Virtual Reality & Intelligent Hardware

• Article •

August 2022 Vol. 4 No. 4: 342–358
10.1016/j.vrih.2021.09.006

Balanced-partitioning treemapping method for digital hierarchical dataset

Cong FENG[1*], Minglun GONG[2], Oliver DEUSSEN[3]

1. *Department of Statistics and Information, Xiangtan Instiutet of Technology, Xiangtan* 411104*, China;*
2. *Memorial University of Newfoundland, Canada;*
3. *University of Konstanz, Germany*

Received 14 August 2021; **Revised** 21 August 2021; **Accepted** 24 September 2021

Abstract: Background The problem of visualizing a hierarchical dataset is an important and useful technique in many real-life situations. Folder systems, stock markets, and other hierarchical-related datasets can use this technique to better understand the structure and dynamic variation of the dataset. Traditional space-filling(square)-based methods have the advantages of compact space usage and node size as opposed to diagram-based methods. Space-filling-based methods have two main research directions: static and dynamic performance. **Methods** This study presented a treemapping method based on balanced partitioning that enables excellent aspect ratios in one variant, good temporal coherence for dynamic data in another, and in the third, a satisfactory compromise between these two aspects. To layout a treemap, all the children of a node were divided into two groups, which were then further divided until groups of single elements were reached. After this, these groups were combined to form a rectangle representing the parent node. This process was performed for each layer of the hierarchical dataset. For the first variant from the partitioning, the child elements were sorted and two groups, sized as equally as possible, were built from both big and small elements (size-balanced partition). This achieved satisfactory aspect ratios for the rectangles but less so temporal coherence (dynamic). For the second variant, the sequence of children was taken and from this, groups, sized as equally as possible, were created without the need for sorting (sequence-based, good compromise between aspect ratio and temporal coherency). For the third variant, the children were split into two groups of equal cardinalities, regardless of their size (number-balanced, worse aspect ratios but good temporal coherence). **Results** This study evaluated the aspect ratios and dynamic stability of the employed methods and proposed a new metric that measures the visual difference between rectangles during their movement to represent temporally changing inputs. **Conclusion** This study demonstrated that the proposed method of treemapping via balanced partitioning outperformed the state-of-the-art methods for several real-world datasets.

Keywords: Information visualization; Treemapping; Balanced partitioning

Citation: Cong FENG, Minglun GONG, Oliver DEUSSEN. Balanced-partitioning treemapping method for digital hierarchical dataset. Virtual Reality & Intelligent Hardware, 2022, 4(4): 342–358

*Corresponding author, 287956404@qq.com

1 Introduction

Techniques for visualizing hierarchical datasets have many important applications. Hierarchies can be more naturally represented using a tree structure in which branches are used to connect data at different hierarchical levels.

When different leaf nodes have different values, the tree structure is limited in that it shows the connections between nodes but not their values. Therefore, area-based methods are often used where the hierarchy is encoded by inclusion and the values are visualized by areas. The sum of the values from the child nodes defines the value of their parent in the hierarchy.

The corresponding area-based methods fall into two categories, namely top-down and bottom-up approaches. Top-down approaches first create an area as the representation of the root node, and then recursively arrange children of the root node into this area[1], whereas bottom-up approaches first make use of a layout algorithm to arrange the representation of all leaf nodes into an area and then recursively pack these representations into larger areas, depending on the input given hierarchy[2].

Rectangles are the most widely used shapes for representing nodes in area-based methods, where the areas of the rectangles correspond to the values of the nodes. Rectangle-based treemapping methods create compact structures that are efficient for computation and easy to read, making them popular for visualizing hierarchical datasets, especially ones that are large and dynamic.

Given that all elements of the hierarchy are rectangles, visually clear structures can be created in a variety of ways. These advantages allow rectangle-based treemapping methods to become popular tools for visualizing hierarchical datasets.

Given the same dataset, there are many ways to generate the corresponding rectangle treemaps. Two factors have an important impact on the quality of the treemap results. One is the visibility and aspect ratio of individual nodes, whereas the other is dynamic stability.

The proposed method improved both the factors using a top-down approach. For each layer of the treemap, we subdivided the children of each node into two groups, which were then subdivided until groups of single elements were reached. In two variants of the algorithm, the grouping was based on the individual sizes of each child and therefore, an instance of the balanced partitioning problem (e.g.,[3]). Both groups were represented by two rectangles arranged to partition the father rectangle. As the recursive division optimized the aspect ratio at each step, this strategy allowed the aspect ratio of all elements to be kept in a better range than previous methods.

The balanced partitioning problem is an NP-hard problem but several approximations exist. One of these is a greedy strategy in which elements can be sorted according to their size and where both bins are filled alternatively so that they always have a similar size. While this achieves good results, the variation between the two groups is maximized and this is not ideal for aspect ratios. Therefore, this study used another group assignment strategy whereby all big elements were grouped into one group and all small elements in another (size-balanced splitting). This created satisfactory aspect ratios for the treemap as well as inconsistencies in the dynamic case when the sizes of elements and the corresponding order changed over time.

Thus, the study developed two other variants of the method to better maintain temporal coherency. The most stable result is obtained if the data are split into two groups of equal cardinalities, regardless of element sizes (number-balanced splitting). A good compromise between temporal stability and aspect ratio is reached when the sequence of children is not sorted, and then two groups, sized as equally as possible, are created using the greedy algorithm (sequence-balanced splitting).

To evaluate the quality of the treemaps, this study used several standard datasets as well as an extremely varying synthetic dataset. The treemaps were numerically analyzed using both established measures[4] as well

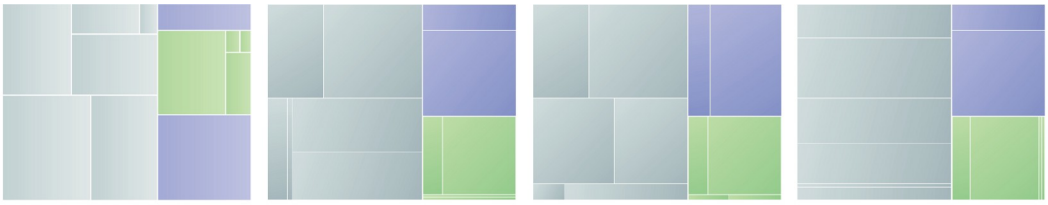


Figure 1 Our treemapping that relies on balanced partitioning is extremely easy to implement. One instance of our size-balanced partitioning (a) Shows a better aspect ratio compared to other methods, e.g., (b) Spiral placement, (c) Strips, (d) Slice and Dice.

as a new measure that takes the movement and shape change of a rectangle into account at the same time.

The evaluation showed that the size-balanced splitting (Figure 1a) achieved excellent aspect ratios with a very simple algorithm, while the number-balanced version was as stable as the state-of-the-art solutions, more commonly used within the field. The sequence-balanced version of the algorithm created treemaps with an ideal composition between both properties and was thus recommended as a strong choice for general-purpose treemapping.

The contributions of this paper hence include:

- Size-balanced partition maintains near-square aspect ratios of rectangles in a treemap such that small elements are better readable.
- A sequence-balanced partition better preserves the dynamic stability while maintaining a good aspect ratio.
- A number-balanced partition that only focuses on the number of elements is the most stable for highly temporally dynamic data.
- A new metric with respect to visual change complements existing dynamic stability measurements, which usually only considers distance and size changes.

As shown in Figure 2, given a sequence of input data belonging to the same hierarchical level, performing a number-balanced partition (Figure 2a) ensures that the number of nodes in the subsets is balanced and that the temporal coherence is well preserved. However, because the total values of the two subsets vary dramatically (8+9 vs. 1+2), the corresponding rectangles have poor aspect ratios. A greedy searching algorithm for the size-balanced partition problem (Figure 2b) sorts all nodes by their sizes in descending order and assigns large nodes first. Even though the two subsets at the first level are perfectly balanced in size, the split at the second level cannot be balanced because of node size variation within the subsets. Like the greedy strategy, the proposed size-balanced partition (Figure 2c) also orders the input nodes according to their sizes, however, instead of alternatively assigning nodes to the two subsets, the small nodes are kept in the same subset and thus reduce size variation within subsets. This allows for balanced partitions at all hierarchical levels. When there is a need to maintain the order of nodes in the input sequence, the proposed sequence-balanced partition (Figure 2d) divides the sequence directly without attempting to achieve better temporal coherence when handling dynamic datasets.

The remainder of this paper proceeds as follows: after reviewing related work, Section 3 demonstrates the proposed method in theory and describes the alternatives for an efficient computation of treemaps, whilst Section 4 presents the new quality measurement for treemapping methods, provides results on three datasets and compares them with related works. Section 5 draws a conclusion and identifies potential opportunities for future work.

2 Related works

Shneiderman[1] proposed the first space-filling-based method for visualizing a hierarchical dataset. This

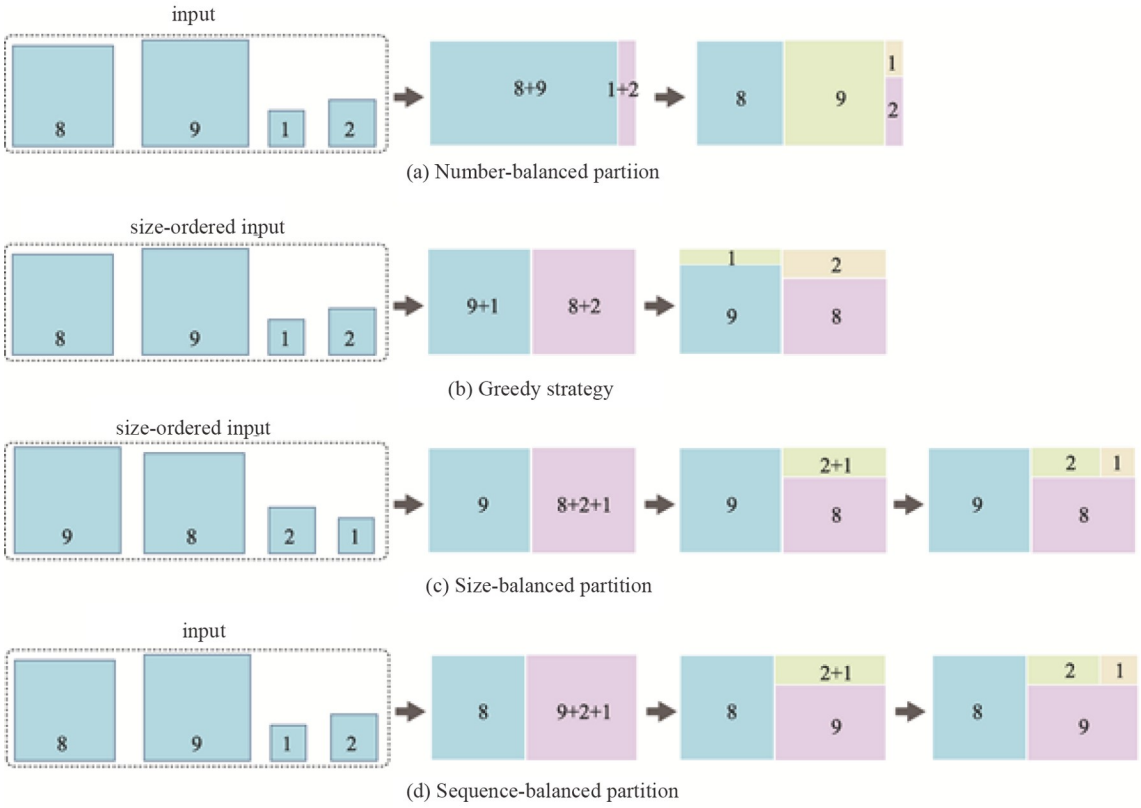


Figure 2 Comparison among different partitioning approaches.

method subdivides the rectangle for the entire representation into smaller rectangles by alternatively splitting the rectangle horizontally and vertically. If a node has many children, many thin rectangles will appear.

Bruls et al. introduced squarified treemaps with the aim of maintaining good aspect ratios for each subdivided rectangle[5]. This method first orders children from large to small after which elements are added successively as rectangles with the aspect ratio being optimized at each step. If the rectangles are too thin, the filling order is changed and thus, the input order is destroyed, and for temporally changing inputs, large rearrangements might occur. Bederson et al. refined squarified treemaps using the strip method, which maintained the order of the input[6]. The child elements are arranged top down and left to right in a sequence of strips. Tu and Shen proposed a spiral-based method to maintain the input sequence during treemapping[7]. Both methods arrange elements linearly, thus creating rectangles with large or small aspect ratios.

Duarte et al. proposed a neighborhood preserving method to maintain the similarity of the input data[8]. This method involves laying out the input data in a rectangle in which a slice line is then drawn and subsequently moved by means of scaling to a place that maintains the ratio of its two parts. While this method enables good preservation of neighborhoods, it does not offer sequence preservation or a dynamically stable result. Lu & Fogarty proposed cascaded treemaps to visualize child dependencies within a treemap using cascades of elements[9]. There is a trade-off between showing the hierarchical structure and space filling.

Many treemap approaches change their appearance significantly when there is a change in the size of the data. Tak & Cockburn proposed a location-drift metric to show the dynamic stability of a mapping method, using the variation in the center of a moving element over time[10]. This metric is particularly useful for comparing local and global movements within treemaps. The authors also introduced Hilbert and Moore curve-based treemaps, which can be used to obtain spatial stability. The addition of extra space to the visualization is key to dynamic stability. This was done by Itoh et al.[2] who used rectangle packing to arrange

child elements in a top-down manner. Sondag et al. presented a dynamic method that updates the rectangles of a treemap from the last time step within a sequence by stretching and flipping neighboring rectangles[4]. This scheme can be combined with different treemaps but may be considered as being too complex to be used for large-scale datasets. More rectangle-based treemapping methods can be found in an online survey[11].

In addition to rectangles, many other forms of treemaps exist. Wang et al. used nested circles to visualize the hierarchical data, packing circles, level by level, according to a top-down sequence[12]. Zhao et al. used a continuous packing method to generate a circular-based treemap and presented a fisheye distortion-based interactive technique for zooming[13]. Fischer et al. attributed circular treemaps to glyphs to visualize time-series data[14]. Balzer et al. proposed Voronoi treemaps using Lloyd's method to compute a centroidal Voronoi diagram using the element weight to control cell sizes[15]. Although this method offers a better aspect ratio than slice-based methods, it involves a much higher computational cost. Görtler et al. presented bubble treemaps that use bubble outlines to pack circles, adding transparency or jaggedness of the outlines to visualize data uncertainty, thus making it more compact than circular packing[16]. Vernier et al. proposed two metrics for comparing the performance of different treemap methods: the first is to normalize the aspect ratio to one whereas the second aims at the rectangular position and shape changes over a time series[17]. A recent survey and classification of different treemap methods can be found in Scheibel et al[18]. Li et al. proposed a barcode treemap method that uses rectangular bars to show the changes in a hierarchical dataset where the width of each bar indicates the node level[19]. An advantage of this method is that it can visualize several variations in a tree structure at the same time. Finally, Scheibel et al. used an initial layout onto which all changes in the dataset were mapped[19].

3 Methods

The proposed algorithm uses a hierarchical dataset as the input and generates the corresponding treemap. First, a rectangular area is created with a default aspect ratio of 1.25 to form the root node that holds the whole dataset, with the data elements at the top hierarchical level being treated as the children of this node. The root rectangle is then split into multiple smaller rectangles, one for each child, after which its area is determined based on the value of the corresponding data element. When a child rectangle contains more than one data element in the hierarchical dataset, it is split further under the same criteria. The recursive splitting process (Figure 3) continues until each rectangle represents a single data element, which is referred to as a leaf rectangle.

The following section explains how to split a given parent rectangle into child rectangles as well as how to split all elements into subgroups based on their size, input sequence, and input location in the sequence.

3.1 Balanced partition of a given node

Given a set of data elements belonging to the same parent node, the objective of this study was to find an appropriate and efficient way to split the parent rectangle into child rectangles, such that: (1) the union of child rectangles fills the parent rectangle; (2) no two child rectangles intersect; (3) the area of each child rectangle is proportional to the value of the corresponding data element; and (4) the shape of each child rectangle is as close to a square as possible.

To achieve these goals, a recursive partition procedure was applied. Set L was split into two subsets, namely L_1 and L_2, each time after which the parent rectangle was cut into two child rectangles with areas being proportional to values in L_1 and L_2. To ensure that the child rectangles were as close to a square shape as possible, the parent rectangle was always cut along its shorter edges. The values of subsets L_1 and L_2 were also balanced so that the two child rectangles had similar areas.

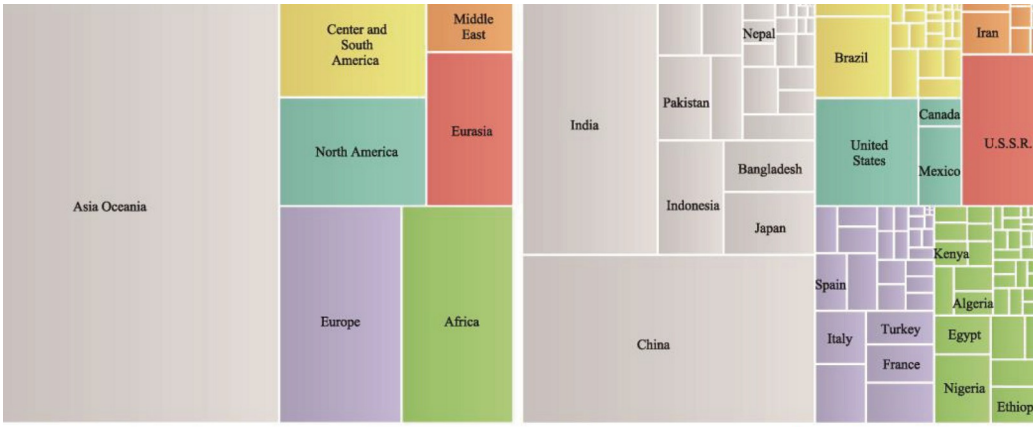


Figure 3 An example of the recursive splitting process: a root rectangle is split first in (a) by continents, and then each continent is split in (b) by countries.

As previously mentioned, splitting a list of numbers (sizes of elements) into two subsets so that the difference between the total values (sum of element sizes) in the two subsets is minimized is a well-known problem in computing and is referred to as the balanced partitioning problem[3]. As a NP-hard problem, several algorithms have been proposed to achieve near-optimal solutions[20]. Commonly used here is the simply greedy algorithm, which sorts all numbers in descending order so that larger numbers are processed first after which the next largest numbers are assigned to the subset with a smaller total value. The pseudocode is presented in Algorithm 1.

Algorithm 1 Greedy solution for balanced partition

```
function SPLIT(L)
 L1 = {};
 L2 = {};
 while L != {} do
x = L.largestElement();
L.remove(x);
if sum(L1) < sum(L2) then
 L1.append(x);
else
 L2.append(x);
end if
 end while
 return L1; L2;
end function
```

While the above greedy algorithm can generally balance the size of the two subsets, further partitioning each subset often leads to unbalanced subtrees (Figure 2b) as the algorithm tends to alternatively assign data elements to the two subsets, resulting in high variance among data values within each set. Assume that the following sequence of element sizes is given: [2,15,20,21]. The partitioning would split this list into two lists: [20,21] and [2,21], which creates a perfect split of the initial rectangle, but two poor splits for the subsequent lists.

To address this problem, the strategy involved splitting L into subsets such that: (1) the intra-variance within each subset was minimized, and (2) the total values for L_1 and L_2 were as close as possible. A simple heuristic for achieving this goal was to group large elements in L_1 and small elements in L_2, so that both subsets had a

low intra-variance. To balance the total value in L_1 and L_2, L_1=∅ and L_2=L were initialized after which the next largest element in L_2 was gradually moved to L_1 until such a move increased the total value difference between the two sets. The pseudocode for this splitting method is presented in Algorithm 2.

Algorithm 2 Variance-minimizing solution for balanced partition

```
function SPLIT(L)
 h = sum(L)/2;
 L1 = {};
 L2 = L;
 x = L2.largestElement();
 while (abs(sum(L1)-h) > abs(sum(L1+x)-h)) do
L1.append(x);
L2.remove(x);
x = L2.largestElement();
 end while
 return L1; L2;
end function
```

This method (Figure 2c) has several advantages in terms of filling the parent rectangle: (1) it is efficient to compute and has a small storage footprint, (2) it is easy to implement, and (3) compared to other methods, such as the unslice and curve-segmentation method[21], it maintains a similar rectangular structure for all child nodes. For further details, see Figure 4.

It should be noted that although a similar idea was used to obtain an as-equal-as-possible partition of the rectangle, this method is different from the pivot by the split size method[6]. The split size method aims to maintain the order of the input sequence within each rectangle and thus involves laying out the two parts of a rectangle according to the sequence order. In contrast, the method used employed by this study recursively divided the element lists until the parts consisted of only a single element. For each pair of child lists, the corresponding rectangle was divided into two parts, creating aspect ratios closer to the square, as shown below.

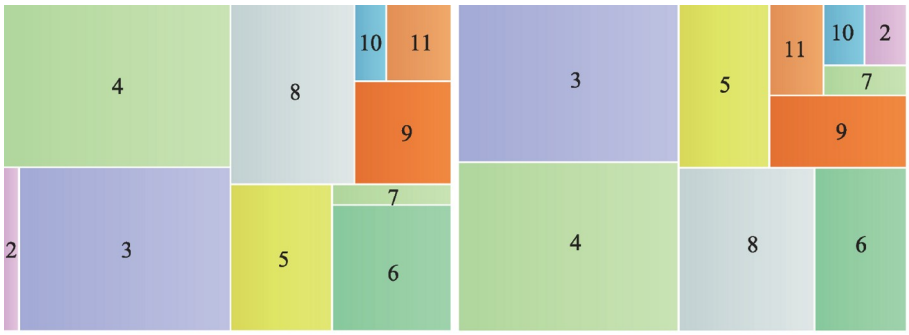
3.2 Sequence-balanced partitioning

The above partitioning algorithm focuses on creating near-square shapes for all elements in the treemap. Owing to the sorting of the elements when splitting each parent node, small changes in the values of elements could affect the sorting result and thus lead to a range of treemaps. Therefore, the temporal coherence among the treemaps may be poor when visualizing a dynamically changing dataset.

A particular advantage of the proposed variance-minimizing solution for balanced splitting is that it also works for sequences that are not sorted. Even though the size difference between the two subgroups may be larger than that of the original algorithm, this solution obtains two approximately equal groups that can be used for dividing the parent rectangle, resulting in a sequence-balanced partition (Figure 2d). Although the aspect ratios may be worse, the sizes of the elements change over time and thus the sequence would not be reordered and temporal stability could be maintained.

3.3 Number-balanced partitioning

The structure of the treemap will only change if, due to changes in element sizes, the splitting must be performed differently. To avoid this type of structural change, this study proposed a third variant of this method. Here, each parent node was, without sorting, successively divided into two groups with equal numbers (cardinality) of elements. It should be noted that the main drawback of this number-balanced partition (Figure 2a) is that it yields a higher variation in aspect ratios and thus results in the reduced visibility and



(a) Greedy strategy(2.88) (b) Size-balanced partition(1.69)

Figure 4 Comparison between greedy algorithm (a) and the size-balanced partition (b). Leaf nodes in this study's results were found to have better aspect ratios, with an average of 1.69 vs. 2.88 by greedy.

esthetic appearance of the small rectangles. However, the method is ultimately advantageous in that it produces a fully dynamic stable treemap result with each subdivision being determined according to the number of input sequences thus allowing for the positions of all elements to be determined continuously.

4 Evaluation and results

The method was tested on several widely used datasets, namely Coffee, Name, and Population, all of which varied over time but not dramatically. Therefore, this study involved the creation of a synthetic dataset with 200 elements over 20 frames in a two-level hierarchical structure. The size of each element varied from less than one to more than 9000. Figure 5 shows the treemapping results compared to the squarified treemaps and incremental methods.

The quality of the different methods was evaluated conducted using two measures. The first involved the readability of the rectangles of a static treemap, which is typically measured using the average mean and the medium aspect ratio[4]. The second involved the stability of different methods for dynamic data where changes in the positions of the child elements were measured as well as any changes in size. This measurement was amended using a method that also measured changes in elements shapes.

4.1 Evaluation of static quality

The readability of the rectangular elements of the treemaps were measured used their average and median aspect ratios. For a parent element with n children where the aspect ratios for each child are denoted as ar_1, ar_2, …, ar_n, the average aspect ratio is computed as $\frac{1}{n}\sum_{i=1}^{n} ar_i$.

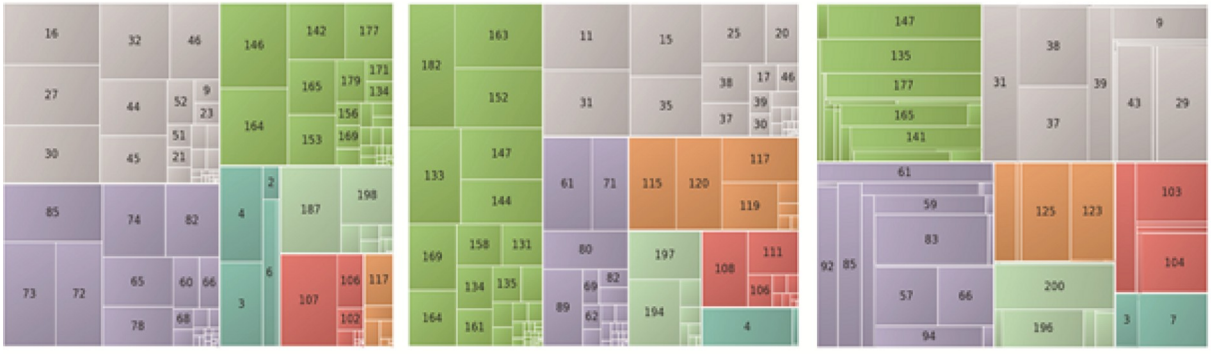
An alternative for measuring the average aspect ratio is as follows: if each child rectangle has a size of w_1, w_2, …, w_n and the size of the parent rectangle is w_sum, then the weighted average aspect ratio is defined by the following weighted sum: $\frac{1}{n}\sum_{i=1}^{n} ar_i \times w_i / \text{w sum}$.

The weighted average aspect ratio has the advantage of reducing the contribution of extremely small elements with large aspect ratios. In the following, the weighted average aspect ratio remained unused as small elements with large aspect ratios often disturb the appearance of a treemap and thus hinder the readability of the content. Small objects should thus be judged as big objects, or one could also add a weighting term between large and small elements.

If several treemaps are given from various data sets or from a dynamically varying single set, the aspect ratio values can be further processed by computing their average and median values. These averaged aspect ratios are referred to as the average mean aspect ratio and average median aspect ratio.

(a) Number-balanced treemaps

(b) Squarified treemaps



(c) Incremental treemaps

Figure 5 Three time steps of a synthetic dataset: (a) result of the number-balanced method, (b) squarified treemaps, and (c) incremental treemaps. Rectangles with the same parent root share the same color.

4.2 Evaluation of dynamic quality

A hierarchical dataset can change over time. For example, in a file, system files might be modified, added, or deleted. To obtain a treemap visualization that reflects such dynamic sets, a treemap can be constructed for each time step but this often results in an incoherent visualization. Another possibility is to update the initial treemap for every time step (incremental design). Both ways have been gone, we will compare our method, which belongs to the first category, with state-of-the-art methods of both kinds.

The distance change[22] was a technique first proposed by Shneiderman and Wattenberg to assess the stability of a treemapping method with a dynamic dataset by measuring the change in the position and size of a rectangle over time. The variance of the distance change can also be used to measure the stability around the average change in distances.

Some methods implement a location-drift metric[10], which focuses on the position change over an entire period of time rather than just between two time steps. The elements of a divided rectangle that move only in a local area should have a lower location drift value than a rectangle with global movement. Sondag et al.[4] used

a dynamic metric. They claim that in a dynamic dataset, the stability of a treemapping method depends on the neighbors of a segmented rectangle. If a segmented rectangle moves significantly at different time steps, but its neighbors remain stable, this rectangle will receive a low stability value. Both metrics are meaningful from a perceptual perspective.

In real life, a dataset in a hierarchical structure can be seen as a sequence-like system and it is for this reason that readability[6] is proposed here as a measurement of the sequential order of a treemapping result. The readability metric calculates the number of required changes for a reader's eye scanning direction corresponding to the input sequence. To optimize this, strip- and spiral-based treemapping layout methods have been proposed in [6] and [7], respectively.

These metrics are useful to some degree but, unfortunately, do not fully capture the dynamic stability of treemaps. For example, these metrics all focus on distance changes but ignore shape variations over time. For a dynamic dataset, the change between two rectangles relies not only on its position differences but also on its visual change. The visual change between two rectangles is shown in Figure 6 below. The two rectangles were aligned with the same center after which the difference between their areas was computed as dashed. This was computed using the following equation:

$$\text{visual change} = s_1 + s_2 + s_3 + s_4 0 \tag{1}$$

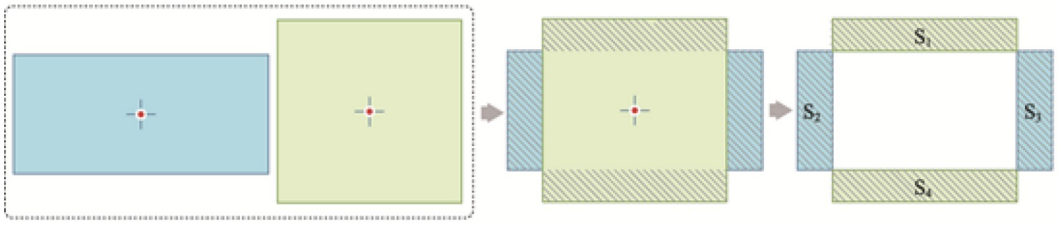


Figure 6 The visual change between two rectangles is measured by their difference in-between their areas (dashed).

To obtain a value for the complete treemap, the *visual change* of its elements was normalized by the change in the dataset size. The absolute size of the changes at two time steps was not considered, however, the size change for each leaf node was accumulated. Thus, the study proposed to measure the stability of a treemapping method as follows: Given a hierarchical dataset with n elements, we specify how much the hierarchical dataset changes at two time steps and normalize these changes according to the accumulated change of the leaf nodes. For example, in the distance change metric, we normalize the distance change metric as follows:

$$\sum_{j=1}^{n} (size(j)_{i+1} - size(j)_i) \tag{2}$$

where i and i+1 are two consequent time steps, and $size(j)$ denotes the size of the jth rectangle in the treemap. The reasoning behind this normalization is that the changes in a treemap closely follow the changes in the dataset.

In this study, both center movement and visual change were used to evaluate the stability of treemapping methods for a dynamic dataset. Both metrics were normalized by the size of the input data (see Equation (2)).

4.3 Results

In all histograms, lower values for visual change and center movement were preferred. For an aspect ratio of 1 to 1.5, was preferred.

The results using the Coffee and Name datasets are shown in [4]. Eleven treemapping methods were tested using these two datasets. A population dataset downloaded from the Internet was also tested. The Coffee dataset contains 85 countries with the amount of coffee imported from 1994 to 2014, with the hierarchy of the coffee dataset presenting three levels. Countries were considered as leaf nodes and were grouped into con-

tinents, such as North America, West Europe, and East Asia. The highest level of the hierarchy includes America, Europe, and Asia.

The Name dataset contains 62 popular names in the Netherlands from 1993 to 2015. This dataset has only one level and the difference between names is smaller than those found in the Coffee dataset. The population dataset includes more than 200 countries from 1980 to 2010 and presents two layers. The top layer is the region layer, which contains Asian Oceania, Europe, North America, Central and South America, the Middle East, and Africa whereas the lower layer contains all countries. Although this is not a deep hierarchical dataset, new elements appear for some time steps, making it useful for dynamic considerations.

The proposed methods were tested using the aforementioned static and dynamic metrics and compared against the existing methods identified above. More specifically, the average and median aspect ratios were used as static metrics and the average center movement and average visual change as dynamic metrics. The average center movement and average visual change were normalized by the global change in the size of the hierarchical datasets.

For the Coffee dataset, the aspect ratio (Figure 7a) varied significantly for the different methods. The size-balanced method received better results than most other methods with the average median aspect ratio being better than that of all other methods. The sequence-balanced method had a better average median aspect ratio than the existing methods, with the average aspect ratio also being better than that of Moore, Spiral, Pivot, Slice and Dice. The number-balanced method did not work well for the average aspect ratio, however, it presented a good performance for the median aspect ratio. For the Coffee dataset, the difference between the different countries was significant. Therefore, many methods have a large average aspect ratio, for example, larger than 100. For this challenging dataset, the proposed method presented good static measurements.

Figure 7b shows the dynamic metric results of the Coffee dataset. The size-balanced and sequence-balanced methods presented a better average center movement than most methods, except for Slice and Dice. For the average visual change, the Slice and Dice technique presented the best stability. The value of the sequence-balanced method was only slightly higher than that of the incremental method and was higher than that for Slice and Dice. Thus, the sequence-balanced method offers good dynamic stability for the coffee dataset. The value of the size-balanced method for the average visual change was in the middle of all these methods with the difference between the size-balanced method and the better methods being less than 1E-07. Hence, this method also performed well with respect to visual changes. The number-balanced method performed well in terms of the dynamic properties, presenting the best average visual change, with its average center movement being only higher than that of the size-balanced, sequence-balanced, and Slice and Dice methods. The results of this in comparison to the other methods can be seen in Figure 8. For the Coffee dataset, the methods performed well both in terms of visualization and stability as many small rectangles can be seen and the dynamic variations can be easily traced in the dataset.

For the Name dataset in Figure 7c, all methods, including the proposed, presented a good static stability except for that of spiral and Slice and Dice. The values of the size-balanced and sequence-balanced methods were approximately 1.5, which was competitive with the incremental, Hibert, Moore, pivot by size, simplified, and strip methods. The number-balanced method also performed well in the dataset with the average and median aspect ratios being approximately 1.8. Many methods perform well on this dataset, as they had only one level and the difference between elements was small.

For a more dynamic name dataset in Figure 7d, Slice and Dice still presented the best dynamic stability among all the methods. The size-balanced and sequence-balanced methods had higher values in the average center movement than both the incremental and Slice and Dice treemaps and were competitive with the spiral and pivot by middle methods. These methods did not perform better than the other methods for the average visual change, however, the number-balanced method performed well for both average center movement and

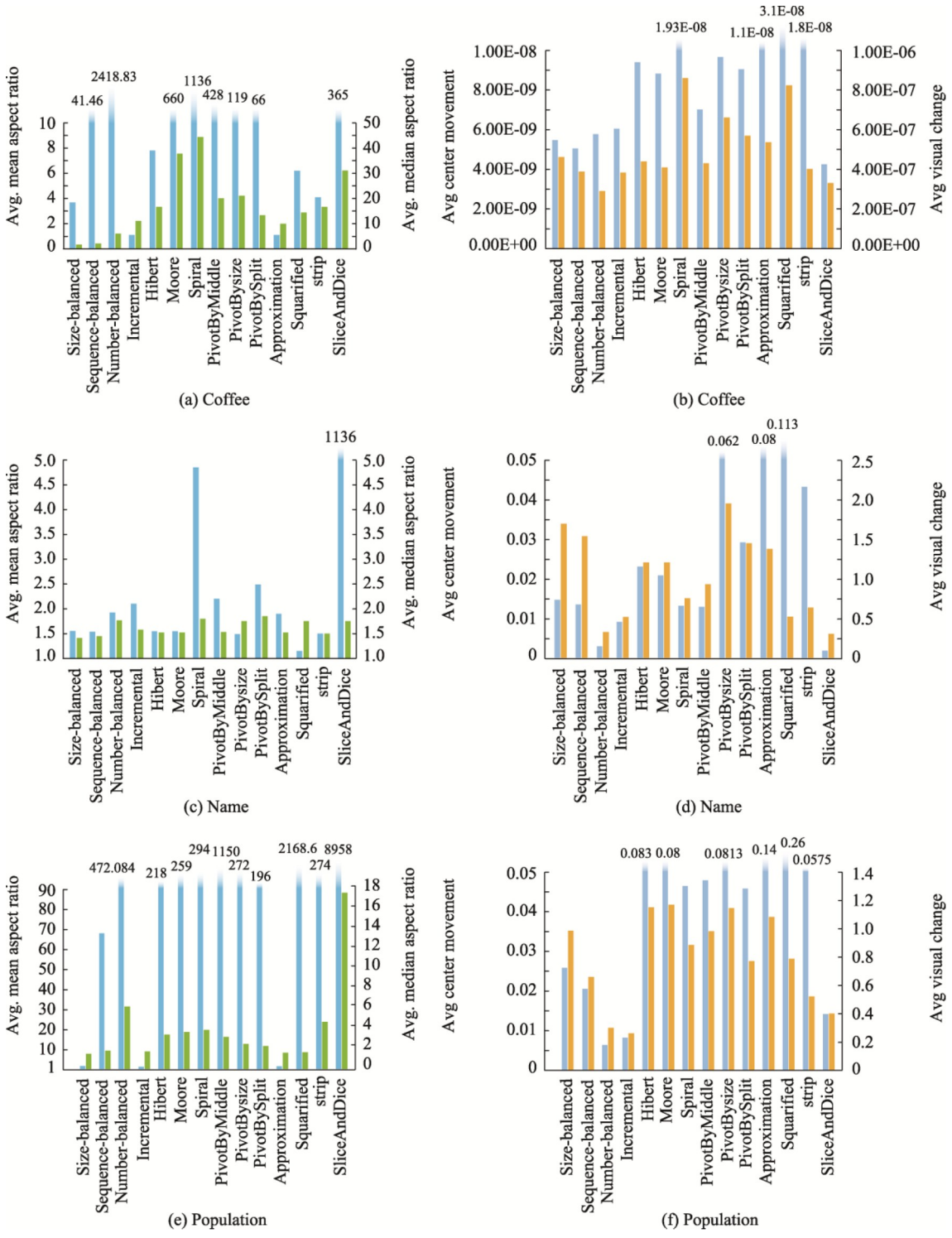


Figure 7 **This study computed the average mean aspect ratios and average median aspect ratios of three datasets (Coffee, Name, Population), which are presented on the left, and their average center movements and visual changes, which are presented on the right.**

average visual change with the result value being only slightly higher than that of the Slice and Dice method. The results for this set for three continuous time steps are shown in Figure 10. The number-balanced and sequence-balanced methods were found to have preserved the sequences well.

For the Population dataset, the proposed methods offer ideal aspect ratios, as shown in Figure 7e, performing better than all other methods except for the incremental and approximation methods. While the size-balanced method presented a slightly higher average mean aspect ratio than these two methods, it also presented a lower

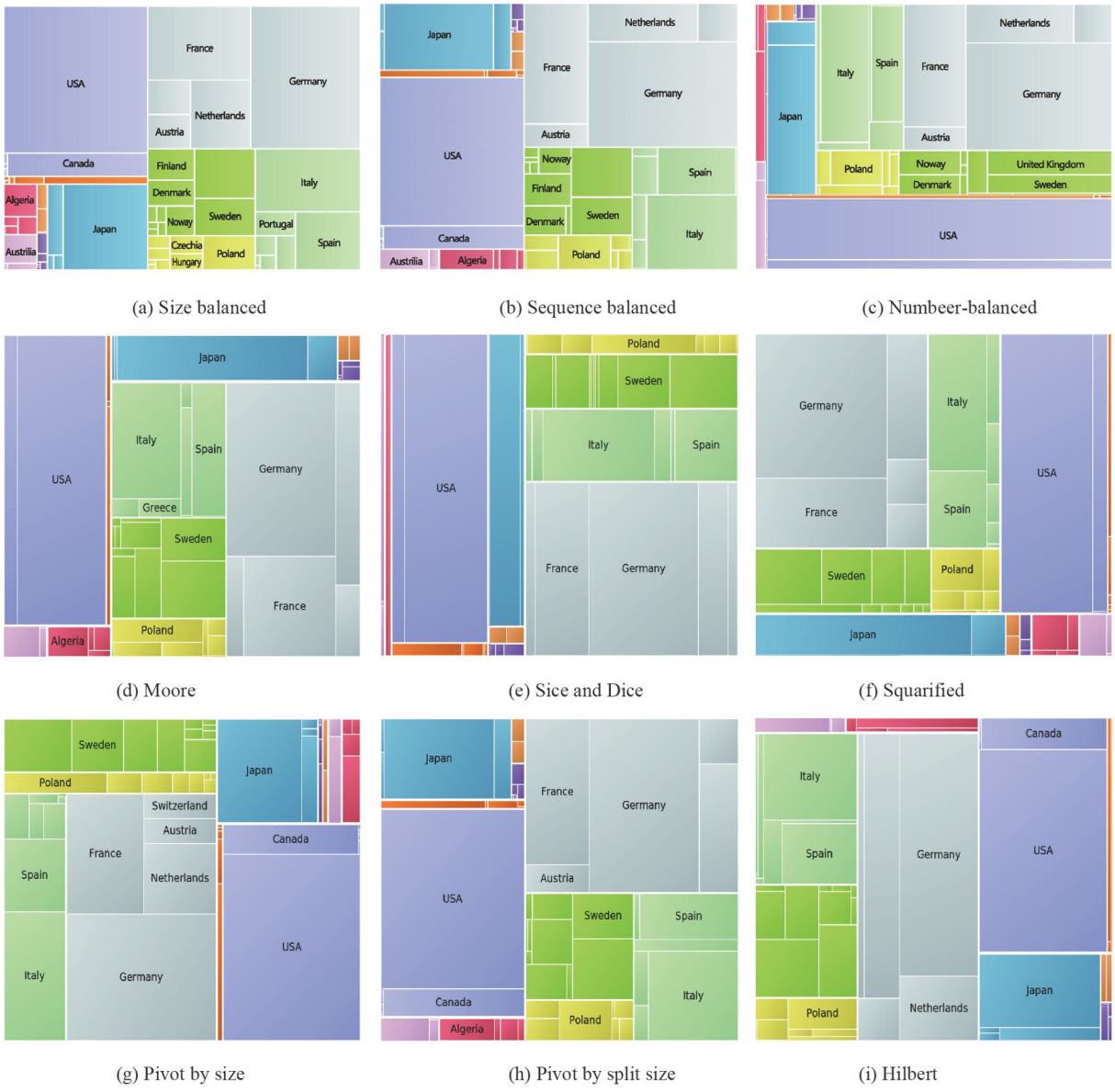


Figure 8 Visual results of the Coffee dataset. The methods (a-c) are compared with Moore (d), Slice and Dice (e), Squarified (f), Pivot by size (g), Pivot by split size (h), and Hibert (i) methods. Leaf nodes with the same parent root share the same color.

average median aspect ratio. The sequence-based method also presented a higher average mean aspect ratio than the incremental and approximation methods but comparable average median aspect ratio results. The number-balanced method did not yield satisfactory results for the aspect ratio. The average mean aspect ratio is better than Pivot by middle, Squarified, and Slice and Dice, and the average median aspect ratio is better than Slice and Dice. Another interesting observation, here, is that the simplified treemap method did not present a satisfactory average mean aspect ratio, which would ideally be more than 2000. This is because the average aspect ratio is easily influenced by very thin elements and squarified treemaps contain extremely thin rectangles.

A dynamic comparison is presented in Figure 7f, showing good center-movement results for the size-balanced and sequence-balanced methods, however, the average center movement of the proposed method was only higher than those of the incremental and slice and dice methods. The visual change of the sequence-based method was higher than for incremental, strip, and slice and dice treemaps, and the visual change of the size-based method in the middle of all methods. The number-balanced method performed well for dynamic

Figure 9 Visual results of the Population dataset. The size-balanced method(a) is compared with Spiral(b), Strip(c), Squarified(d), Slice and Dice(e), and Pivot by middle(f) methods. Leaf nodes with the same parent root share the same color.

properties, competing with the Incremental and Slice and Dice methods.

In addition, the study also showed the size-balanced treemap in Figure 9, compared to spiral, strip, squarified, slice and die, and pivot by middle. The results of other methods were obtained using the software provided by Sondag et al.[4].

It should be noted that the local moves method[4] aims to solve the problem of dynamic stability by using other treemapping methods for initialization and to form an approximation approach. In dealing with a simple dataset, such as the Name dataset, the proposed method and local-moves method performed similarly with respect to aspect ratio, however, the proposed method was found to better preserve the sequences. For complicated datasets, the local moves method sometimes fails with very low effectiveness. Figures 5 and 10 visually compare the results relating to the proposed method with those of the local moves, showing that similar aspect ratio results were obtained but that the sequence-balanced and number-balanced variants demonstrated much better preservation of both sequence and dynamic stability.

Finally, the computation performances of different algorithms are reported in Table 1, where the time is measured in milliseconds. The results clearly show that all three variants of the proposed algorithm can generate satisfactory results faster than most existing approaches.

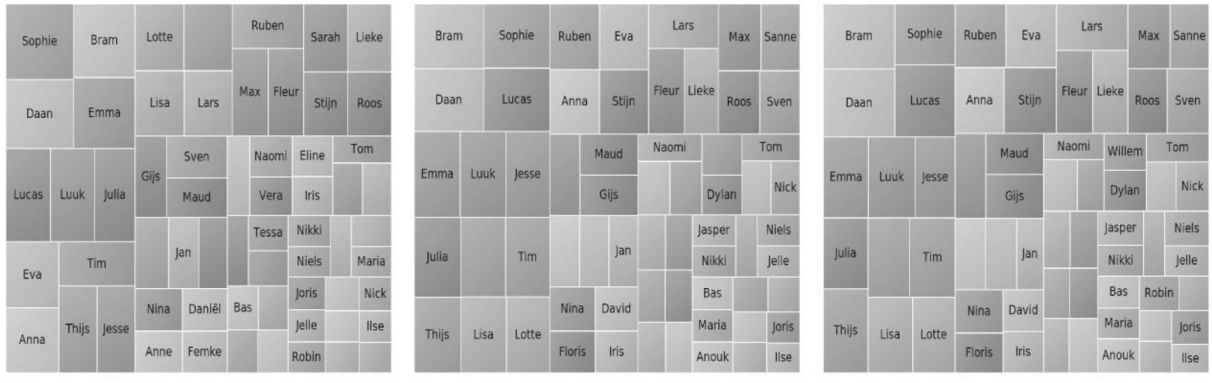
5 Discussion and conclusion

From the problem discussed in the abstract, this study aimed to obtain better visible performance of small elements as well as achieve dynamically stable results, resulting in the input dataset being visualized in a relatively stable position of the treemap. However, satisfactory static and dynamic performance can be difficult to achieve at the same time in one treemapping method and it is for this reason that the study put these two directions into one framework. In the framework, we have three sub-branches: one has good static

(a) Size-balanced

(b) Sequence-balanced

(c) Number-balanced



(d) Incremental

Figure 10 Visual results of three continuous years in Name dataset (from 1993 to 1995). The proposed methods (a-c) are compared with the Incremental (d) method. Slighter gray color indicates the earlier input whereas the darker gray color indicates the later input.

performance while with worse dynamic performance; one has good dynamic stable results while hard to visualize all small elements; one has static and stable dynamical performance between the former two.

This study proposed effective treemapping strategies that either maintain near-square aspect ratios for the rectangles of a treemap so that small elements are more visible or create stable layouts for time-dependent sequences (stock market variation with time), while these two aspects can be weighted in their influence.

Table 1 Computation performance of different approaches

Method	Time	Method	Time	Method	Time
Size Order	7.8	Sequence Order	20	Number Order	3.5
Slice and Dice	7.2	Pivot by Middle	25	Pivot by Size	28
Pivot by Split	22	Strip	22	Squarified	30
Spiral	23	Moor	25	Hilbert	38
Approximation	78	Incremental	75		

Three alternatives for arranging the results are thus designed: a size-balanced visualization maintains near-square aspect ratios, and a number-balanced version preserves the temporal coherence. An in-between is formed by a sequence-balanced partitioning method that maintains both aspects reasonably well.

For better comparison and evaluation, the study introduced a new metric that is defined based on the visual changes. It complements existing measurements for dynamic stability that usually only consider distance and size changes. This metric reflects shape variations, in addition to absolute distance and size changes. By comparing and discussing various methods using this measure in addition to other established measures, the study presented the behavior of the treemapping results with respect to standard datasets.

In the future, it is recommended that studies continue minimizing variances and maintaining temporal coherence. The use of a non-sliced method to divide rectangles is also an interesting exploration direction.

For the number-balanced partitioning method, we can combine hierarchical clustering and layout methods to not just two as close as possible. Moreover, for digit twins, the methods proposed in this study can be used for hierarchical dataset layout and visualization, which can be helpful for digit twin pipelines.

Declaration of competing interest

We declare that we have no conflict of interest.

References

1 Shneiderman B. Tree visualization with tree-maps. ACM Transactions on Graphics, 1992, 11(1): 92–99
DOI: 10.1145/102377.115768

2 Itoh T, Yamaguchi Y, Ikehata Y, Kajinaga Y. Hierarchical data visualization using a fast rectangle-packing algorithm. IEEE Transactions on Visualization and Computer Graphics, 2004, 10(3): 302–313
DOI: 10.1109/tvcg.2004.1272729

3 Wikipedia: The partition problem. https://en.wikipedia.org/wiki/Partition problem

4 Sondag M, Speckmann B, Verbeek K. Stable treemaps via local moves. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(1): 729–738
DOI: 10.1109/tvcg.2017.2745140

5 Treemaps S, Bruls M, Huizing K, Wijk J. Squarified Treemaps. 2000

6 Bederson B B, Shneiderman B, Wattenberg M. Ordered and quantum treemaps. ACM Transactions on Graphics, 2002, 21(4): 833–854
DOI: 10.1145/571647.571649

7 Tu Y, Shen H W. Visualizing changes of hierarchical data using treemaps. IEEE Transactions on Visualization and Computer Graphics, 2007, 13(6): 1286–1293
DOI: 10.1109/tvcg.2007.70529

8 Duarte F S L G, Sikansi F, Fatore F M, Fadel S G, Paulovich F V. Nmap: a novel neighborhood preservation space-filling algorithm. IEEE Transactions on Visualization and Computer Graphics, 2014, 20(12): 2063–2071
DOI: 10.1109/tvcg.2014.2346276

9 Lü H, Fogarty J. Cascaded treemaps: examining the visibility and stability of structure in treemaps. In: Proceedings of Graphics Interface 2008. Windsor, Ontario, Canada, Canadian Information Processing Society, 2008, 259–266

10 Tak S, Cockburn A. Enhanced spatial stability with Hilbert and Moore treemaps. IEEE Transactions on Visualization and Computer Graphics, 2013, 19(1): 141–148
DOI: 10.1109/tvcg.2012.108

11 Shneiderman B. Treemaps for space-constrained visualization of hierarchies including the history of treemap research at the university of maryland. 1992

12 Wang W X, Wang H, Dai G Z, Wang H. Visualization of large hierarchical data by circle packing. In: Proceedings of the SIGCHI

Conference on Human Factors in Computing Systems. Montréal Québec Canada, New York, NY, USA, ACM, 2006, 517–520
DOI:10.1145/1124772.1124851

13 Zhao H S, Lin L. Variational circular treemaps for interactive visualization of hierarchical data. In: 2015 IEEE Pacific Visualization Symposium (PacificVis). Hangzhou, China, IEEE, 2015, 81–85
DOI:10.1109/pacificvis.2015.7156360

14 Fischer F, Fuchs J, Mansmann F. ClockMap: enhancing circular treemaps with temporal glyphs for time-series data. In: Eurographics Conference on Visualization. 2012

15 Balzer M, Deussen O. Voronoi treemaps. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005. Minneapolis, MN, USA, IEEE, 2005, 49–56
DOI:10.1109/infvis.2005.1532128

16 Görtler J, Schulz C, Weiskopf D, Deussen O. Bubble treemaps for uncertainty visualization. IEEE Transactions on Visualization and Computer Graphics, 2018, 24(1): 719–728
DOI: 10.1109/tvcg.2017.2743959

17 Vernier E, Sondag M, Comba J, Speckmann B, Telea A, Verbeek K. Quantitative comparison of time-dependent treemaps. Computer Graphics Forum, 2020, 39(3): 393–404
DOI: 10.1111/cgf.13989

18 Scheibel W, Weyand C, Döllner J. EvoCells–a treemap layout algorithm for evolving tree data. In: 9th International Conference on Information Visualization Theory and Applications (IVAPP). 2018

19 Li G, Zhang Y, Dong Y, Liang J, Zhang J, Wang J, Mcguffin M J, Yuan X. BarcodeTree: scalable comparison of multiple hierarchies. IEEE Transactions on Visualization and Computer Graphics, 2020, 26(1): 1022–1032
DOI: 10.1109/tvcg.2019.2934535

20 Korf R E. Multi-way number partitioning. In: Proceedings of the 21st International Joint Conference on Artificial Intelligence. Pasadena, California, USA, 2009, 538–543

21 Scheibel W, Trapp M, Limberger D, Döllner J. A taxonomy of treemap visualization techniques. In: Proceedings of the 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications. Valletta, Malta, SCITEPRESS-Science and Technology Publications, 2020
DOI:10.5220/0009153902730280

22 Shneiderman B, Wattenberg M. Ordered treemap layouts. In: IEEE Symposium on Information Visualization, 2001. INFOVIS 2001. SanDiego, CA, USA, IEEE, 2001, 73–78
DOI:10.1109/infvis.2001.963283