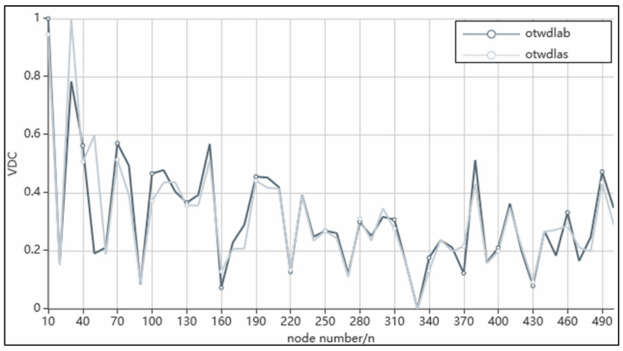
(b) The ADC of all algorithms other than Slice-and-DICE

Fig. 9. The ADC evaluation index

Regarding the variance of distance changes, the closer a value is to zero, the better the performance. As you can see from Fig. 10(a), the VDC of the Slice-and-Dice algorithm is best, while the other algorithm has a different degree of shock. In this paper, our algorithm and the Squarified-SP algorithm are better than the other algorithms in this aspect, when using smaller shocks.



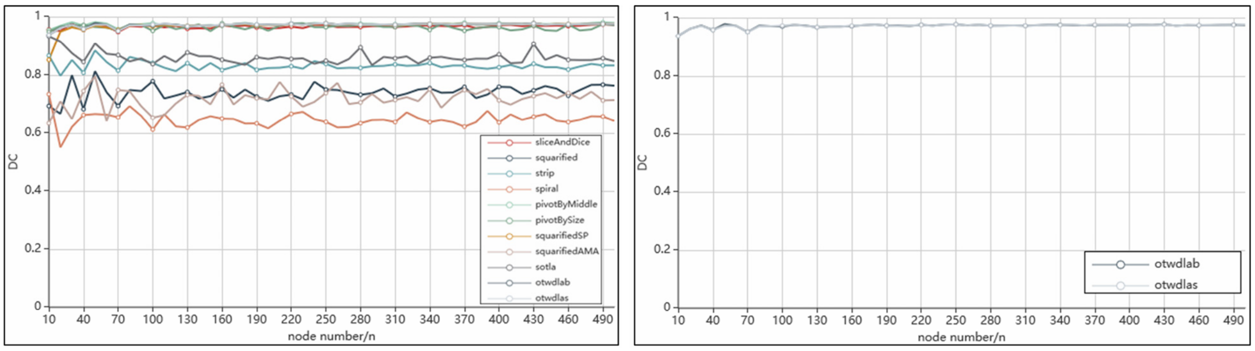
(a) The VDC of all algorithms



(b) The VDC of OTWDLA

Fig. 10. The VDC evaluation index

Regarding the distance correlation, as seen from Fig. 11, the algorithm in this paper performs well in this aspect, with values between 0.95 and 1, and the Slice-and-Dice, pivot-by-middle, pivot-by-size, and Squarified-SP algorithms are all at one level.



(a) The DC of all algorithms (b) The DC of OTWDLA

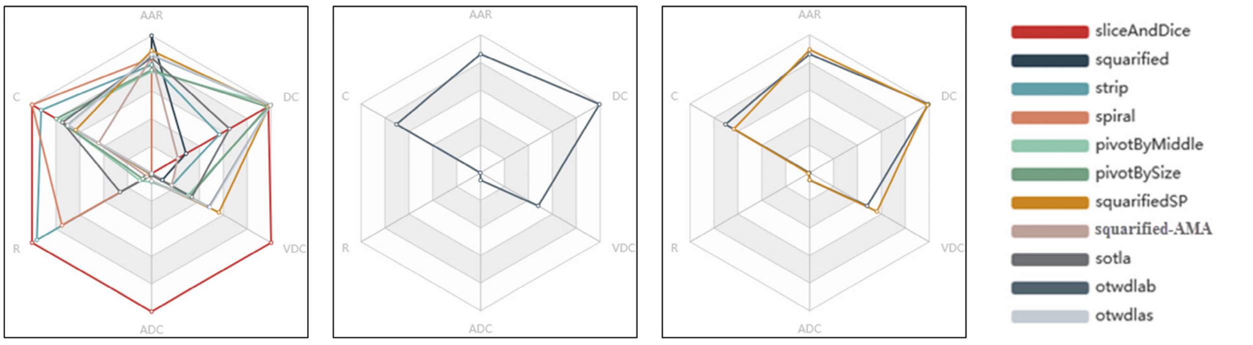
Fig. 11. The DC evaluation index

The experimental results of each evaluation index can be seen in Table 2.

Table 2. The value range of each evaluation indexes

Evaluation Index	AAR (1~10)	C (0~1)	R (0~1)	ADC	VDC (0~1)	DC (0~1)
Slice-and-Dice	10	1	1	0.040~0.814	0~1	0.947~0.970
Squarified	1.061~1.423	0.008~0.333	0~0.147	0.003~0.004	0~1	0.666~0.814
Strip	1.640~7.925	0.842~0.956	0.375~1	0.003~0.009	0~1	0.797~0.884
Spiral	1.708~4.420	1	0.250~0.850	0.003~0.013	0~1	0.550~0.734
Pivot-by-middle	1.965~4.406	0.725~0.889	0~0.222	0.014~0.036	0~1	0.954~0.981
Pivot-by-size	1.736~5.042	0.748~0.889	0~0.177	0.005~0.013	0~1	0.947~0.977
Squarified-SP	1.435~2.702	0.586~0.778	0.018~0.250	0.007~0.032	0~1	0.852~0.975
SOTLA	1.791~3.485	0.715~0.897	0.125~0.445	0.005~0.010	0~1	0.834~0.932
Squarified-AMA	1.748~3.451	0.411~0.667	0.017~0.389	0.003~0.004	0~1	0.634~0.796
OTWDLAB	1.828~3.076	0.675~0.889	0~0.125	0.011~0.036	0~1	0.937~0.978
OTWDLAS	1.828~2.887	0.675~0.889	0~0.083	0.006~0.038	0~1	0.937~0.978

The comparison and characteristics of each algorithm are analyzed from six different evaluation indexes. As shown in Fig. 12, after the values of the six evaluation indexes are normalized, the various evaluation indexes indicate that the greater the value of the index is, the better the is performance of the algorithm in the aspect being measured. That is, the larger the area of the entire corresponding algorithm is in the radar chart, the better the comprehensive performance is. Based on the comparative analysis of the algorithms, it can be seen that the Slice-and-Dice algorithm has the best comprehensive performance, but the aspect ratio is the worst, when applied to each of the layer nodes of the small datasets. The Squarified algorithm is undoubtedly the best algorithm in terms of the aspect ratio, but the other aspects of performance are not very good, are suitable for a large amount of data and do not need to clear the sequential dataset. In Fig. 12(b), our algorithm is very good in terms of the average aspect ratio, continuity, the distance correlation performance. Compared with the Strip and Spiral algorithm, the algorithm of this paper sacrifices a small amount of readability, improves the average distance change, variance of distance changes and distance correlation, and has obvious advantages in terms of the average aspect ratio and continuity. As shown in Fig. 12(c), the present algorithm was compared with the Squarified-SP algorithm and is similar in all aspects of performance, but the complexity of our algorithm is much simpler. Table 3 shows the performance evaluation of the indexes. In this paper, the largest advantage is that the average aspect ratio is obtained by using uncomplicated algorithm logic. At the same time, the continuity of performance is good and is suitable for continuity of sequential data visualization.



(a) Evaluation indexes of all algorithms (b) Evaluation index of OTWDLA (c) The evaluation indexes of OTWDLA and Squarified-SP (d) Legend

Fig. 12. Radar chart of evaluation indexes

Table 3. The evaluation indexes of our algorithms

Evaluation Index	AAR	C	R	ADC	VDC	DC
OTWDLAB	☆☆☆☆	☆☆☆	☆	☆☆	☆☆☆☆	☆☆☆☆☆
OTWDLAS	☆☆☆☆	☆☆☆	☆	☆☆	☆☆☆☆	☆☆☆☆☆

Note. ☆☆☆☆☆ represents the best, ☆☆☆☆ represents good, ☆☆☆ represent average, ☆☆ represents general, ☆ represents poor.

5 Conclusion

This paper proposed a new weighting divide orderly treemap layout algorithm (OTWDLA), and we analyze and study the visualization of hierarchical data. In this paper, by comparing many types of treemap layout algorithms, the experimental results show that the algorithm has high ratings for the average aspect ratio, correlation distance, continuity and variance of distance changes performance. This algorithm offers a useful compromise for obtaining good results in AAR and C. It is not the best in terms of comprehensive performance but has a very good effect on the orderly hierarchical data. The OTWDLA algorithm has the following features: (1) the algorithm has a very low complexity, it obtains a good average aspect ratio, and the rectangular shape is closer to the square, which is advantageous for human eye recognition; and (2) it has good continuity, which facilitates data searching. The OTWDLA can be used on data with a sequential requirement. Some examples include the following situations: Forest disaster statistics with time and space sequences of different periods in different locations; the fluctuation of stocks over time; changes in the click rate, transmission and transfer after a press release over time; and shopping website commodity sales data. At the same time, the implementation of this algorithm is very good for data without a sequential requirement. In the future, we will work to improve the ratio of their overall performance, optimize the square of the treemap, and enhance its readability.

Acknowledgements

The authors wish to thank the reviewers for their comments. This work is supported by the Fundamental Research Funds for the Central Universities (2015ZCQ-XX).

References

[1] W. Chen, Z. Shen, Y. Tao, Data Visualization, Publishing House of Electronics Industry, Beijing, China, 2013.

[2] B. Johnson, B. Shneiderman, Tree-maps: a space-filling approach to the visualization of hierarchical information structures, in: Proc. IEEE Conference on Visualization, 1991.

[3] B. Shneiderman, M. Wattenberg, Ordered treemap layouts, in: Proc. IEEE Symposium on Information Visualization, 2001.

[4] Y. Tu, H. W. Shen, Visualizing changes of hierarchical data using treemaps, IEEE Transactions on Visualization & Computer Graphics 13(2007) 1286-93.

[5] B.B. Bederson, B. Shneiderman, M. Wattenberg, Ordered and quantum treemaps: making effective use of 2D space to display hierarchies, ACM Transactions on Graphics 21(2002) 833-854.

[6] J. Wood, J. Dykes, Spatially ordered treemaps, IEEE Transactions on Visualization & Computer Graphics 14(2008) 1348-1355.

[7] Y. Chen, H. Hu, Z. Li, Performance compare and optimization of rectangular treemap layout algorithms, Journal of Computer-Aided Design & Computer Graphics 25(2013) 1623-1634.

[8] M. Bruls, Squarified Treemaps, in: W. C. de Leeuw, H. Pagendarm, F. H. Post, B. Waltzer (Eds.), Data Visualization, Springer-Verlag, Wien, Austria, 2000, pp. 33-42.

[9] T. Itoh, Y. Yamaguchi, Y. Ikehata, Y. Kajinaga, Hierarchical data visualization using a fast rectangle-packing algorithm, IEEE Transactions on Visualization & Computer Graphics 10(2004) 302-313.

[10] J.J. Van Wijk, d.W. Van, H., Cushion treemaps: visualization of hierarchical information, in: Proc. IEEE Symposium on Information Visualization (INFOVIS'99), 1999.

[11] M. Balzer, O. Deussen, C. Lewerentz, Voronoi treemaps for the visualization of software metrics, in: Proc. the 2005 ACM symposium on Software visualization, 2005.

[12] J. Liang, S. Simoff, Q.V. Nguyen, M.L. Huang, Visualizing large trees with divide & conquer partition, in: Proc. International Symposium on Visual Information Communication and Interaction, 2013.

[13] K. Wetzel, Pebbles-using circular treemaps to visualize disk usage. <http://lip.sourceforge.net/ctreemap.html>, 2003 (accessed 02.03).

[14] B. Parsia, T. Wang, J. Golbeck, Visualizing web ontologies with cropcircles, in: Proc. End User Semantic Web Interaction WS, 2005.

[15] W. Wang, H. Wang, G. Dai, H. Wang, Visualization of large hierarchical data by circle packing, in: Proc. Conference on Human Factors in Computing Systems, 2006.

[16] F. Fischer, J. Fuchs, F. Mansmann, ClockMap: enhancing circular treemaps with temporal glyphs for time-series data, in: Proc. Eurographics Conference on Visualization, 2012.

[17] H.S. Zhao, L. Lü, Z.T. Bo, Variational circular treemaps for hierarchical data, Journal of Software 27(2016) 1103-1113.

[18] S. Hahn, J. Döllner, Hybrid-treemap layouting, in: Proc. Eg/vgtc Conference on Visualization, 2017.

[19] T. Baudel, B. Broeksema, Capturing the design space of sequential space-filling layouts, IEEE Transactions on Visualization & Computer Graphics 18(2012) 2593-602.

[20] Z. Xin, X. Yuan, Treemap Visualization, Journal of Computer-Aided Design & Computer Graphics 24(2012) 1113-1124.

[21] Y. Chen, Y. Jia, Y. Sun, A squarified treemap layout algorithm using block-based sorting, Journal of Computer-Aided Design & Computer Graphics 25(2013) 731-737.

[22] H. Hu, Y. Chen, Y. Zhen, R. Liu, A squarified and ordered treemap layout algorithm, Journal of Computer-Aided Design & Computer Graphics 26(2014) 1703-1710.

[23] C. Zhang, X. Xiang, P. Zhang, Research and optimization of visual layout algorithm based on squarified treemap, Computer Engineering & Applications 53(9)(2017) 208-212.