J Vis (2022) 25:875–896
https://doi.org/10.1007/s12650-022-00830-1

REGULAR PAPER

Yan-Chao Wang · Yidan Xing · Feng Lin · Hock-Soon Seah · Jie Zhang

OST: a heuristic-based orthogonal partitioning algorithm for dynamic hierarchical data visualization

Received: 2 September 2021 / Revised: 24 November 2021 / Accepted: 2 February 2022 / Published online: 28 February 2022

Abstract Tools for intuitive visualization of dynamic datasets are highly demanded for capturing information and revealing potential patterns, especially in understanding the trend of data changes. We propose a novel resolution-independent heuristic algorithm, termed Orthogonal Stable Treemap (OST), to implicitly display dynamic hierarchical data value changes. OST adopts a site-based method as the Voronoi treemap (VT), to preserve the layout stability for diversified data values. Meanwhile, OST partitions the whole canvas with horizontal or vertical lines, instead of the lines with arbitrary orientations in VT. Technical innovations are made in three parts: Initialization of site state to speed up the algorithm and preserve the layout; efficient computation of orthogonal rectangular diagram to partition the empty canvas; self-adaption of site state to quickly reach an equilibrium. The performance of OST is quantitatively evaluated in terms of computation complexity, computation time, convergence rate, visibility, and stability. Moreover, qualitative evaluations (use case and user study) are demonstrated on the dynamic work-in-process dataset in the wafer fab. Evaluation results show that OST combines the advantages of layout stability and tidiness, contributing to easier and faster plot understanding.

Keywords Orthogonal rectangles · Treemap · Dynamic hierarchical data · Implicit hierarchy visualization · Resolution-independent

1 Introduction

Visualization of dynamic (time-varying) hierarchical data has long been demanded, because hierarchical data structures are quite common in people's daily lives (Graham and Kennedy 2010; Schulz 2011), and also in the scenario of Cyber-Physical Production Systems (CPPS) where a large amount of data generated by CPPS makes context-aware insights and online root-cause-effect analysis possible (Monostori 2014; Wang et al. 2019). For example, analysis of the dynamic performance information of machines which are organized into a hierarchy structure contributes to the completion of the established capacity at different abstract levels, such that customer needs can be met in time (Mönch et al. 2011). Implicit hierarchy visualization methods which focus on the value within each node and positionally encode the hierarchy by node overlap

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s12650-022-00830-1.

Y.-C. Wang (✉) · F. Lin · H.-S. Seah · J. Zhang
School of Computer Science and Engineering, Nanyang Technological University, Singapore, Singapore
E-mail: yanchao.wang@ntu.edu.sg

Y. Xing
Department of Biomedical Engineering, National University of Singapore, Singapore, Singapore

or inclusion are suitable for this objective to display the trend of value changes in a more space-efficient way (Schulz et al. 2011).

Implicit hierarchy visualization methods, in terms of the partitioning method, can be divided into non-site-based methods (i.e., treemap Shneiderman 1992) and site-based methods (i.e., Voronoi treemap (VT) Balzer and Deussen 2005). Non-site-based methods divide the empty canvas into rectangular sub-regions (Shneiderman and Wattenberg 2001; Bederson et al. 2002; Wood and Dykes 2008) or other shapes (Wattenberg 2005; Liang et al. 2012; Tak and Cockburn 2013) such that the area is associated with the relative sizes of the respective sub-hierarchies. Site-based methods, in a more general way, partition the canvas into polygon shapes based on the distance to prior specified sites; afterward, the position and weight of the sites may be iteratively adapted in order to adjust the area of the sub-regions (Sud et al. 2010; Gotz 2011). However, both treemap and VT have flaws in the flexibility of adjustment for visualization plots. To be specific, treemap is sensitive to even small changes in data, while VT is difficult for comparison due to its nested polygonal layout formed by segments with arbitrary orientations (Graham and Kennedy 2010; Wang et al. 2016). Although some recent researches (Scheibel et al. 2018; Vernier et al. 2018; Sondag et al. 2018) focus on designing additional algorithms on treemap to preserve the layout stability with small data change, their performance on large data changes is not discussed (Vernier et al. 2020). In contrast, the iterative adaption of VT makes it very suitable for handling dynamic data, regardless of whether the data change large or small. In response to these problems, we propose an orthogonal stable treemap to keep a balance between layout stability and tidiness.

The proposed orthogonal stable treemap (OST) partitions the empty canvas into nested orthogonal rectangles based on the distance to prior specified sites. On one hand, a site-based method is flexible to neighborhood design and diversified data values. On the other hand, orthogonal rectangular layout with sides parallel to the axes of Cartesian coordinates (as known as axis-aligned rectangles, rectilinear rectangles, or rectilinear polygons) is much tidier than polygonal layout formed by segments with arbitrary orientations in VT. To achieve this, we first propose an initialization process to determine an initial position and weight for each site by assessing either a treemap layout or the OST layout of the previous time step. After initialization, we propose a site-based space partitioning algorithm, including a relative distance calculation method and an accompanying layout generation algorithm. The new distance takes the relative positions and weights of two sites into consideration rather than a single site in the ordinary VT. Then, the layout generation algorithm, a sweepline + skyline heuristic algorithm inspired by the sweepline algorithm for Voronoi diagram generation (Fortune 1987) and the skyline algorithm in handling cutting and packing problems (Burke et al. 2004), is proposed to generate an orthogonal rectangular layout, partitioning the empty canvas based on the new distance calculation method. Lastly, an update process is designed to iteratively adjust the site distribution, such that the area of the orthogonal rectangular cells will match their associated data values. An orthogonal treemap will be obtained if this process is recursively continued layer by layer until the whole hierarchical structure is traversed. The performance of OST is quantitatively and qualitatively evaluated by extensive experiments, including character analysis of OST (computation complexity, computation time, convergence rate, visibility, and stability), use case on wafer fab dynamic data, and user study to analyze user experience. The results show that OST provides a stable layout on dynamic data while presenting a tidier layout which makes it easier for users to track the nodes and compare the size of shapes, achieving our objective of this paper. Discussion on the performance evaluation is made in the end, as well as the drawbacks and the future work.

This work is an extension of our previous work (Wang et al. 2020). Besides the original contributions, which are the proposed new orthogonal distance calculation method and the accompanying layout generation algorithm, we extend the previous work in several aspects. Firstly, to handle dynamic data, an initialization algorithm based on the layout of the previous time step is designed to preserve the layout stability. Experimental results show that this initialization significantly not only increases the convergence rate of OST, but also contributes to preserving layout stability. Secondly, the scenarios that may lead to empty cells during the layout generation are analyzed and avoided. Thirdly, we propose a state update process for each site to iteratively adjust the layout and reduce the computation time. Experimental results show that OST needs less time and preserves a stable layout over time. Lastly, extensive evaluations including the quantitative evaluation and qualitative evaluation are conducted to analyze the performance of OST, especially when handling dynamic hierarchy data.

The rest of this paper is organized as follows. Recent work related is reviewed in Sect. 2. The background knowledge and notations used in this paper are described in Sect. 3. In Sect. 4, we describe the proposed orthogonal space partitioning algorithm. The performance of our algorithm is quantitatively

evaluated in Sect. 5, and qualitatively evaluated including a use case in Sect. 6 and a user study in Sect. 7. Finally, discussions and conclusions are made in Sects. 8 and 9, respectively.

2 Related work

In this section, we discuss the previous research work which is closely associated with this work, including implicit hierarchy visualization methods, as well as methods designed for dynamic hierarchical data.

2.1 Implicit hierarchy visualization

Implicit hierarchy visualization methods mainly differ in the canvas subdivision strategies used to generate layouts (Kong et al. 2010). Based on whether the sites are referred to during the subdivision, we divide the methods into two clusters: non-site-based methods and site-based methods, both with the inclusion of edge representation according to the design space definition (Schulz et al. 2011; Scheibel et al. 2020a, b).

Non-Site-Based Methods Some implicit hierarchy visualization methods that partition the whole space without considering the sites are treated as non-site-based methods, such as the treemap. These methods position the data by following some rules or experiences in order to get expected configurations, which sometimes are also named as heuristic-based algorithms. Starting from the original treemap in 1992 (Shneiderman 1992), a large number of variants are proposed in the literature (Graham and Kennedy 2010; Wang et al. 2016; Scheibel et al. 2020a).

The squarified treemap focuses on the emergence of thin, elongated rectangles in the standard treemaps and presents a new subdivision method such that the resulting rectangles have a lower aspect ratio (Bruls et al. 2000; Lai et al. 2015). The ordered treemap layout is the first type of treemap layout that takes stability into consideration (Shneiderman and Wattenberg 2001). In their work, two pivot-based algorithms (pivot-by-size and pivot-by-middle) are proposed to ensure that items near each other in the original data will be near each other in the final layout. The split algorithm used in the ordered and quantum treemaps (Bederson et al. 2002) is a modification of the squarified treemaps, following a given one-dimensional ordering. The spiral treemap positions the one-dimensional ordering of the input data along the border following a circular arrangement or an S-shape (Tu and Shen 2007). Different from previous methods which only consider one dimension, the spatially order treemaps consider two-dimensional consistency by relating node order to Euclidean distance from the parent node's top-left corner (Wood and Dykes 2008). Another work that also focuses on spatially order for the geolocated quantitative data is the weighted maps (Ghoniem et al. 2015). We notice the layout generation problem in treemap which is similar to the two-dimensional (2D) bin packing which is an optimization problem with a wide range of applications in resource management. This is also discussed by Schulz et al. (2011). Since many heuristic algorithms (Burke et al. 2004; Wang and Chen 2015) have been proposed to solve the bin packing problem, how to utilize them in the layout generation in treemap would be an interesting research direction, and some researchers have made contributions (Itoh et al. 2004; Kobayashi et al. 2012; Chen et al. 2017).

Non-rectangular treemaps are also found in the literature. Jigsaw map has nicely shaped regions and stable layout by considering Hilbert curves or H curves (Wattenberg 2005). They generate irregular shapes with too many jigsaws which are not easy to be compared with. A modification is proposed by splitting the space into rectangles (Tak and Cockburn 2013; Scheibel et al. 2021). To relax rectangular constraint, angular treemaps describe a divide-and-conquer method to partition the space into various shapes (Liang et al. 2012). Besides that, the treemap layouts that produce irregular nested shapes by subdividing the Gosper curve (Auber et al. 2013) and along a one-dimensional continuum with pre-determined columns of grid (Armitage 2014) are also proposed.

Site-Based Methods Other implicit hierarchy visualization methods partition the space based on a series of pre-defined sites, such as the Voronoi treemap. The Voronoi treemap is originally presented by Balzer and Deussen (2005). By relaxing the constraint of rectangular shapes, they utilize Voronoi tessellations to generate polygonal subdivisions. They firstly initialize a set of sites with initial weight values and then compute the Voronoi tessellations based on distance functions. By adaptively altering the weight value of each site, it enables a dedicated Voronoi region in the next iteration step. Finally, the computation will be stopped when a good enough layout is reached. Later, the Voronoi treemaps are utilized to visualize dynamic hierarchical data owing to its adjustment ability (Sud et al. 2010; Gotz 2011). However, the calculation of these Voronoi treemaps is computationally expensive as a random-sampling strategy is used

to compute the Voronoi tessellations. In 2012, Nocaj and Brandes (2012) propose a resolution-independent algorithm by calculating the Voronoi tessellations with power diagrams, such that the new algorithm is faster in both theory and practice. An improvement is then made by setting an initial position for visualizing varying hierarchies (Hahn et al. 2014).

Neighborhood treemap (Nmap) (Duarte et al. 2014), which successively bisects a set of pre-defined sites on the horizontal or vertical directions and then scales the bisections to match the value of each site, is also a site-based method. Although no distance function is used during the segmentation, Nmap also needs sites representing the similarity relationships of data elements to be positioned in the canvas. Thus, Nmap can preserve similarity relationships among data elements very well. However, no evidence shows that Nmap can produce stable layouts with dynamic data. Circle packing (Wang et al. 2006) can also be treated as a site-based method since the generation of the layouts is based on the center of each circle, as well as the recently proposed bubble treemaps (Görtler et al. 2018).

2.2 Dynamic hierarchy visualization

The main target of using implicit hierarchy visualization methods to visualize the time-varying data is to display the value changes that evolve over time. One type of method merges two or more snapshots of time-varying hierarchical data into one layout, such as the contrast treemaps with spiral layouts (Tu and Shen 2007). The contrast treemap allows direct comparison of the values at two-time points by encoding the information from two different snapshots of time-varying data. Another example is the ClockMap in 2012 (Fischer et al. 2012), in which a combination of clock-based glyph and a circular treemap is proposed for comparative tasks on large amounts of hierarchically structured time series data.

The second type of method uses multiple displays to show each snapshot of the time-varying data. The site-based method, Voronoi treemap, is modified for computing stable updates with dynamic data. The fast dynamic Voronoi treemap proposed by Sud et al. in 2010 designs a re-seeding algorithm to provide stable updates as well as a GPU-based iterative algorithm for fast computation (Sud et al. 2010). The dynamic Voronoi treemap proposed by Gotz in 2011 uses a warping step to preserve the stability of the layout. The layouts of these two algorithms are based on resolution-dependent approximate computation, which leads to high computation time. In 2012, Nocaj and Brandes (2012) utilize Aurenhammer's method for power diagrams (Aurenhammer 1987) to remove the resolution limit. Later, an improvement by using a deterministic initial-distribution approach to reduce the variation in site positioning is proposed (Hahn et al. 2014). Researcher also considers adjusting the non-site-based method for dynamic hierarchical data recently (Chen et al. 2017; Sondag et al. 2018). Inspired by the thought of querying an array, ordered small multiple treemap (Chen et al. 2017) locates a specific node and preserves a relatively stable order of nodes in the layouts. However, the rigid layout may lead to a high aspect ratio if the value changes largely. The incremental treemap proposed by Sondag et al. (2018) changes the layout using only local modifications to handle the data value changes. The local movement makes it possible to explore the full range of options for choosing layouts, although it leads to a large computation time which is prohibitive for interactive applications. The greedy insertion treemap (Vernier et al. 2018) aims to preserve neighborhoods, by utilizing a layout tree to record the data structure where each node may have one top-right and one bottom-left subtrees. Similarly, with the same aim, the balanced partitioning treemap (Feng et al. 2019) utilizes a binary tree to record the data structure. A combination of the site-based and non-site-based methods is also proposed by mixing layout generation methods based on the characteristics and changes of the dataset (Bethge et al. 2017).

3 Preliminaries

In this section, we introduce the basic concepts and notations. A diagram in the implicit hierarchy visualization is a partitioning of a plane into sub-regions, and then, a treemap layout is a recursive partitioning of a plane. Taking the site-based VT as an example, a Voronoi diagram is a partitioning of a plane into sub-regions based on distances to a set of points within a bounded region. (We only consider the partitioning in a bounded region rather than the whole 2D plane.) These sub-regions are often called *Voronoi cells* (or *cells*), and the points in the plane are called *sites*. Then, recursively partitioning each sub-region with one cell for each leaf node, a Voronoi treemap layout is produced.

3.1 Voronoi diagram

Formally, given a bounded region $\Omega \subset R^2$ and a set of n sites $S = \{s_1, s_2, \ldots, s_n\}$, the Voronoi diagram divides Ω into a set of Voronoi cells υ_{s_i}, one for each site s_i based on a distance function. Then, the cell υ_{s_i} can be expressed as

$$\upsilon_{s_i} = \{p \in \Omega |\ dist(p, s_i) < dist(p, s),\ \forall s \in S, s \neq s_i\}, \tag{1}$$

where $dist(p, s_i)$ is the distance between point p and site s_i. The distance can be the Euclidean distance, the power Euclidean distance, or other distance functions. Thus, the Voronoi diagram is defined as the collection of Voronoi cells, $\upsilon_S = \{\upsilon_{s_1}, \ldots, \upsilon_{s_n}\}$.

3.2 Weighted Voronoi diagram

In a Voronoi diagram, the area of a cell is fixed and only depends on the positions of its associated and neighboring sites. Hence, in order to use the areas of cells to depict additional information (e.g., data value), a control mechanism is required. To achieve this, a positive real weight is associated with each site and should be considered when calculating the distance.

Several kinds of weighted Voronoi diagrams are proposed in the literature including additively weighted Voronoi diagram (Fortune 1987), power weighted Voronoi diagram (Aurenhammer 1987), and multiplicative weighted Voronoi diagram (Aurenhammer and Edelsbrunner 1984). We take the power weighted Voronoi diagram as an example. Formally, let $W = \{w_1, w_2, \ldots, w_n\}$ be a set of positive weights associated with the set of sites S correspondingly. Then, the power weighted Euclidean distance can be written as follows:

$$dist_{wpe}(p, s_i) = \|p - s_i\|^2 - w_i. \tag{2}$$

Increasing the weight value will increase the area of the cell. However, it is nonlinear in general. Besides that, a too-large weight value may lead to empty cells. We will discuss this overweight issue in the description of our algorithm.

3.3 Centroidal Voronoi diagram

In a weighted Voronoi diagram, since the positions of the sites are fixed, the generated diagram by adjusting the associated weight may lead to a strange shape. However, in the implicit hierarchy visualization, shapes with a good aspect ratio (i.e., the ratio of the sides of the oriented minimum bounding rectangle is close to one) ensure good readability and visibility. Hence, the centroidal Voronoi diagram, a special type of Voronoi diagram that the site s_i is located at the center of each cell υ_{s_i} (Du et al. 1999), is used to control the shape of cells. Let $\overline{\upsilon_{s_i}}$ be the polygonal boundary, the centroid $c_i = centroid(\overline{\upsilon_{s_i}})$ can be calculated in linear time.

3.4 Convergence requirement

Although the final layout requires that the area of each cell should be in proportion to the associated value, it cannot guarantee convergence (Gavrilova 1998; Du et al. 1999). Hence, we will say the process converges if the area error is smaller than a threshold. Formally, let $A(\upsilon_{s_i})$ and $A(\Omega)$ be the area of cell υ_{s_i} and the whole bounded region Ω, respectively, while let $value_{s_i}$ and $value_S$ be the associated value of site s_i and the whole set S, respectively. We assume $E_{threshold}$ be the threshold of the area error, and then, the convergent requirement can be expressed as:

$$\frac{1}{A(\Omega)} \cdot \sum_{s_i \in S} \left| A(\upsilon_{s_i}) - A(\Omega) \cdot \frac{value_{s_i}}{value_S} \right| < E_{threshold}. \tag{3}$$

3.5 Voronoi treemap

The Voronoi treemap is a recursive partitioning of a plane, the same as the treemap. Starting from the root of a hierarchy, a weighted centroidal Voronoi diagram is generated in the region Ω with one cell for each child of the root. An iterative optimization process is taken to adaptively alert the value of the weight and the

position of the site until the convergence requirement in Eq. (3). Once completed, the above-mentioned processes recurse to subdivide each region by assigning each υ_{s_i} as Ω to handle the children node of each s_i, if s_i is not a leaf node, until all the leaves of the hierarchy are represented by cells with desired areas. In this case, a Voronoi treemap layout is obtained.

4 Algorithm

Algorithm 1 : Compute Orthogonal Stable Treemap (single layer)

Input: D; Ω; j_{max}; $E_{threshold}$;
Output: Υ;
OST(D, Ω, j_{max}, $E_{threshold}$)

1: $\Upsilon = \{\}$;
2: $\upsilon_{S^0} = Treemap(d^1)$;
3: **for** $t = 1 : T$
4: $[S_1^t, W_1^t] = Initialization(\upsilon_{S^{t-1}}, \Omega, d^t)$;
5: **for** $j = 1 : j_{max}$
6: $\upsilon_{S_j^t} = WORDiagram(S_j^t, \Omega, W_j^t)$;
7: $err = \frac{1}{A(\Omega)} \cdot \sum_{s_i^t \in S^t} |A(\upsilon_{s_i^t}) - A(\Omega) \cdot \frac{value_{s_i^t}}{value_{S^t}}|$;
8: **if** $err < E_{threshold}$, **then**
9: **break** ;
10: $[S_{j+1}^t, W_{j+1}^t] = UpdateS\&W(S_j^t, \upsilon_{S_j^t}, W_j^t)$;
11: $\upsilon_{S^t} = \upsilon_{S_j^t}$;
12: $\Upsilon.push(\upsilon_{S^t})$;
13: **return** Υ ;

In this section, we present the OST algorithm for visualizing the dynamic hierarchical dataset $D = \{d^1, d^2, \ldots, d^T\}$ where d^t be the hierarchical data at time step t. For each d^t, the site position S^t and associated weight W^t are generated in order to produce the layout υ_{S^t}. All the layouts are saved in Υ for D. Due to the recursive feature of the implicit hierarchy visualization as mentioned in Sect. 3.5, let's consider a single layer hierarchy for our algorithm description. The pseudo-code of OST is listed in Algorithm 1.

An initialization process is first conducted on the hierarchical data to generate the initial position and weight for each site. This is achieved by the function *Initialization*($\upsilon_{S^{t-1}}$, Ω, d^t) (Line 4). For d^t, we initialize the site position S_1^t by transforming the previous layout $\upsilon_{S^{t-1}}$ into the current canvas. In particular, for d^1, we utilize the layout of a treemap *Treemap*(d^1) as the input of this initialization process. After that, an iterative process is taken to generate the weighted centroidal orthogonal rectangular diagram by the function *WORDiagram*(S_j^t, Ω, W_j^t) (Line 6) and update the site state by the function *UpdateS&W*(S_j^t, υ_{S^t}, W_j^t) (Line 10). If the convergence requirement (Line 7) is met before the maximum iteration limit j_{max}, then the iterative process will be stopped and the latest diagram $\upsilon_{S_j^t}$ will be recorded. The layout of each hierarchical data d^t will be saved in Υ.

4.1 Initialization of site state

To preserve the layout stability among the successive time steps, using the layout of the previous time step to compute the new layout is well accepted which is referred to as state-aware treemaps (Vernier et al. 2020). When it comes to our OST, we design an initialization mechanism to determine the initial position S_1^t and the initial weight W_1^t for each d^t, such that it can not only contribute to the layout stability, but also increase the convergence rate.

The initial position S^t for $t = 0$ is set with the aid of the squarified treemap algorithm (Bruls et al. 2000) for its good aspect ratio (Sondag et al. 2018) by transforming the sites in the squarified treemap into our canvas for d^1, while previous layout $\upsilon_{S^{t-1}}$ is used for other d^t. The main difficulty in this initialization is how to transform the sites in the squarified treemap or previous layouts into a new canvas Ω, considering shape difference. Note that the bounded region Ω in Algorithm 1 can be rectangles (for the root node) or orthogonal rectangles (for the non-root node). Here, we design a re-scale method for the transformation. A general case to show the transformation is illustrated in Fig. 1. As shown in Fig. 1a, the site position (x, y) is re-scaled to the interval [0, 1], concerning the relative height H and width W to obtain the relative position (R_x, R_y). When this relative position is used in the new canvas, it is decoded according to the shape of the new canvas. This decoding is conducted by considering R_y first (Fig. 1b) to find the new y' based on H', and then R_x (Fig. 1c) to find the new x' based on W' at y=y'. For S^t with $t \neq 0$, the initialization is based on the layout of the previous time step, and the same initialization process can also be applied. The advantage of the re-scale transformation is that it is suitable to transform the site into both rectangular and orthogonal rectangular canvases, as in our algorithm, orthogonal rectangular canvases are common.

The initial weight W^t for $t = 0$ is set to half of the area of the site's cell in the treemap ($w_i^t = A(\upsilon_{s_i^t})/2$) based on our experience. Compared to a random initial status, the designed initial status is a good solution and closer to the final result which contributes to a better convergence rate of the algorithm. For W^t with $t \neq 0$, the initial weight is the same as the weight in the previous time step.

4.2 Computation of weighted orthogonal rectangular diagram

The fundamental of a site-based space partition is the distance calculation. Here, we introduce a new distance function to axis-aligned subdivide the space and then consider two scenarios that may lead to empty

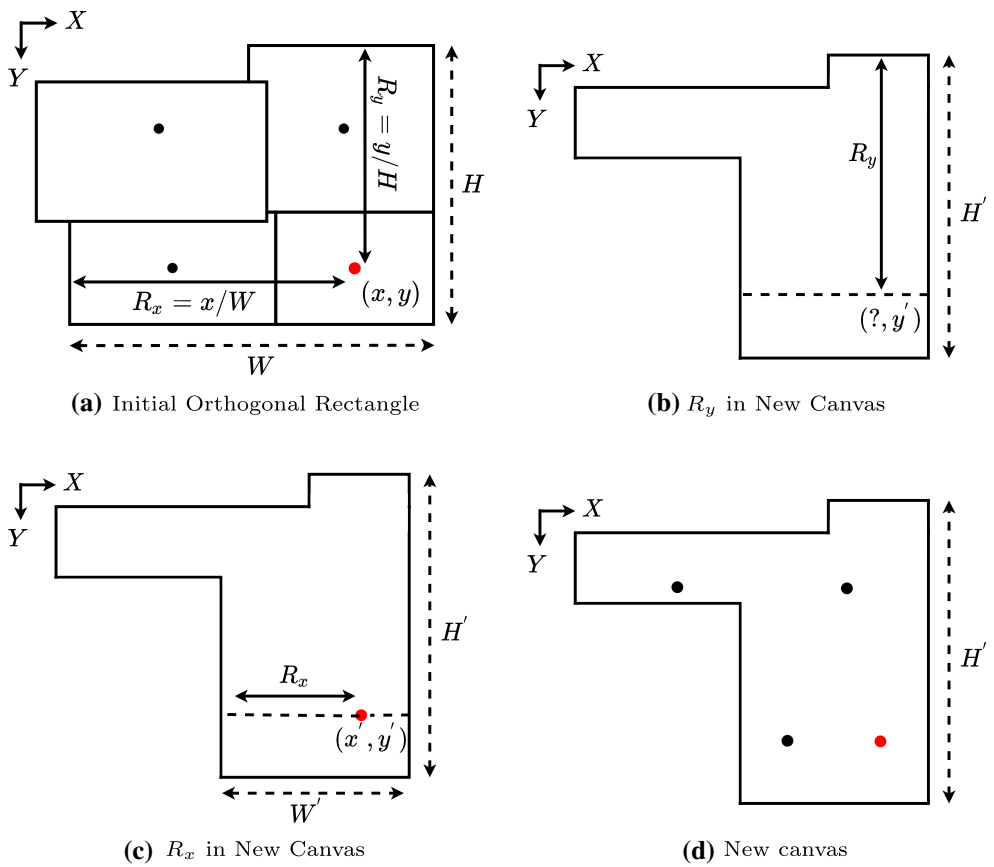


Fig. 1 The initial orthogonal rectangle is divided into four cells as shown in (**a**). To transform the site (x, y) (red dot) into the new canvas, the y' is first confirmed in (**b**), following by x' in (**c**). The final result is shown in (**d**)

cells. Lastly, a heuristic algorithm to automatically partition the space into orthogonal rectangular sub-regions based on the distance function is proposed, avoiding the scenarios leading to empty cells.

4.2.1 Distance function

Our distance function considers the relative positions of two sites rather than single-site likes that in Euclidean distance, in order to orthogonally partition the space. Formally, when calculating the orthogonal distance $dist_o(p, s_a, s_b)$ between point p and site s_a, we consider the relative positions of site pair s_a and s_b to decide which coordinate axis should be considered, where no other sites can be found within the rectangle formed by point p and site pair s_a and s_b. Two kinds of relative positions between site s_a and s_b based on their x-axis difference $x_{\Delta ab} = |x_a - x_b|$ and y-axis difference $y_{\Delta ab} = |y_a - y_b|$ are considered as illustrated in Fig. 2a, b. For the case in Fig. 2a, since $x_{\Delta ab} > y_{\Delta ab}$, site pair s_a and s_b are horizontal neighbors and the distance between point p and site s_a should be the horizontal distance $|x_p - x_{s_a}|$. For the case in Fig. 2b where site pair s_a and s_b are vertical neighbors, the distance between point p and site s_a should be the vertical distance $|y_p - y_{s_a}|$. For the case that $x_{\Delta ab} = y_{\Delta ab}$, we break the tie by treating them as horizontal neighbors. Then, considering the weight w_a, $dist_o(p, s_a, s_b)$ can be defined as:

$$dist_o(p, s_a, s_b) = \begin{cases} |x_p - x_{s_a}| - w_a & \text{if } x_{\Delta ab} \geq y_{\Delta ab}, \\ |y_p - y_{s_a}| - w_a & \text{if } x_{\Delta ab} < y_{\Delta ab}. \end{cases} \tag{4}$$

where $x_{\Delta ab} = |x_a - x_b|$ and $y_{\Delta ab} = |y_a - y_b|$. However, our distance function considering the relative positions is not safe for generating segmentation lines between two sites. In the following, we introduce the cases where segmentation lines cannot be properly generated and how we handle them.

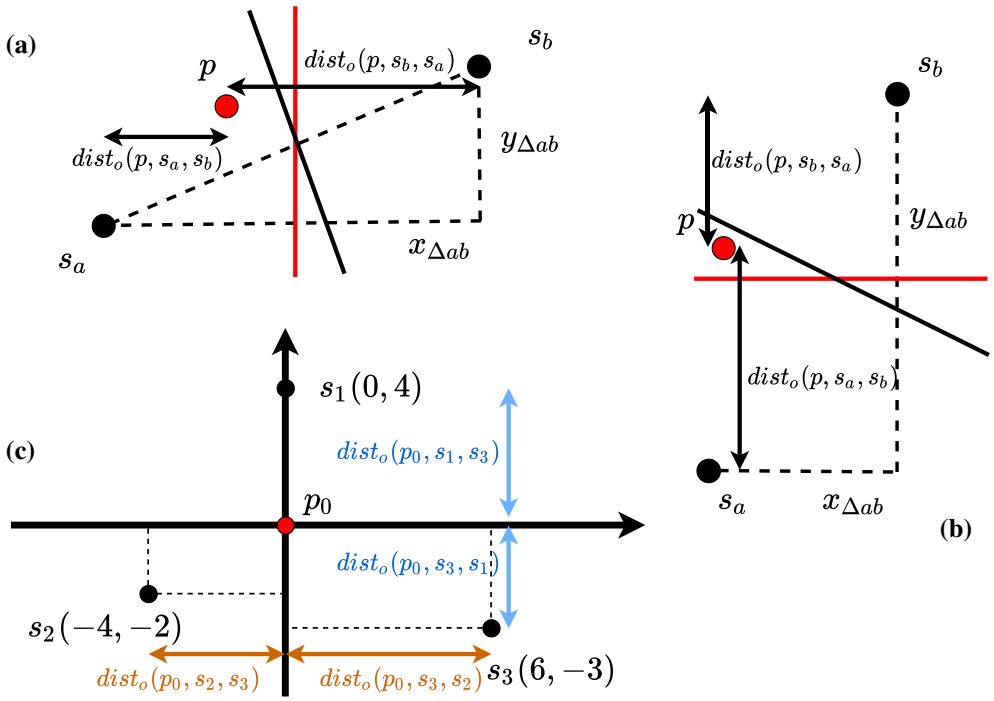


Fig. 2 Two kinds of relative positions between sites s_a and s_b where $x_{\Delta ab} = |x_a - x_b|$ and $y_{\Delta ab} = |y_a - y_b|$. The black solid line is the segmentation based on Euclidean distance, while the red solid line is an axis-aligned segmentation. The distance $dist_o(p, s_a, s_b)$ and $dist_o(p, s_b, s_a)$ in both cases are illustrated. **c** A scenario where the proposed distance function may lead to a tie and cause an empty cell

4.2.2 Empty cell scenario 1: overweight

For the case where the sum of w_a and w_b is larger than the Chebyshev distance between the two sites ($max(x_{\Delta ab}, y_{\Delta ab})$), we treat it as the overweight case. In this overweight case, we cannot find a point p on the segment between site s_a and site s_b with equal distance to both sites according to our distance function in Eq. (4). To guarantee that there is no cell with an empty region, the segmentation lines should be located in between (Nocaj and Brandes 2012).

To handle the overweight case, we locate the segmentation line according to the ratio of the weight values of both sites. In other cases, we follow the distance function in Eq. (4). The pseudo-code of the generation of the segmentation line is depicted in Algorithm 2.

Algorithm 2 : Generate Weighted Segmentation Line

Input: s_a; s_b; w_a; w_b;
Output: $L_{s_a s_b}$;
GenerateWLine(s_a, s_b, w_a, w_b)

1: **if** site s_a and s_b are horizontal neighbors, **then**
2: **if** $|x_{s_a} - x_{s_b}| - w_a - w_b \geq 0$, **then** $L_{s_a s_b} = min(x_{s_a} + w_a, x_{s_b} + w_b) + \frac{1}{2}(|x_{s_a} - x_{s_b}| - w_a - w_b)$;
3: **else**
4: **if** $x_{s_a} < x_{s_b}$, **then** $L_{s_a s_b} = x_{s_a} + \frac{w_a}{w_a + w_b} \cdot |x_{s_a} - x_{s_b}|$;
5: **else** $L_{s_a s_b} = x_{s_b} + \frac{w_b}{w_a + w_b} \cdot |x_{s_a} - x_{s_b}|$;
6: **else**
7: **if** $|y_{s_a} - y_{s_b}| - w_a - w_b \geq 0$, **then** $L_{s_a s_b} = min(y_{s_a} + w_a, y_{s_b} + w_b) + \frac{1}{2}(|y_{s_a} - y_{s_b}| - w_a - w_b)$;
8: **else**
9: **if** $y_{s_a} < y_{s_b}$, **then** $L_{s_a s_b} = y_{s_a} + \frac{w_a}{w_a + w_b} \cdot |y_{s_a} - y_{s_b}|$;
10: **else** $L_{s_a s_b} = y_{s_b} + \frac{w_b}{w_a + w_b} \cdot |y_{s_a} - y_{s_b}|$;
11: **return** $L_{s_a s_b}$;

4.2.3 Empty cell scenario 2: multiple sites

When dealing with multiple sites, the distance function may lead to a tie. In other words, mathematically, in most case $dist_o(p, s_a, s_b) \neq dist_o(p, s_a, s_c)$. Figure 2c shows a case of this scenario where $dist_o(p, s_3, s_2) \neq dist_o(p, s_3, s_1)$. Considering the horizontal neighborhood site pair s_2 and s_3, point p_0 has a smaller distance to s_2 ($dist_o(p, s_3, s_2) \geq dist_o(p, s_2, s_3)$). In this case, point p_0 should belong to site s_2. While, considering the vertical neighborhood site pair s_1 and s_3, point p_0 is closer to s_3 ($dist_o(p, s_1, s_3) \geq dist_o(p, s_3, s_1)$). In this case, point p_0 should belong to site s_3. For this scenario, we design a heuristic algorithm, termed as sweepline + skyline algorithm, to generate segmentation lines among multiple sites by giving priority to the site pair on the left, which will be described in the next part.

4.2.4 The sweepline + skyline algorithm

The sweepline + skyline algorithm aims to automatically partition the canvas into orthogonal sub-regions based on the new distance function and avoid the scenarios leading to empty cells. The algorithm is inspired by the sweep line algorithm for the Voronoi diagram (Fortune 1987) and the skyline algorithm in cutting and packing problems (Burke et al. 2004). The sweepline used in our algorithm is a vertical line moving from left to right. The length of the sweepline is equal to the height of the outer bounding rectangle of Ω. When the sweep line hits a new site, the neighborhood relationship of this new site and all its left-side site pairs is checked to generate vertical or horizontal segmentation lines in between. Meanwhile, a skyline is defined to

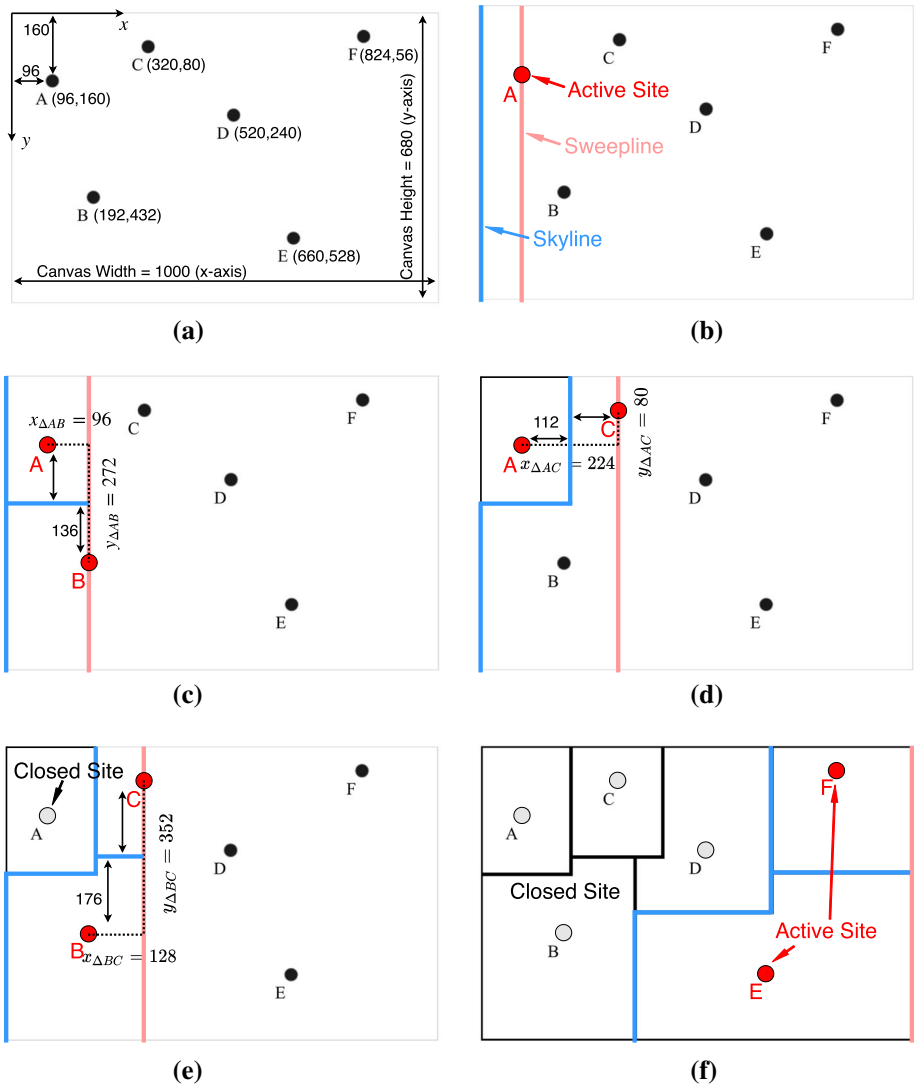


Fig. 3 Overview of the sweepline + skyline algorithm. **a** Initially, six sites are positioned inside the canvas based on their initial positions. **b** The skyline (in blue) and the sweepline (in pink) are initialized, and the first site *A* (left-most) is activated. **c** The sweepline is swept to site *B* and a horizontal segmentation line L_{AB} is added to the skyline. **d** The sweepline is swept to site *C*, and a vertical segmentation line is generated between sites *A* and *C*. In this case, site *A* is closed. A bounding polygon for site *A* is formed by the skyline, the canvas boundary, and the vertical segmentation line. Then, the skyline is updated, and site *A* is marked as closed status (**e**). The process is continued until the last site *F* is considered (**f**). The residual sites (*E* and *F*) will then be closed and the bounding polygon for each site is formed by the current skyline, sweepline, and the canvas boundary together

record the current segmentation lines for all active sites. When the sweep line hits a new site and new segmentation lines are generated, the skyline will be updated correspondingly. The pseudo-code for the whole process is depicted in Algorithm 3.

Figure 3 illustrates an example of this process. As shown in Fig. 3a, for a given rectangular canvas with $width = 1000$ and $height = 680$, six sites are positioned based on their coordinates. (We follow the image coordinate system where the y-axis is down.) The size of the canvas and the positions of sites are the input value of our algorithm. The first step (Fig. 3b) is to create a vertical sweepline and a vertical skyline. The sweepline is initially located on the left-most site *A*, while the skyline is on the left-hand side of the canvas. The second step (Fig. 3c) is to sweep the sweepline from left to right to hit the next site *B*. Since $y_{\Delta AB} = 272$ is larger than $x_{\Delta AB} = 96$, a horizontal line $L_{AB} : y = 296$ is built between sites *A* and *B* according to Algorithm 2. The skyline is then updated by adding a horizontal line segment. After checking all the left-hand-side site pairs of site *B*, the sweepline moves to the next site *C* (Fig. 3d). Since site *A* is not closed and

is the horizontal neighbor of site C, then a vertical line $L_{AC} : x = 208$ is built. Once a vertical line is generated, the left site A of these horizontal neighbor sites should be closed and the bounding polygon of site A is formed based on the current skyline and the new vertical segment (as well as the canvas boundary). After that, the skyline is updated. Since site B is not closed and is the vertical neighbor of site C, then a horizontal line $L_{BC} : y = 256$ is built and the skyline is updated again (Fig. 3e). This process is repeated until the sweepline reaches the last site F (Fig. 3f). The residual sites (E and F) will then be closed and the bounding polygon for each site is formed by the current skyline, sweepline, and Ω together. An animation of this example can be found in the supplementary materials.

When the bounded region Ω is not a rectangle, the proposed sweepline + skyline algorithm is still suitable. The skyline used in our algorithm is the left boundary of Ω, while the sweepline is a vertical line with equal length as the height of the outer bounding rectangle of Ω. For orthogonal rectangular region Ω, the initial skyline will be formed by multiple segments as the case shown in Fig. 3c. When the sweepline moves toward the next site, the skyline will be updated based on the new bisector and the boundary of Ω as the process shown in Fig. 3.

According to Algorithm 3, function *WORDiagram* has an $O(n^2)$ computation complexity in the worst case. However, much fewer site pairs exist as some of the sites have been closed already (Line 5 in Algorithm 3). Here, to simplify the process, we define the concept of a valid neighborhood relationship between sites. For a site pair, if there are no other sites located in the rectangle region formed by the two sites, then these two sites are valid neighbors. Instead of updating the diagram for each site pair, we only check the valid site pair. We will experimentally evaluate the performance of our OST in Sect. 5.1. It should be noted that in Algorithm 1: $OST(D, \Omega, j_{max}, E_{threshold})$ in the original paper (Wang et al. 2020), the inputs of function *WORDiagram* are S_j^t, and W_j^t. However, the superscript t and the subscript j are not used in Algorithm 3. Hence, we ignore them to avoid misunderstanding.

Algorithm 3 : Compute Weighted Orthogonal Rectangular Diagram

Input: S; Ω; W;
Output: v_S;
WORDiagram (S, Ω, W)

1: Sort S in the order of x_{s_a} ascending ;
2: Initialize $L_{sweepline}$, $L_{skyline}$ based on Ω, $v_S = [\,]$;
3: **for** $a = 1 : n$
4: **for** $b = 1 : a - 1$
5: **if** site s_b is not closed, **then**
6: **if** site s_a and s_b are vertical neighbors, **then**
7: $L_{s_a s_b} = GenerateWLine(s_a, s_b, w_a, w_b)$;
8: Update $L_{skyline}$ with $L_{s_a s_b}$;
9: **else**
10: $L_{s_a s_b} = GenerateWLine(s_a, s_b, w_a, w_b)$;
11: Mark site s_b as closed;
12: Generate v_{s_b} for s_b based on $L_{sweepline}$, $L_{skyline}$, and Ω;
13: $v_S.push(v_{s_b})$;
14: Update $L_{skyline}$ with $L_{s_a s_b}$;
15: Update $L_{sweepline}$;
16: **for** $a = 1 : n$
17: **if** site s_a is not closed, **then**
18: Generate v_{s_a} for s_a based on $L_{sweepline}$, $L_{skyline}$, and Ω;
19: $v_S.push(v_{s_a})$;
20: **return** v_S;

4.3 Self-adaption of site state

The state of each site (S, W) should be updated iteratively so that the area can match its value which is the key point of the implicit hierarchy visualization. Here, to update the position and the weight value of each site (Hahn et al. 2014), we propose a new process that deals with our orthogonal segmentation. The pseudo-code for the whole process is depicted in Algorithm 4.

We consider the adjustment rate $f_{adapt,j}$ at iteration j as the ratio of the area of the current cell to the target cell (Line 3). To avoid oscillation, we limit $f_{adapt,j}$ to $1 \pm \rho$ where ρ is set to 0.05 in this paper for our evaluation. Then, w_i and s_i are adjusted accordingly (Line 4-6). It should be noted that the new site position needs to be within the cell to avoid any collision (Line 7).

Compared with previous methods (Nocaj and Brandes 2012; Hahn et al. 2014), a significant advantage is that our adjustment skips the calculation of the distance to the nearest neighbor. In these previous methods, the calculation of the distance to the nearest neighbor is essential as they have to guarantee that the sum of two updated weight values should be smaller than the distance between this site pair, in order to avoid empty cells. This is the scenario that we mentioned in Sect. 4.2.2. In our method, in response to this scenario, we have proposed the new segment generation method in Algorithm 2. In this case, we do not need to calculate the distance to the nearest neighbor during the adjustment.

An additional consideration is about how to remove sites from the layout or add new sites. The removal of a site is straightforward by setting the value of the site to zero. While for the insertion of a new site, we set the initial position of the new site to the top-left corner of its parent node's cell and the initial weight based on its associated value.

Algorithm 4 : Adapt the positions and weights of sites.

Input: S_j^t; v_{S^t}; W_j^t;
Output: S_{j+1}^t; W_{j+1}^t;
UpdateS&W (S_j^t, v_{S^t}, W_j^t)

1: $\rho = 0.05$;
2: **for** $i = 1 : n$
3: $\quad f_{adapt,j} = min(max(\frac{A(\Omega) * value_{s_i}}{A(v_{s_{i,j}}) * value_S}, 1-\rho), 1+\rho)$;
4: $\quad w_{i,j+1} = w_{i,j} * \sqrt{f_{adapt,j}}$;
5: $\quad c_i = centroid(\overline{v_{s_{i,j}}})$;
6: $\quad s_i^* = s_{i,j} + (c_i - s_{i,j}) * (1 - 0.5\rho)$;
7: $\quad s_{i,j+1} = s_i^*$ within $\overline{v_{s_i}}$? s_i^* : $s_{i,j}$;
8: $\quad \rho$ += $f_{adapt,j}$;
9: **return** S_{j+1}^t, W_{j+1}^t;

4.4 Implementation

The implementation of the proposed algorithm is in JavaScript. The hierarchical structure formed by *d3.hierarchy()* from the D3.js package (Bostock et al. 2011) can be directly used in our code. Hence, it is convenient for the user to generate the OST plot. The source code with examples of our algorithm can be found in the supplementary materials and will be uploaded to the Github website.

5 Experiments and comparative analyses

The performance of our OST has been quantitatively evaluated from multiple aspects, including computation complexity, computation time (per iteration), convergence rate, visibility, and stability. The first three aspects affect the interactive performance (Nocaj and Brandes 2012), while the last two aspects are the two main components of the treemap quality (Vernier et al. 2020).

Our experiments were conducted on three datasets, including several random datasets (single layer with 50 - 50000 sites), the global GDP dataset (two layers with 43 leaf nodes), and the Flare class hierarchy (four

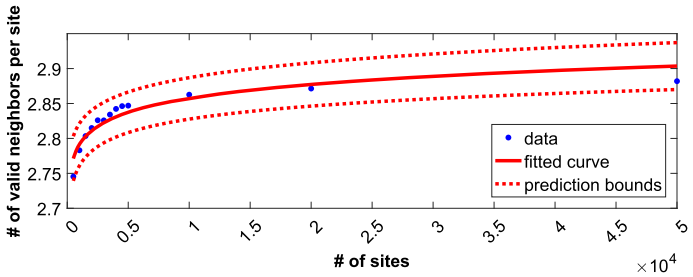


Fig. 4 The fitting line and the prediction for the average number of valid neighbors per site

layers with 220 leaf nodes). Our experiments were conducted on an HP desktop (Window 10, Intel Xeon CPU, 3.6 GHz, 16 GB memory).

5.1 Computation complexity

The computation complexity of a single layer OST is related to the complexity of function *WORDiagram* and the complexity of function *UpdateS&W*, as depicted in Algorithm 1. Moreover, the complexity of function *UpdateS&W* is $O(n)$ (Hahn et al. 2014). Hence, we focus on the complexity of the function *WORDiagram*.

We consider the worst-case and the average case. In function *WORDiagram*, all the non-closed sites will be checked to update the state for generating new segmentation lines and updating the skyline. In the worst case, it will lead to an $O(n^2)$ computation complexity. However, in an average-case scenario, much fewer valid site pairs exist. The OST algorithm was run on a series of datasets with a different number of sites (from 500 to 50000). For each dataset, we randomly initialize the position of sites and then calculate the number of valid neighbors for each site according to Algorithm 3. Each dataset was tested one hundred times, and the average results are recorded. We formulate the relationship between the number of sites (x) and the average number of valid neighbors per site (y) by fitting (as shown in Fig. 4), which can be expressed as:

$$y = 0.03002 * log(x) + 2.583. \tag{5}$$

The goodness-of-fit statistics in this case are SSE=0.00067, R-square=0.9600, Adjusted R-square=0.9564, and RMSE=0.0078 which indicate that the fitting function is suitable. Hence, based on the experimental result, in an average-case scenario, the computation complexity of a single layer OST is $O(T \cdot j \cdot n \cdot \log(n))$ where T is the number of time index, j is the number of iteration, and n is the number of sites.

When it comes to the multi-layer case, the amount of non-leaf nodes in the multi-layer hierarchy will affect the computation complexity. For each non-leaf node, the single-layer OST will run once. For a full n-ary tree with m layers (Depth $= m$), there are $\frac{n^m - 1}{n - 1}$ non-leaf nodes. Then, in an average-case scenario, the computation complexity of a full n-tree with m layers OST is $O(T \cdot j \cdot \frac{n^m - 1}{n - 1} \cdot n \cdot \log(n))$.

5.2 Computation time

We compare the running time of our algorithm during a single iteration to that of the VT with the JavaScript implementation. The experiments were conducted on a series of random datasets with a different number of

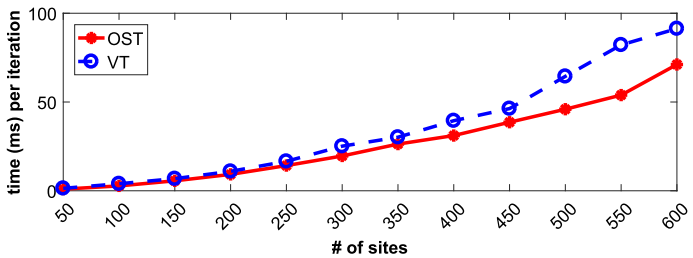


Fig. 5 The running time (in ms) for a single iteration

sites (from 50 to 600). For each dataset, both algorithms run 1000 iterations (we comment out the convergence requirement in Eq. (3), such that the algorithm will keep running until the maximum iteration number is reached). Figure 5 plots the average running time (in *ms*).

Based on the experimental results, our OST needs less time per iteration than the VT as shown in Fig. 5. The reason for this is the self-adaption process in Algorithm 4. In our modification, we merge the update of positions and weights into one step and handle the overweight case in the generation of the segmentation line. Hence, in one iteration of our algorithm (Algorithm 1), we only need to draw the diagram once rather than twice in VT (Nocaj and Brandes 2012; Hahn et al. 2014).

5.3 Convergence rate

The converging speed is considered as the changes of the area error ratio *err* along with the iteration index *j* in Algorithm 1. We generate a single-layer hierarchical dataset with 200 randomly positioned sites that are associated with random values. Both VT and our OST were run 5 times. We also compared our initialization process with a random initial status for OST. The *err* in each iteration is then plotted in Fig. 6.

As illustrated, OST has the fastest convergence. It is believed that the reason for this result is our reasonable initialization process. Since the layout of OST is similar to the treemap, using treemap to set the initial positions significantly contributes to the fast convergence in our algorithm. However, although our algorithm can converge under a tough constraint (0.01 area error ratio), it should be noted that OST has a larger fluctuation than VT. We consider that this is due to the non-consistent distance function used to partition the space (sometimes horizontal, sometimes vertical).

5.4 Visibility

The aspect ratio is often used to measure the visibility of the layouts (i.e., visual quality). As the cells in OST are orthogonal rectangles, the aspect ratio of the axis-aligned minimum bounding box is measured. We set the maximum iteration number to 500 and the area error threshold for convergence to 0.05. The results are illustrated as boxplots shown in Fig. 7.

When considering the mean aspect ratio (the red line), our algorithm is slightly better than the binary treemap and the squarified treemap on the random dataset and has similar results on two real datasets . When comparing the different initial statuses of our algorithm, we can find that with the designed initialization process, our algorithm has a better aspect ratio (the closer to one the better) and a small distribution range.

5.5 Stability

Several metrics have been proposed in the literature to evaluate the treemap layout stability. Layout distance change measures the average change of each cell in position and shape between two successive layouts (Shneiderman and Wattenberg 2001). Average centroid positioning (ACP) measures the average distance moved by the centroid of a rectangle (Hahn et al. 2014). Moreover, location drift (Tak and Cockburn 2013), visual change (Vernier et al. 2018), and corner travel distance (Vernier et al. 2020) are also used to measure how much individual rectangles move. Meanwhile, the relative change of pairs of rectangles is also studied to measure the layout stability (Scheibel et al. 2018; Sondag et al. 2018). However, these metrics are designed for layouts with rectangular cells. In this paper, we use ACP to measure stability, as it can easily be

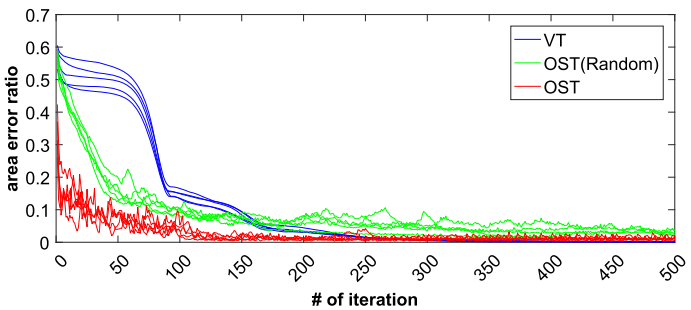


Fig. 6 The convergence rate of the Voronoi treemap, OST with random initial status and with our initialization process

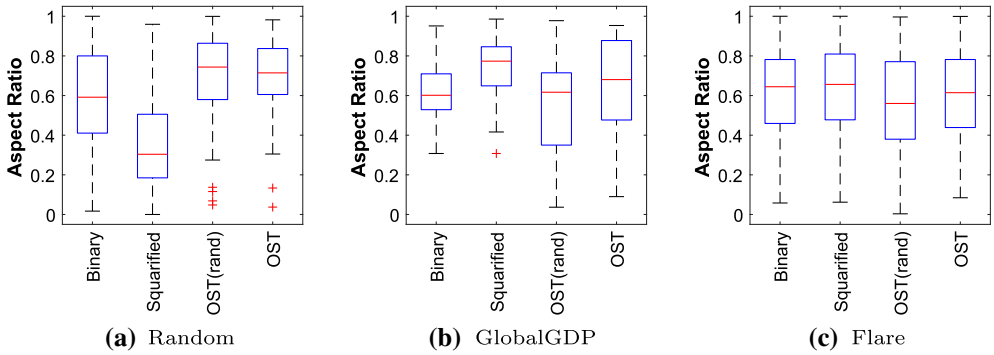


Fig. 7 The boxplots of the aspect ratio of different algorithms on three datasets

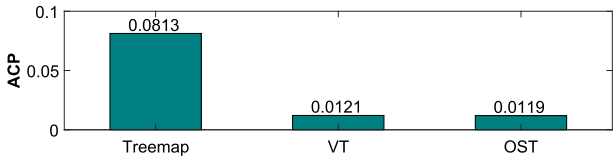


Fig. 8 The average centroid-positioning (ACP) value of three algorithms: squarified treemap, Voronoi treemap, and our OST

extended to non-rectangular treemaps. In our experiment, squarified treemap, VT, and OST generated corresponding plots for the globalGDP_2018 and globalGDP_2019 datasets. Then, the difference between the two layouts produced by each method is plotted in Fig. 8.

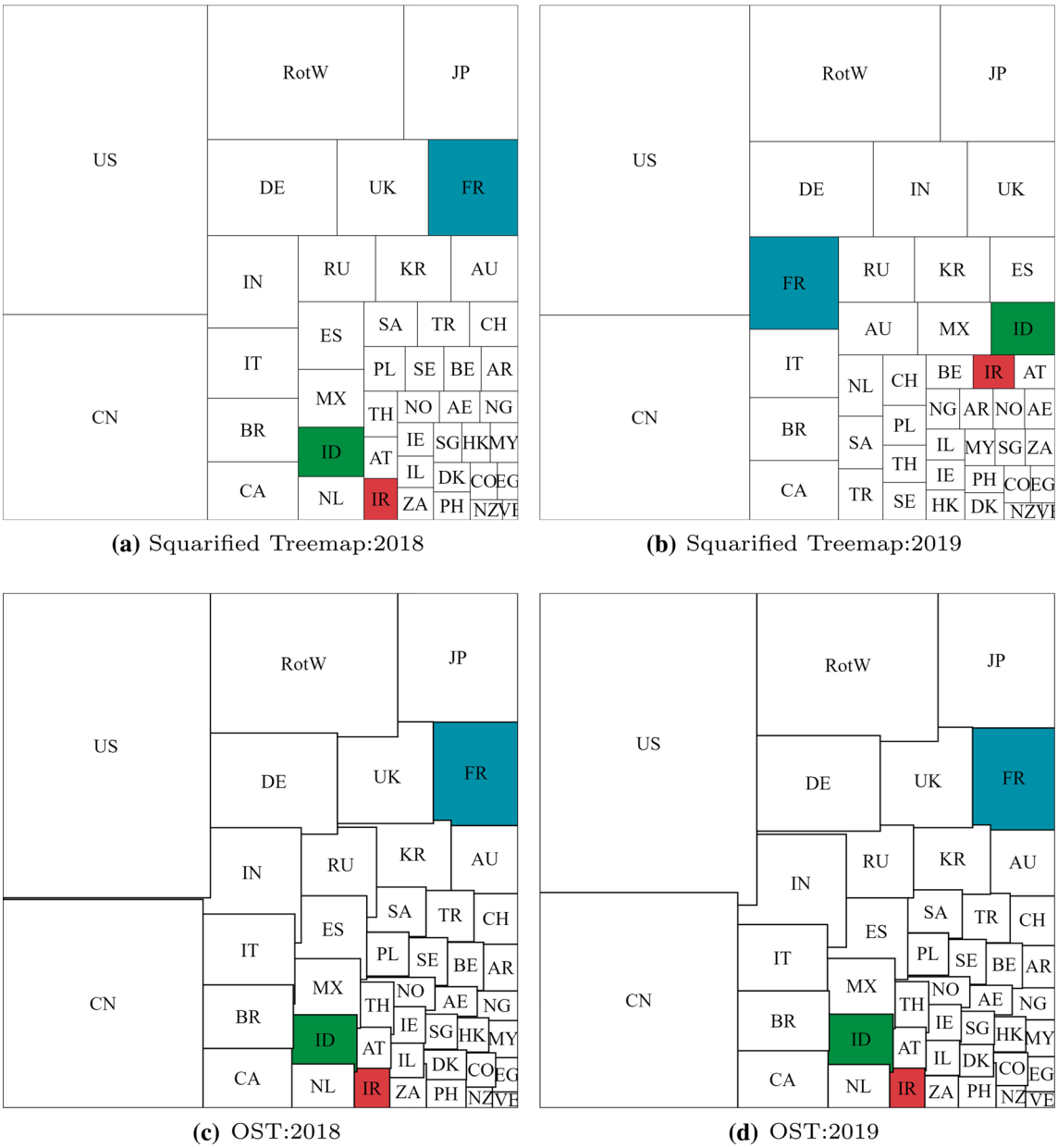
Squarified treemap has a large ACP value which means the layout is not stable. Meanwhile, VT and OST have small values. It means that the layout of OST will not change significantly with small data changes. The layouts of the squarified treemap and our OST are illustrated in Fig. 9, in which we highlight three nodes for comparison. We use the same color for the same site in these four plots. Obviously, in Fig. 9a, b, sites change dramatically, especially at the bottom-right corner. Meanwhile, the plots (Fig. 9c, d) generated by our OST only have a slight change.

6 Use case: semiconductor wafer fab WIP data visualization

Wafer fab as the most important part of semiconductor manufacturing accounts for more than 75% of the total cycle time as well as the largest component of cost (Mönch et al. 2011). Tracking the output of wafer fab can be achieved through the analysis of machine information from the lowest level (machines) to the highest level (wafer fab), which is organized in a hierarchical form.

In this use case, we applied the proposed OST method to a semiconductor wafer fab work-in-process dataset generated under a CPPS project. In the wafer fab, machines can be organized into a hierarchy structure (wafer fab → work area → work center → machine group → machine) based on the process flows and machine functionality. The work-in-process (WIP) information is of primary importance as WIP helps to monitor the performance of the whole system and a balanced WIP distribution contributes to meeting the product demand (Zhang et al. 2017). Hence, here, we focus on the visualization of dynamic WIP data in the wafer fab. The WIP data we used were collected from industry, and the main purpose is to find out the trend of value changes caused by the machine down in the whole system.

Figure 10 shows the average WIP information (*AvgWIPLots*) of each work center (wafer fab → work area → work center) at different times. Comparing Fig. 10a, b, at the work area 'OVN', the *AvgWIPLots* of work center 'OVN_138' increases significantly. Meanwhile, the *AvgWIPLots* of 'OVN_139' decreases at $T2$ as well as work centers under the work area 'WET' except 'WET_215'. At $T4$ (Fig. 10d), the *AvgWIPLots* of work centers under work area 'WET' except 'WET_215' recovers to a similar level at $T1$. Moreover, work centers under the work area 'CVD' follow a similar trend. The performance of this use case indicates



(a) Squarified Treemap:2018 **(b)** Squarified Treemap:2019

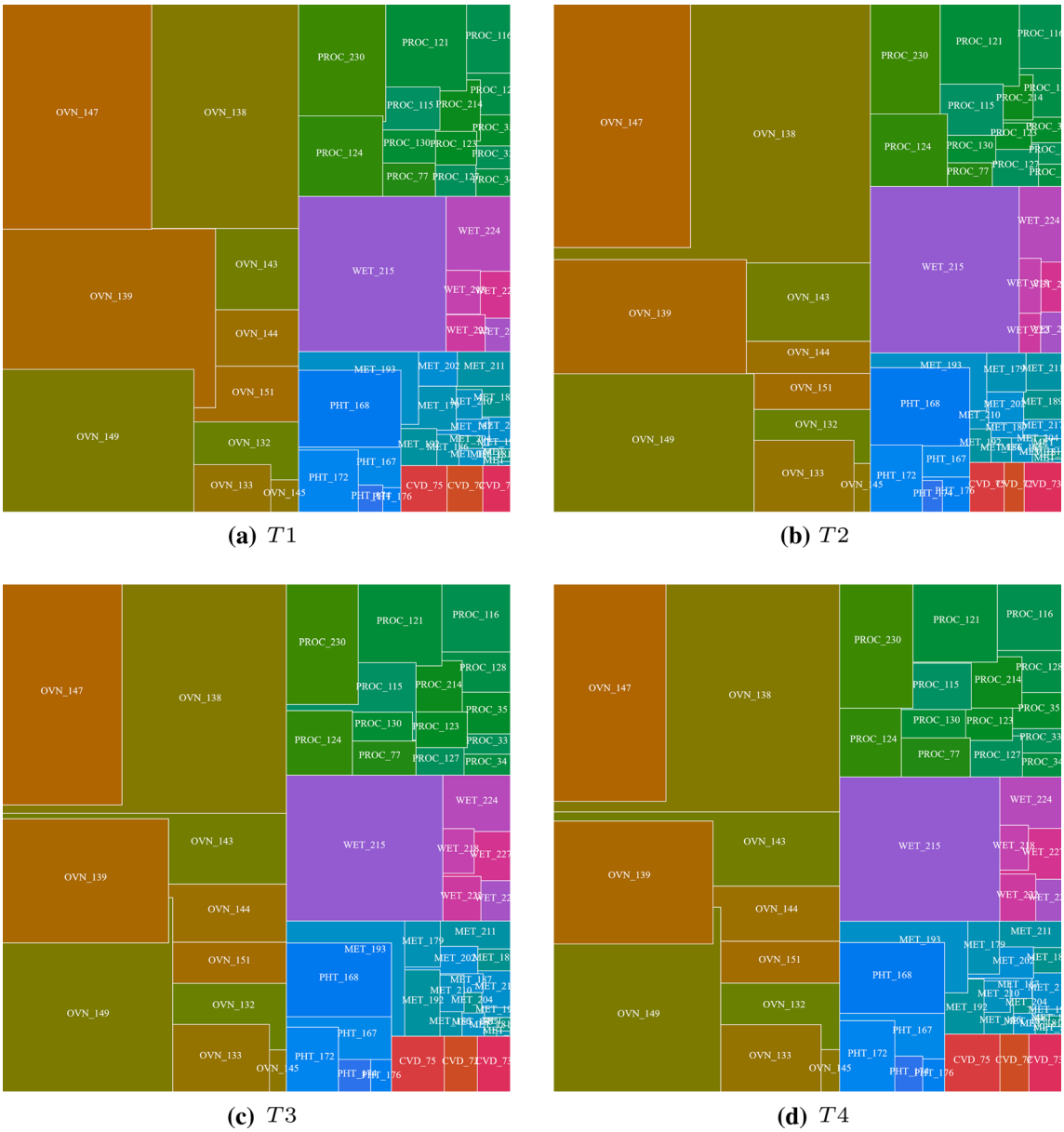
(c) OST:2018 **(d)** OST:2019

Fig. 9 The layouts of the squarified treemap (**a**, **b**), and OST (**c**, **d**) on the globalGDP2018 and globalGDP2019 datasets. According to the initialization process introduced in Sect. 4.1, the initialization of (**c**) is based on (**a**), and the initialization of (**d**) is based on (**c**). Three nodes ('FR', 'ID', and 'IR') are highlighted for stability comparison

that it is efficient to track nodes over time in our stable layouts. Another example to display the finished WIP lots information of each work center is also illustrated here in Fig. 11.

7 User study

A usability study is commonly used in the research community to understand the potential and limitations of a proposed visualization method (Long et al. 2017; Fiedler et al. 2020). We further performed a user study to assess the performance of our OST method. Twelve participants (9 males, 3 females), who are university students and unfamiliar with any implicit hierarchy visualization methods, joined our study. In this study, we used the semiconductor wafer fab WIP data of around 50 work centers captured at five different time



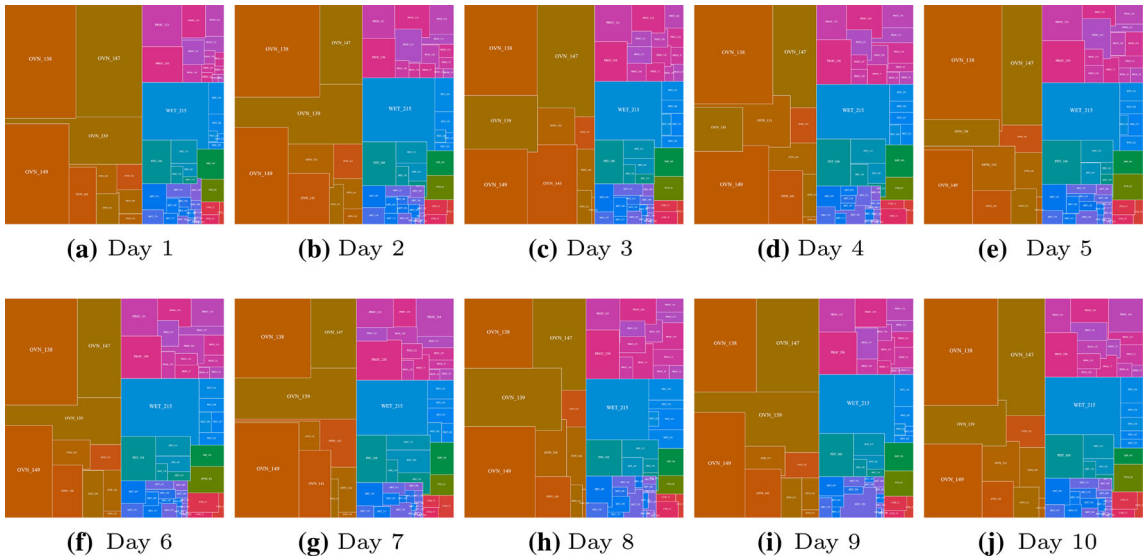
(a) $T1$ **(b)** $T2$

(c) $T3$ **(d)** $T4$

Fig. 10 The average WIP lots information at the work center level. The layout in **a** is initialized based on the squarified treemap, while the rest three layouts are initialized based on the previous layout

points. Before the user study, all the participants were given some knowledge about the implicit hierarchy visualization methods (Treemap, VT, and OST).

The user study was divided into three parts. In the first part, participants were asked to find nodes in the time-varying layouts to evaluate the layout stability. Since the label of each node will be depicted when hovering over the node, all the participants can easily find the corresponding nodes and the time spent was recorded as the metric to evaluate the layout stability. In the second part, participants were asked to observe particular nodes and answer questions, aiming to identify whether the participants can correctly compare the size of nodes. In the last part, all the participants were asked to sort the three methods from 'easy' to 'difficult' in terms of locating a particular node and comparing particular nodes, based on their experience during the previous two parts of the user study. After sorting, participants are asked whether the outline of the cell helps them to answer the two questions in this part. The questionnaire is described in Table 1. In order to avoid a preconceived impression, we divided all the participants into three groups (G_1, G_2, G_3) and assigned them the three visualization methods in different orders (G_1: Treemap → OST → VT; G_2: OST →



(a) Day 1 **(b)** Day 2 **(c)** Day 3 **(d)** Day 4 **(e)** Day 5

(f) Day 6 **(g)** Day 7 **(h)** Day 8 **(i)** Day 9 **(j)** Day 10

Fig. 11 Additional plots on the finished WIP lots information (per day) at the work center level. As illustrated, the layout stability is preserved in our OST method

Table 1 The questionnaire used in the user study

No.	Questions
Q1	Find the work center 'PROC_123'
Q2	Find the work center 'PHT_172'
Q3	Find the work center 'WET_220'
Q4	At T4, which work center under the work area 'OVN' has the largest AvgWIPLots value?
Q5	When does 'OVN_138' have the largest AvgWIPLots value?
Q6	When does 'CVD_75' have the smallest AvgWIPLots value?
Q7	According to your experience, which method is easy to locate a particular node? Sort the three methods from easy to difficult. Does the outline of the cell help you locate the node? (Asked after sorting)
Q8	According to your experience, which method is easy to compare the size of a particular node? Sort the three methods from easy to difficult. Does the outline of the cell help you compare the size? (Asked after sorting)

Table 2 The user study result for Q7 & Q8 (frequency)

Q7	Easy	Medium	Difficult
Treemap	0	0	12
VT	4	8	0
OST	8	4	0

Q8	Easy	Medium	Difficult
Treemap	8	4	0
VT	0	2	10
OST	4	6	2

Treemap → VT; G_3: VT → Treemap → OST;). The results of all the questions are shown in Fig. 12 and Table 2.

For locating particular nodes as illustrated in Fig. 12 (left), the time spent in VT and OST is much less than that in the treemap and participants spend similar time in VT and OST. This observation is confirmed by the subjective impression survey of the participants in *Q*7. All the participants believe that the treemap is not easy for positioning and 2/3 people think OST is easier to locate a particular node than VT as shown in Table 2. These results indicate that a stabler layout contributes to locating particular nodes. According to the feedback (3 participants), the outline change in VT makes them confused and they prefer orthogonal segments as their orientations are the same. For the rest participants (9 participants), the outline change has no effect on them. For *Q*7, we believe that the layout stability is the most important factor as all the

participants ranked the squarified treemap as the last one. Meanwhile, an outline formed by segments with only two orientations is a little bit easier for location, and we treat it as the secondary factor.

Figure 12 (right) illustrates the accuracy of the second part of the user study where participants were asked to compare the node size. Treemap has a better performance as it uses rectangles to represent nodes which is easier for comparing size. Our OST has slightly better performance than VT in $Q4$ and $Q6$. We believe that this is due to our orthogonal design in partitioning the canvas. The results of $Q8$ indicate that participants prefer rectangular shapes when comparing the size while nobody likes the polygonal layout in VT. 7 participants believe that the rectangular outline helps them to compare the size, while the rest participants feel that all the outlines are not easy to judge or compare the size.

In all, the results of the user study show that the proposed OST can preserve the layout stability and the orthogonal rectangular outline is slightly helpful for easy positioning. The orthogonal rectangular outline is not the best choice for size comparison, but is still acceptable as no outline can guarantee the accuracy of the comparison.

8 Discussion

Orthogonality The orthogonal layout has been proved to be suitable for human beings in the explicit hierarchy visualization (e.g., node-link diagram) and network visualization (Burch et al. 2011; Kieffer et al. 2016). When it comes to the implicit hierarchy visualization, previous work which divides existing treemap elements into orthoconvex and L-shape to preserve an aspect ratio constraint is also proposed (de Berg et al. 2014). In terms of orthogonality, our OST aims to preserve the stability of dynamic data while delivering a tidier layout with the help of orthogonal shapes. We did not deliberately maintain the aspect ratio in OST, as we used the concept of centroidal Voronoi diagram which ensures a good aspect ratio.

Entire Computation Time The entire computation time of OST is determined by the computation time per iteration discussed in Sect. 5.2 and the number of iteration needed for convergence discussed in Sect. 5.3. In practice, we set the maximum iteration number j_{max} to 500 and set the area error threshold for convergence $E_{threshold}$ within the range [0.01, 0.1]. When the users have lower priority on the representation accuracy, they can have a large $E_{threshold}$, such as 0.1. In this case, according to Fig. 6, OST only needs less than 50 iterations and the entire computation time of OST is only one-third of that of VT as VT needs around 150 iterations to reach the threshold. Taking the GlobalGDP and Flare datasets which are used in Sect. 5 as examples. When $E_{threshold} = 0.1$, OST needs 40.12 ms and 309.25 ms to generate one layout for GlobalGDP and Flare, respectively, while VT needs 117.05 ms and 1108.22 ms. We can find that OST needs much less time compared with VT. On the other hand, if $E_{threshold}$ is set to a small value (i.e., 0.01), more iterations are needed for both methods before convergence or maybe the j_{max} is reached. In this case, VT has a probability of converging faster than OST and needs less entire computation time compared with OST. In our experiments, when we set $E_{threshold} = 0.01$, OST needs 226.43 ms and 2161.32 ms, while VT needs 250.11 ms and 2375.19 ms, respectively.

Programming Language Compared with the original running time of the VT provided by Nocaj and Brandes (2012) (in Java), both our algorithm and the Voronoi treemap package need much more time (in JavaScript). Since no hardware-accelerated code is used in all cases, we believe that this difference is due to the capabilities of different programming languages. It would be one interesting future work to increase the efficiency by implementing the algorithm in Java in the backend and then displaying the results in JavaScript in the frontend.

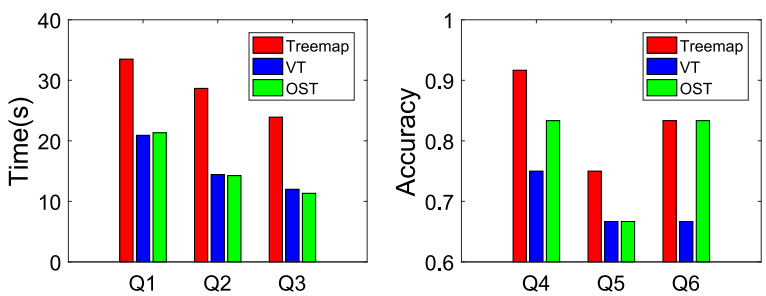


Fig. 12 The user study results for Q1 to Q6

Objective The objective of OST is to preserve the layout stability for dynamic data while delivering a tidier layout by using orthogonal rectangles. As known to all, the advantage of VT is the stability but not tidiness as its polygonal shape, while the advantage of treemap is its tidiness with rectangles but not stable for dynamic data. Based on the results of the user study, our OST has almost the same stability performance compared to VT, and outperforms the treemap (Fig. 12(Q1–Q3) and Table 2 Q7). Meanwhile, participants believe that OST is easier to compare the size of shape than VT, which indirectly explains the tidiness of the layout (Fig. 12(Q4–Q6) and Table 2 Q8). In general, we believe that the proposed OST keeps a balance between VT and treemap method, achieving the original design goal.

Drawbacks & Future Work One drawback of our OST is the large fluctuations during the iteration as shown in Fig. 6, due to the area error in the non-monotonously decreasing convergence. Although our initialization and self-adaption strategies ensure an acceptable small value, a mechanism to smooth the curve is needed to guarantee the convergence. Another drawback is the occasionally appeared thin strips which make some elements odd, which also needs to be handled in our future work. Lastly, both OST and VT have the problem of not being able to completely accurately represent the value of each cell with its area. Although the update procedure described in Algorithm 4 aims to ease this problem, it is still important to improve stability without losing accuracy in some use cases. Some relevant works are designed for this purpose, including the incremental treemap (Sondag et al. 2018) and the greedy insertion treemap (Vernier et al. 2018). However, both of them have some disadvantages. For example, the incremental treemap needs large computation as they use the greedy search method. While the greedy insertion treemap is based on a specific structure. Moreover, both of them are not considering the flexibility for neighborhood design which is one advantage of our OST and VT. Hence, how to modify our OST such that it can improve the stability without losing accuracy is an interesting future direction.

9 Conclusions

To conclude this work, we have described a novel site-based implicit hierarchy visualization method with nested orthogonal rectangles for dynamic data to preserve layout stability. Moreover, OST has a simplified layout, as segments have only two orientations (vertical and horizontal) for dynamic data at all time steps. Experimental results have shown that OST requires less computation time and converges faster than VT, while having a comparable aspect ratio to the squarified treemap. We have achieved the original design objective and users have given positive feedback.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s12650-022-00830-1.

Acknowledgements This work was partially supported by the A*STAR Cyber-Physical Production System (CPPS)-Towards Contextual and Intelligent Response Research Program, under the RIE2020 IAF-PP GrantA19C1a0018, and Model Factory @SIMTech. This work is also partially supported by a Grant MOE 2017-T1-001-053-04 from Ministry of Education, Singapore.

References

Armitage J (2014) Method and system for generating a columnar tree map. 8854371. https://www.freepatentsonline.com/8854371.html

Auber D, Huet C, Lambert A, Renoust B, Sallaberry A, Saulnier A (2013) Gospermap: using a gosper curve for laying out hierarchical data. IEEE Trans Visual Comput Graphics 19(11):1820–1832

Aurenhammer F (1987) Power diagrams: properties, algorithms and applications. SIAM J Comput 16(1):78–96

Aurenhammer F, Edelsbrunner H (1984) An optimal algorithm for constructing the weighted Voronoi diagram in the plane. Pattern Recogn 17(2):251–257

Balzer M, Deussen O (2005) Voronoi treemaps. In: Proc. INFOVIS. IEEE, pp 49–56

Bederson BB, Shneiderman B, Wattenberg M (2002) Ordered and quantum treemaps: making effective use of 2d space to display hierarchies. ACM Trans Graphics 21(4):833–854

Bethge J, Hahn S, Döllner J (2017) Improving layout quality by mixing treemap-layouts based on data-change characteristics. In: Proceedings of the conference on vision, modeling and visualization, pp 69–76

Bostock M, Ogievetsky V, Heer J (2011) D^3 data-driven documents. IEEE Trans Visual Comput Graphics 17(12):2301–2309

Bruls M, Huizing K, Van Wijk JJ (2000) Squarified treemaps. In: Data visualization 2000. Springer, pp 33–42

Burch M, Konevtsova N, Heinrich J, Hoeferlin M, Weiskopf D (2011) Evaluation of traditional, orthogonal, and radial tree diagrams by an eye tracking study. IEEE Trans Visual Comput Graphics 17(12):2440–2448

Burke EK, Kendall G, Whitwell G (2004) A new placement heuristic for the orthogonal stock-cutting problem. Oper Res 52(4):655–671
Chen Y, Du X, Yuan X (2017) Ordered small multiple treemaps for visualizing time-varying hierarchical pesticide residue data. Vis Comput 33(6–8):1073–1084
de Berg M, Speckmann B, van der Weele V (2014) Treemaps with bounded aspect ratio. Comput Geom 47(6):683–693
Du Q, Faber V, Gunzburger M (1999) Centroidal Voronoi tessellations: applications and algorithms. SIAM Rev 41(4):637–676
Duarte FS, Sikansi F, Fatore FM, Fadel SG, Paulovich FV (2014) Nmap: a novel neighborhood preservation space-filling algorithm. IEEE Trans Visual Comput Graphics 20(12):2063–2071
Feng C, Gong M, Deussen O, Huang H (2019) Treemapping via balanced partitioning. Proc. computational visual media (CVM'19)
Fiedler C, Scheibel W, Limberger D, Trapp M, Döllner J (2020) Survey on user studies on the effectiveness of treemaps. In: Proceedings of the 13th international symposium on visual information communication and interaction, pp 1–10
Fischer F, Fuchs J, Mansmann F (2012) Clockmap: enhancing circular treemaps with temporal glyphs for time-series data. In: Eurographics conference on visualization. Eurographics Association, pp 97–101
Fortune S (1987) A sweepline algorithm for Voronoi diagrams. Algorithmica 2(1–4):153
Gavrilova M (1998) Proximity and applications in general metrics. Ph.D. thesis, University of Calgary
Ghoniem M, Cornil M, Broeksema B, Stefas M, Otjacques B (2015) Weighted maps: treemap visualization of geolocated quantitative data. In: Visualization and data analysis 2015, vol 9397. International Society for Optics and Photonics, p 93970
Görtler J, Schulz C, Weiskopf D, Deussen O (2018) Bubble treemaps for uncertainty visualization. IEEE Trans Visual Comput Graphics 24(1):719–728
Gotz D (2011) Dynamic voronoi treemaps: a visualization technique for time-varying hierarchical data. Phys Rev A 30(2):150–156
Graham M, Kennedy J (2010) A survey of multiple tree visualisation. Inf Vis 9(4):235–252
Hahn S, Trümper J, Moritz D, Döllner J (2014) Visualization of varying hierarchies by stable layout of Voronoi treemaps. In: Proc. IVAPP. IEEE, pp 50–58
Itoh T, Yamaguchi Y, Ikehata Y, Kajinaga Y (2004) Hierarchical data visualization using a fast rectangle-packing algorithm. IEEE Trans Visual Comput Graphics 10(3):302–313
Kieffer S, Dwyer T, Marriott K, Wybrow M (2016) Hola: human-like orthogonal network layout. IEEE Trans Visual Comput Graphics 22(1):349–358
Kobayashi A, Misue K, Tanaka J (2012) Edge equalized treemaps. In: Proc. IV. IEEE, pp 7–12
Kong N, Heer J, Agrawala M (2010) Perceptual guidelines for creating rectangular treemaps. IEEE Trans Visual Comput Graphics 16(6):990–998
Lai Y-J, Cheng P-H, Lu L-W, Rau C-R (2015) A visualization by innovated squarified treemap for somatosensory data analysis. In: 2015 IEEE 4th global conference on consumer electronics (GCCE). IEEE, pp 587–588
Liang J, Nguyen QV, Simoff S, Huang ML (2012) Angular treemaps-a new technique for visualizing and emphasizing hierarchical structures. In: Proc. IV. IEEE, pp 74–80
Long LK, Hui LC, Fook GY, Zainon WMNW (2017) A study on the effectiveness of tree-maps as tree visualization techniques. Proc Comput Sci 124:108–115
Mönch L, Fowler JW, Dauzere-Peres S, Mason SJ, Rose O (2011) A survey of problems, solution techniques, and future challenges in scheduling semiconductor manufacturing operations. J Sched 14(6):583–599
Monostori L (2014) Cyber-physical production systems: roots, expectations and r & d challenges. Procedia Cirp 17:9–13
Nocaj A, Brandes U (2012) Computing Voronoi treemaps: faster, simpler, and resolution-independent. In: Computer graphics forum, vol 31. Wiley Online Library, pp 855–864
Scheibel W, Limberger D, Döllner J (2020) Survey of treemap layout algorithms. In: Proceedings of the 13th international symposium on visual information communication and interaction, pp 1–9
Scheibel W, Trapp M, Limberger D, Döllner J (2020) A taxonomy of treemap visualization techniques. In: VISIGRAPP (3: IVAPP), pp 273–280
Scheibel W, Weyand C, Bethge J, Döllner J (2021) Algorithmic improvements on Hilbert and Moore treemaps for visualization of large tree-structured datasets
Scheibel W, Weyand C, Döllner J (2018) Evocells-a treemap layout algorithm for evolving tree data. In: VISIGRAPP (3: IVAPP), pp 273–280
Schulz H-J (2011) Treevis.net: a tree visualization reference. IEEE Comput Graphics Appl 31(6):11–15
Schulz H-J, Hadlak S, Schumann H (2011) The design space of implicit hierarchy visualization: a survey. IEEE Trans Visual Comput Graphics 17(4):393–411
Shneiderman B (1992) Tree visualization with tree-maps: 2-d space-filling approach. ACM Trans Graphics 11(1):92–99
Shneiderman B, Wattenberg M (2001) Ordered treemap layouts. In: Proc. INFOVIS. IEEE, pp 73–78
Sondag M, Speckmann B, Verbeek K (2018) Stable treemaps via local moves. IEEE Trans Visual Comput Graphics 24(1):729–738
Sud A, Fisher D, Lee H-P (2010) Fast dynamic Voronoi treemaps. In: Proc. ISVD. IEEE, pp 85–94
Tak S, Cockburn A (2013) Enhanced spatial stability with Hilbert and Moore treemaps. IEEE Trans Visual Comput Graphics 19(1):141–148
Tu Y, Shen H-W (2007) Visualizing changes of hierarchical data using treemaps. IEEE Trans Visual Comput Graphics 13(6):1286–1293
Vernier EF, Comba JLD, Telea AC (2018) A stable greedy insertion treemap algorithm for software evolution visualization. In: 2018 31st SIBGRAPI conference on graphics, patterns and images (SIBGRAPI). IEEE, pp 158–165
Vernier E, Sondag M, Comba J, Speckmann B, Telea A, Verbeek K (2020) Quantitative comparison of time-dependent treemaps. In: Computer graphics forum, vol 39, pp 393–404. Wiley Online Library

Wang YC, Liu JG, Lin F, Seah HS (2020) Generating orthogonal Voronoi treemap for visualization of hierarchical data. In: Proc. CGI. ACM
Wang Y, Chen L (2015) Two-dimensional residual-space-maximized packing. Expert Syst Appl 42(7):3297–3305
Wang G, Nakanishi T, Fukuda A (2016) 2-d layout for tree visualization: a survey. In: Proc. MATEC Web of conferences, vol 56. EDP Sciences
Wang W, Wang H, Dai G, Wang H (2006) Visualization of large hierarchical data by circle packing. In: Proc. CHI. ACM, pp 517–520
Wang Y, Zhang Q, Lin F, Seah HS (2019) Engineqv: investigating external cause of engine failures based on geo-temporal association. In: 2019 IEEE Pacific visualization symposium (PacificVis). IEEE, pp 184–188
Wattenberg M (2005) A note on space-filling visualizations and space-filling curves. In: Proc. INFOVIS. IEEE, pp 181–186
Wood J, Dykes J (2008) Spatially ordered treemaps. IEEE Trans Vis Comput Gr 14(6)
Zhang C, Bard JF, Chacon R (2017) Controlling work in process during semiconductor assembly and test operations. Int J Prod Res 55(24):7251–7275