The Visual Computer
https://doi.org/10.1007/s00371-023-02864-4

ORIGINAL ARTICLE

PowerHierarchy: visualization approach of hierarchical data via power diagram

Yuyou Yao[1] · Tao Li[1] · Wenming Wu[1] · Gaofeng Zhang[2] · Liping Zheng[1]

Accepted: 2 April 2023

Abstract
Voronoi treemaps are widely used for hierarchical data visualization. Existing methods calculate the visualization layouts of hierarchical data by combining the proportion optimization of weights and Lloyd's method of sites. However, this may not only produce results with large area errors but also require more time consumption. Besides, the relative visualization position of the same data element between adjacent frames in dynamic hierarchical data may be changed abruptly, resulting in unclear visual results. To this end, we propose an efficient and topological structure preserved visualization approach, called PowerHierarchy, for visualizing hierarchical data. Firstly, an improved version of the power diagram computing algorithm is introduced to generate the visualization layouts of each data element in the hierarchy. Unlike random initialization, we construct a centroidal Voronoi tessellation as input and then use a Breadth-First traversing strategy to adapt the depth information to produce visual layouts of static hierarchical data. Based on this, an updating scheme is presented for visualizing dynamic hierarchical data, where previous results are iteratively fed as inputs to initialize current layouts. Besides, the external boundary sites and their subsites are projected onto the visual boundary and then moved into the visual region with the relative position preserved. Experimental results on several datasets demonstrate the efficiency, accuracy, and topology preservation advantage of our proposed visualization approach.

Keywords Power diagram · Treemap · Hierarchical data · Visualization

1 Introduction

Hierarchical data have been frequently used in our daily life and practical applications, e.g., digital document organization, class design in the software development process [1], etc. Time-dependent hierarchical data, also known as dynamic hierarchical data [2], have recently appeared in various fields, e.g., the real-time data monitoring system, etc. In view of the characteristic of hierarchical data, the rest, except dynamic hierarchical data, can be considered static. To completely capture the underlying information in these hierarchies, data visualization techniques are used to make them more accessible and understandable.

As an efficient visualization technique, Voronoi treemaps combine the classical treemaps with the ordinary Voronoi diagram or weighted Voronoi diagram (power diagram). This method recursively subdivides the visual region into nested polygonal subregions to produce visualization results, as shown in Fig. 1. In view of visualizing static hierarchical data, existing methods mainly involve (1) centroid optimization and (2) weight optimization. The fundamental principle of these methods [4] is to simultaneously increase or decrease the weights proportionally to the missing or excess area and optimize each site to its cell center by Lloyd's method [3, 5]. However, the weights optimized in this manner may cause tiny or even empty cells, resulting in a significant area error of subregions in the generated visualization results, as shown in the red and blue regions in Fig. 1. In addition, to the best of our knowledge, Lloyd's method is the simplest but not the best way for centroid optimization, which requires much more time consumption.

In terms of dynamic hierarchical data visualization via Voronoi treemaps, several variants extend the static hierarchical data visualization techniques. A dynamic Voronoi

✉ Liping Zheng
zhenglp@hfut.edu.cn

[1] School of Computer Science and Information Engineering, Hefei University of Technology, Hefei 230601, China

[2] School of Software, Hefei University of Technology, Hefei 230601, China

Published online: 09 May 2023

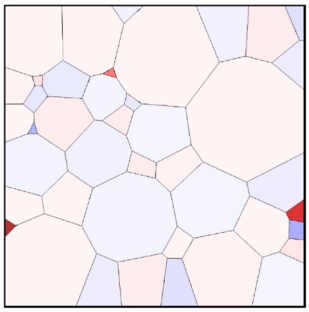


Fig. 1 In the results generated by the algorithm in [3], area errors in target size are indicated by color (red = too big, white = correct, blue = too small), where color denotes the degree to which the actual size of a target area deviates from its expected size

treemap (DVT) [6] is proposed to maintain a desirable aspect ratio for time-varying hierarchical data. However, it suffers from the instability of a large variance jump in the region position. Sud et al. [7] present a GPU-based approach for visualizing dynamic hierarchical data, where the additively weighted Voronoi diagram is calculated by GPU accelerating during frames, and the previous layouts are utilized to initialize current Voronoi treemaps. However, similar to DVT, the topology structures of visualization layouts produced by this method cannot be preserved in each frame, as shown in Fig. 13a–d. That is, the regions corresponding to the same data element may "jump" between adjacent frames.

Therefore, we focus on the limitations of previous work on computing Voronoi treemaps, and the topological structure preservation of layouts is taken into consideration. Unlike existing methods, we propose an efficient visualization approach called PowerHierarchy, for visualizing hierarchical data. To achieve it, we first borrow the idea of numerical optimization to calculate the layouts for static hierarchical data visualization instead of using proportion optimization and Lloyd's method. Specifically, we designed an improved version of the power diagram computing algorithm from [8]. A centroidal Voronoi tessellation (CVT) is calculated as input rather than random initialization to avoid a variety of corner cases (e.g., empty cells, etc.) for better computation stability. Based on this, an updating scheme is presented for visualizing dynamic hierarchical data. Like [7], the previous results (weights and sites) are fed as inputs to initialize current Voronoi treemaps in our updating scheme. However, the presence of these external boundary sites may cause a large variance in the regional location, resulting in the topological structures of layouts cannot be preserved. Hence, we design a projection method, combined with the initialization strategy, to preserve the topological structure of layouts during in-between changes. Those external boundary sites and their subsites are projected onto the visual boundary and then moved into the corresponding visual region. Experimental results on various static datasets demonstrate the effectiveness, efficiency, and stability of our method. Besides, the PowerHierarchy is also used to visualize the 2010–2019 global GDP and the 2019–2022 consumer price index. These dynamic hierarchical data results validate the topology preservation of visualization layouts between adjacent frames. We contribute the following:

- An improved version of the power diagram computing algorithm from [8], generating visualization results for static hierarchical data with better stability, efficiency, and accuracy.
- An updating scheme to visualize dynamic hierarchical data with the topological structure of visualization layouts is preserved, avoiding the possibility of the relative position of the same data element between adjacent frames changing abruptly.

Even though numerical optimization is used for power diagram computation and other applications, to the best of our knowledge, PowerHierarchy is the first method to apply it for visualizing hierarchical data. Using Newton's and the L-BFGS methods to compute the visualization layouts significantly improves the computational efficiency and accuracy of visualization results than state-of-the-art methods via Voronoi treemaps. Additionally, the topological structure preservation of layouts during frames is considered in PowerHierarchy, which allows it to generate smooth visualization layouts without "jump", which is superior to existing methods based on Voronoi treemaps.

The remainder of this paper is organized as follows. Section 2 briefly reviews related work. Section 3 introduces the preliminary power diagrams and Voronoi treemaps. PowerHierarchy for hierarchical data visualization is explained in Sect. 4. In Sect. 5, some experimental results on various datasets are provided to evaluate the efficiency and accuracy of PowerHierarchy, and some conclusions are given in Sect. 6.

2 Related work

In this section, we briefly review the hierarchical data visualization with Voronoi treemaps, and then the prior approaches to the computation of power diagrams are discussed. An exhaustive review of data visualization methods could be referred to [9, 10], and we focus on the visualization methods with Voronoi treemaps in the following.

2.1 Static hierarchical data visualization

Treemaps [9, 11] have been widely used to visualize hierarchical data, which apply the recursive space-filling approach to subdivide the 2-dimensional plane into nested subregions.

Each node in the hierarchy has a name and associated value. Over the past three decades, studies have introduced various techniques to produce visualization layouts, e.g., Slice and Dice [12], Squarified treemaps [13], circular treemaps [14], ordered treemaps [15, 16], and bubble treemaps [17], etc. Voronoi treemaps, a powerful technique for hierarchical data visualization with the polygonal visual region, have received much attention from researchers.

Balzer and Deussen [4] introduce the basic iterative algorithm for Voronoi treemaps. Lloyd's method is utilized to compute the centroidal Voronoi diagrams, coupled with the weight adaption method to control cell areas. The weights of sites are increased or decreased proportionally to the missing or excess area. However, small or empty Voronoi cells may appear this way, affecting the computation stability of Voronoi treemaps. Nocaj and Brandes [5] propose a resolution-independent analytic method to compute the power diagrams, which improves the weight optimization scheme to reduce the number of iterations and achieve fast computation. Nevertheless, this method limits the weight of a new site to the minimum distance of the cell it belongs to and its maximum weight, which may cause many Voronoi cells to be small. Hahn et al. [3] introduce a weaker limit weight optimization method, which limits the weight of a new site to the minimum distance of the nearest neighbor site. Though this method could circumvent too small Voronoi cells, some cells still have large area errors, as shown in Fig. 1.

The essence of these methods to visualize static hierarchical data is the proportion optimization for weights and Lloyd's method for sites [3–5], as shown in Table 1. However, this may require more iteration and cannot generate high-accuracy visualization results, as shown in Fig. 1. In this paper, we first borrow the idea of numerical optimization [8] and apply it to hierarchical data visualization. Unlike previous work [3], we utilize Newton's method for weight optimization to produce a more accurate visual layout. The L-BFGS method with super-linear convergence is used for site optimization rather than Lloyd's method with linear convergence. Moreover, to improve the computation stability of our method, we calculate a CVT before the weight and site optimization, which is fed as input to produce the visual layouts.

2.2 Dynamic hierarchical data visualization

Dynamic hierarchical data is also referred to as time-dependent hierarchical data, where the structure and value of data elements change over time. A straightforward visualization way is to recalculate the layout as data changes, but it is time-consuming. Several approaches [18, 19] have been introduced to improve the stability and efficiency of visualizing dynamic hierarchical data [2]. Here we focus on these methods using Voronoi treemaps.

For visualizing dynamic hierarchical data based on Voronoi treemaps, Gotz et al. [6] present a DVT technique to maintain a desirable aspect ratio. They randomly initialize the Voronoi treemap as data changes and then optimize the weights and sites to produce visualization results. Nevertheless, there may be a large variance jump in the regional position when rendering dynamic data, and the optimization based on random initialization is time-consuming. Sud et al. [7] developed a GPU-based technique for visualizing dynamic hierarchical data via additively weighted Voronoi diagrams. The previous layouts are taken as inputs to initialize current Voronoi treemaps. This method achieves better computational efficiency due to the parallel calculating of the additively weighted Voronoi diagram and the initialization with previous layouts. However, the topological structures of visualization layouts of two adjacent temporal data sequences generated by [7] are not guaranteed to preserve, which causes the results to be unclear and confusing, e.g., the "jump" in Fig. 13b–d.

This paper considers topological structure preservation, and we introduce an updating scheme for visualizing dynamic hierarchical data. Similar to [7], the current Voronoi treemaps are initialized with previous layouts. More importantly, these external boundary sites and their subsites are projected onto the visual boundary, then moved into the visual region with relative position preserved.

2.3 Power diagram & its computation

As a significant extension of the Voronoi diagram, the power diagram assigns a weight to each site to achieve the capacity-constrained characteristic. Centroidal power diagram (CPD) defines a particular power diagram where each site is located at the mass center of its power cell, thereby producing a good 1:1 aspect ratio. Based on this, combined with the capacity constraints, a centroidal capacity-constrained power diagram (CCCPD) is obtained, which could be applied to hierarchical data visualization.

Several methods have been introduced to calculate the CCCPD. Balzer [20] presents a false-position method to optimize the weights, augmented by Lloyd's method to optimize the sites, producing the CCCPD. Owing to the one-by-one iteration strategy, this method is inefficient. De Goes et al. [21] utilize Newton's method to optimize weights, which significantly improves the computational efficiency of CCCPD. Furthermore, Xin et al. [8] apply the L-BFGS method with super-linear convergence to optimize sites rather than Lloyd's method with linear convergence. Recently, Zheng et al. [22] extended the capacity constraints to the general cases and introduced the hybrid capacity-constrained centroidal power diagram (HCCCPD). With the development of hardware, existing methods take advantage of GPU to compute the power diagram [7]. Zheng et al. [23] provide a GPU-CPU

Table 1 Briefly review of existing methods for hierarchical data visualization via Voronoi treemaps

Method	Data	Optimization		Topology	
		Weight	Site	Initial	Preserve
Balzer [4]	Static	Proportion	Lloyd	✗	✗
Nocaj [5]	Static	Proportion	Lloyd	✗	✗
Hahn [3]	Static	Proportion	Lloyd	✗	✗
Sud [7]	Dynamic	Proportion	Lloyd	Previous	✗
Gotz [6]	Dynamic	Proportion	Lloyd	Random	✗
Ours	Static & Dynamic	Newton	L-BFGS	Previous	✓

hybrid algorithm to accelerate the construction of the power diagram, thereby significantly improving the computational efficiency.

In view of visualizing hierarchical data via power diagrams, existing methods extend the algorithm in [4] to improve the stability of visual layout computation, as shown in Table 1. However, these methods optimize weights proportionally and update sites by Llody's method, which causes more time consumption and large area error. Motivated by numerical optimization in previous work, we introduce an improved version of the power diagram computing algorithm from [8] to calculate the visualization layouts.

Overall, to highlight the limitations of previous work and emphasize the motivation of our work, Table 1 provides a brief review and comparison of existing methods for hierarchical data visualization via Voronoi treemaps. For visualizing static hierarchical data, previous work has used proportional weight optimization coupled with Lloyd's method to generate visualization layouts, which may cause small or even empty cells with large area errors, as shown in Fig. 1. In view of dynamic hierarchical data, some techniques have designed various initialization strategies to speed up the visualization layout computation for each frame, but the topology is not considered, resulting in the visualization layouts being unclear and confusing. To address these problems, we provide a visualization approach called PowerHierarchy for hierarchical data visualization. On the one hand, we propose an improved version of the power diagram computation algorithm in [8], using Newton's method for weight optimization and the L-BFGS method for site optimization to quickly generate accurate visualization layouts of static hierarchical data. On the other hand, we introduce an updating scheme for dynamic hierarchical data visualization, where the previous results are taken as initialization of current layouts, and these external boundary sites and their subsites are moved into the current visual region with the relative position preserved, producing visualization layouts with topology preservation.

3 Preliminary

In this section, we discuss the definition of the power diagram and its variants. The Voronoi treemaps are then explained in detail.

3.1 Voronoi diagram & power diagram

The Voronoi diagram (also called "Voronoi tessellation") [24] defines a spatial subdivision of a given domain $\Omega \subset R^2$. Given a set $S = \{s_i\}_{i=1}^n$ of n distinct points (also called "sites") in the plane. Based on the Euclidean distance, the Voronoi diagram is a partition of the domain Ω into n regions $V = \{V(s_i)\}_{i=1}^n$ without producing holes or overlaps. Each region $V(s_i)$ of the site s_i, called Voronoi region, is defined as:

$$V(s_i) = \{s \in \Omega | \|s - s_i\| \leq \|s - s_j\|, \forall j \neq i\} \tag{1}$$

As an extension of Voronoi diagrams, power diagrams [25] introduce weights $W = \{w_i\}_{i=1}^n$ to sites, that is, a parameter w_i is assigned to each site s_i. Each region $P(s_i)$ (called "power cell") is redefined as:

$$P(s_i) = \{s \in \Omega | \|s - s_i\|^2 - w_i \leq \|s - s_j\|^2 - w_j, \forall j \neq i\} \tag{2}$$

where $d(s, s_i) = \|s - s_i\|^2 - w_i$ is redefined as the power distance. The power diagram degenerates to a Voronoi diagram when the weights of all sites are equal [25].

3.2 CCCPD

The area of a power cell depends on the relative position of the site and its neighbors. Power diagrams introduce a weight to each site, having precise capacity constraints. By imposing the centroid and capacity constraints on each site, we can obtain a CCCPD. That is, each site s_i is located at the mass center of its power cell $P(s_i)$, and the capacity (e.g., area, etc.) m_i is equal to the target value c_i:

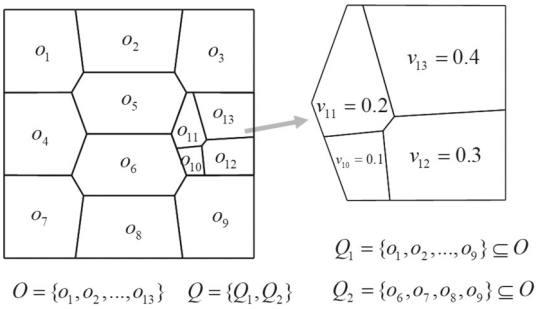


Fig. 2 Exemplification of a Voronoi treemap in the squared visualization domain, where $O = \{o_1, o_2, \cdots, o_{13}\}$ represents the set of data nodes, and the polygonal regions are the corresponding visualization layouts

$$\begin{cases} s_i = s_i^* = \frac{\int_{p(s_i)} s\rho(s)ds}{\int_{P(s_i)} \rho(s)ds} \\ m_i = \int_{P(s_i)} \rho(s)ds = c_i \\ \sum_{i=1}^{n} m_i = \int_{\Omega} \rho(s)ds \end{cases} \tag{3}$$

where $\rho(s)$ represents the density and s_i^* denotes the centroid of the power cell $P(s_i)$. Aurenhammer et al. [26] introduce the formulation $F(S, W)$ of the optimal power diagram, and the gradient of $F(S, W)$ can be shown as [21]:

$$\begin{cases} F(S, W) = \sum_{i=1}^{n} \int_{P(s_i)} \|s - s_i\|^2 ds - \sum_{i=1}^{n} w_i(m_i - c_i) \\ \nabla_{w_i} F(S, W) = c_i - m_i \\ \nabla_{w_i} m_j = -\frac{1}{2} \cdot \frac{|e_{ij}^*|}{|e_{ij}|} \\ \nabla_{s_i} F(S, W) = 2m_i(s_i - s_i^*) \end{cases} \tag{4}$$

where e_{ij} represents the regular edge between two adjacent sites s_i and s_j, and e_{ij}^* refers to the dual edge separating the power cell $P(s_i)$ and $P(s_j)$. Notably, the CVT could be computed based on Eq. (4) by setting all weights of sites equal [27].

3.3 Voronoi treemap

A Voronoi treemap [4] is the recursive subdivisions of a region into the cells of a centroidal Voronoi diagram (CPD is considered in this paper), and that is defined in the following. Let $O = \{o_1, o_2, ..., o_n\}$ denote a set of objects, associated with positive values $v_i \in R > 0, i = 1, 2, ..., n$. Based on the description in [5], the additive extension to subsets of $Q \subseteq O$ is defined: $V(Q) = \sum_{i:o \in Q} v_i$.

The hierarchical partition of O is a rooted tree $T = (Q, E; r)$, where nodes Q represents the subset of O, and edges E explains the set inclusion. The root $r \in Q$ represents O, and the leaf nodes are the singleton sets $\{o_i\}_{i=1}^{n}$. Each inner node expresses the set formed by the union of its children sets.

The hierarchical partition is represented by the Voronoi treemap. The bounded region V_Q, represented the entire objects, is subdivided by a set of centroidal Voronoi diagrams, in which these Voronoi cells are recursively subdivided such that the leaf nodes are denoted by these cells with an area proportional to their value. The children of each $q \in Q$ are represented by $child(q)$, and the target area of $c \in child(q)$ is $A(V_q) \cdot \frac{v(c)}{v(q)}$, where $A(V_q)$ is the area of the region V_q, $v(c)$ and $v(q)$ are the value of data node.

4 PowerHierarchy for hierarchical data visualization

The majority of PowerHierarchy is static hierarchical data visualization (Sect. 4.2) and dynamic hierarchical data visualization (Sect. 4.3). Before that, the problem of visualizing hierarchical data is discussed (Sect. 4.1).

4.1 Problem statement

Voronoi treemaps recursively divide the primal region into nested subregions, as indicated in Sect. 3.3. Weighted Voronoi diagrams address the aspect ratio and area requirement instead of using the ordinary Voronoi diagram. Two generalizations, the additively weighted Voronoi diagram and the power diagram, could be formed by imposing the weight characteristic on the ordinary Voronoi diagram. The bisector shape of two neighboring power cells is a hyperbolic curve in the former, which may cause difficulty in the Voronoi diagram construction. In contrast, power diagrams produce straight lines that resemble the ordinary Voronoi diagram. Therefore, we concentrate on visualizing hierarchical data using the power diagram in this paper.

Hierarchical data could be classified into two types based on their characteristic: static hierarchical data and dynamic hierarchical data. The structure and values of data elements change in the latter, in contrast to the former, where they do not. However, regardless of whether the hierarchical data is static or dynamic, the fundamental components of data visualization are the following:

(1) *Area constraint* Power diagrams are used in hierarchical data visualization to convey data information, with the area size of each power cell corresponding to the value of the relative data element. That is, the desired size A_i^{target} of a data node in the hierarchy is the same as the area size A_i^{current} of its power cell $P(s_i)$: $A_i^{\text{current}} = A_i^{\text{target}}$.
(2) *Centroid constraint* Previous work use the CVT to recursively divide the primal region into nested subregions to address the high aspect ratio of treemaps, which can enhance the readability of visual results. Therefore, the centroid constraint is imposed on the layout computation. That is, the CPD (see Sect. 3.2) is adopted in our work.

4.2 Visualization of static hierarchical data

Then, an improved version of the power diagram computing algorithm from [8] is introduced in detail to visualize static hierarchical data.

4.2.1 Static hierarchical data

Static hierarchical data is frequently utilized in our daily life, i.e., the organization of a company. The essence feature of this data is that the structure and value of data elements never change. Static hierarchical data visualization based on Voronoi treemaps mainly focuses on calculating the visual layout of each data node in the visualization region, satisfying the area and centroid constraints.

Existing methods [3–5] increase or decrease the weights proportionally to the missing or excess area and optimize each site to its cell center by Lloyd's method. However, these methods may cause large area errors in visualization layouts (as shown in Fig. 1), and more time consumption is also required. To this end, we introduce a visualization approach called PowerHierarchy, which recursively computes the power diagram of each data node in the hierarchy, and power cells are denoted as the visualization layouts.

4.2.2 Visualization layout computation

In this paper, we borrow the idea of numerical optimization and apply it to hierarchical data visualization layout computation in the PowerHierarchy. To achieve it, we introduce an improved version of the power diagram computing algorithm from [8], which generates the layout of static hierarchical data progressively based on Eq. (4). Specifically, a CVT rather than random initialization is taken as input. Then the following two steps are iteratively performed: (1) optimizing the weights to comply with the *area constraint* by Newton's method, and (2) optimizing the sites to satisfy the *centroid constraint* by the L-BFGS method. Based on this, the Breadth-First traversing strategy adapts the depth information, thereby producing the visualization layout of static heirarchical data. The pseudo-code of visualization layout computation is given in Algorithm 1, and the relevant computation process is presented in Fig. 3. Notably, the power diagram computing algorithm is experimentally proved super-linear convergence, and more detailed implementations could be found in [8].

(1) *Weight optimization* The area size of each power cell is strongly related to the weights of its neighbors. Therefore, to adhere to the *area constraint* in Sect. 4.1, the first step is optimizing the weight of each site, that is, $A_i^{\text{current}} = A_i^{\text{target}}$. Similar to [21], Newton's method is applied for weight optimization, and the Hessian matrix of $F(S, W)$ could be calculated based on Eq. (4), where m_i is the current area A_i^{current} and c_i is the target area A_i^{target} of power cell $P(s_i)$.

(2) *Position optimization* Due to the separation of optimization of the weights and sites, Nocaj and Brandes [5] point out that the L-BFGS method may be applied to compute the visualization layout. Still, its influence on the power diagram and the area requirements is not guaranteed. Xin et al. [8] experimentally prove the super-linear convergence of the L-BFGS method for computing the centroidal power diagram, which is empirically faster than Lloyd's method with linear convergence. Therefore, we utilize the L-BFGS method for optimizing the position of each site to satisfy the *centroid constraint*, and the gradient of $F(S, W)$ with respect to the site s_i is given in Eq. (4).

(3) *Improvement* The primary power diagram computing algorithm in [8] employs random initialization, which suffers from various corner cases, e.g., empty cases. To circumvent these corner cases, we construct a CVT based on random initialization as input to improve the stability of the layout computation of static hierarchical data. Notably, there is no need for additional code for the CVT, and all that is required is to set the weights of all sites to be equal. Our work sets the termination condition for the CVT calculation to 10^{-3}. Therefore, the CVT could be generated using the L-BFGS method based on Eq. (4). Although the CVT computation consumes more time, we observe that timing only takes up less than 10% of the total computation time for the visualization layouts. Moreover, these corner cases, e.g., small or empty cells, could be avoided, improving computational stability.

Algorithm 1 Visualization layout computation of static hierarchical data

Require: visual boundary Ω, hierarchical data r, termination condition ϵ
Ensure: visual layouts (Power diagram (S, W))
$q \leftarrow r$ //data node queue
while $!q.empty()$ **do**
  $node \leftarrow q.front(); q.pop()$
  $n_{node} \leftarrow child(node).size()$
  for $i = 1, 2, \cdots, n_{node}$ **do** // area constraints
    $c_i \leftarrow \frac{v(child(node)[i])}{v(node)} \cdot A(V_{node})$
  end for
  S_{node} to be n_{node} randomly generated sites
  Compute CVT layout based on Eq. (4)
  while $\delta > \epsilon$ **do** // layout computation of a data node
    Update weights W by Newton's method
    Compute gradients $\nabla_{s_i} F(S_{node}, W_{node})$ in Eq. (4)
    Estimate the step-size for S_{node} by a line-search
    Update S_{node} and construct power diagram
    Compute $A_i^{current}$ and A_i^{target}
    $\delta \leftarrow \max\{|A_i^{current} - A_i^{target}|\}$
  end while
  $q \leftarrow child(node)[i]$, for $i = 1, \cdots, n$
end while

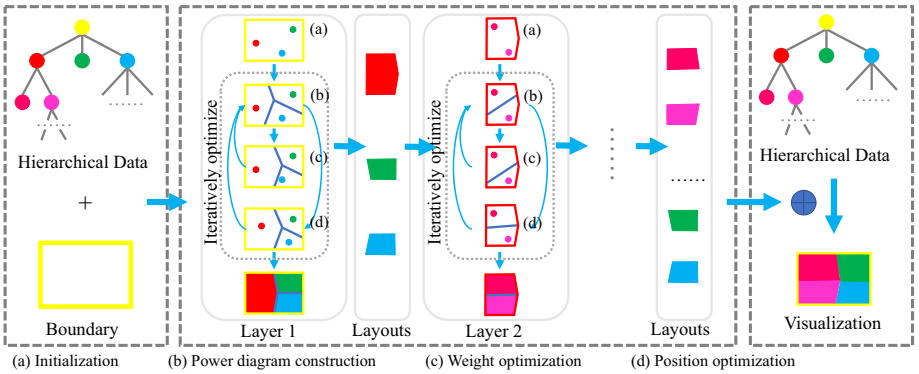


Fig. 3 Visualization layout computation of static hierarchical data, where a CVT is calculated as input rather than a randomly initialized power diagram

Although numerical optimization has been used in various applications, to the best of our knowledge, it is the first time to be applied for visualizing hierarchical data. Compared to previous work [3–5], PowerHierarchy could generate visualization layouts with smaller area errors in less time. Therefore, the proposed algorithm could be easily extended to visualize dynamic hierarchical data.

4.3 Visualization of dynamic hierarchical data

Considering dynamic hierarchical data based on Algorithm 1, PowerHierarchy provides an updating scheme to compute the visualization layout with the topology structure preserved.

4.3.1 Dynamic hierarchical data

Dynamic hierarchical data is commonly used in daily life, e.g., the annual visualization of global GDP, etc. A common feature in these data is that the structures and values of data elements change over time. Therefore, the visualization of dynamic hierarchical data is an increasingly essential field of data visualization.

The data tree structure and the values of data nodes are two aspects that play a significant role in dynamic hierarchical data. To be more specific, dynamic hierarchical data can be classified into three categories depending on these two characteristics, which are illustrated below with the organization of computer documents as an example.

- **Case 1** The value of a data node changes, but the structure of the data tree remains unchanged, e.g., modifying some digital documents.
- **Case 2** The structure of the data tree changes, but the values of data nodes stay constant, e.g., shifting digital documents to different directories.
- **Case 3** The values of data nodes and the structure of the data tree alter, e.g., modifying some digital documents and then moving them to other directories.

The goal of dynamic hierarchical data visualization is to calculate the visual layout of each data node during in-between changes. A straightforward way is to recalculate the visual layout of hierarchical data in each frame, which is simple but requires more time consumption. Therefore, previous work [6, 7] introduces several techniques for visualizing dynamic hierarchical data, such as initialization with previous layouts, GPU acceleration, etc. However, topological structures of visualization layout of two adjacent temporal data sequences are not guaranteed to preserve, resulting in unclear and confusion, such as position "jump" of the same data node in an adjacent frame. Therefore, swiftly producing the visual layouts for each frame while preserving the topology is the primary objective of dynamic hierarchical data visualization in our work. In light of this, PowerHierarchy offers an updating scheme to visualize dynamic hierarchical data, using Algorithm 1 as the foundation for computing the visualization layouts.

4.3.2 Updating scheme

Regarding the structure and value of hierarchical data, as explained in Sect. 4.3.1, three different circumstances are considered when visualizing dynamic hierarchical data. In this paper, we propose an updating scheme in PowerHierarchy to visualize dynamic hierarchical data, where the topological structures of visual layouts are preserved during in-between changes. Figure 4 illustrates the computation process of dynamic hierarchical data with topology preservation.

Specifically, the initialization strategy using previous layouts is utilized in our method, which is similar to that of [7]. These sites and associated weights are used to initialize current Voronoi treemaps, followed by weight optimization and position optimization to satisfy the area centroid and capacity constraints. However, as hierarchical data changes make the visual layout of the same data element vary between frames, some sites may lie outside their parent's visual region, resulting in the position of the same data element changing abruptly. To this end, we propose a projection strategy

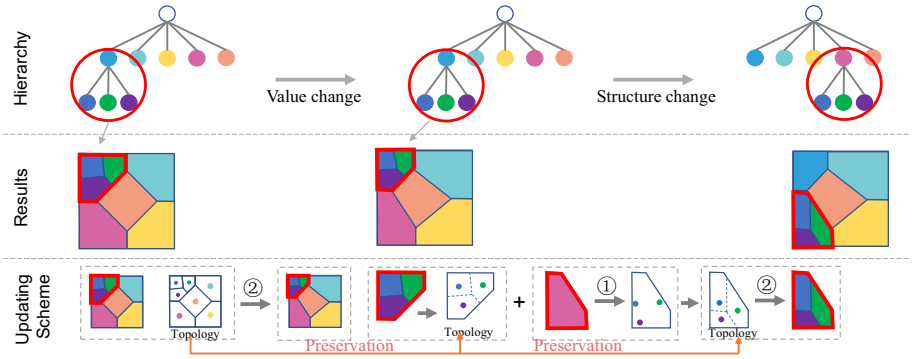


Fig. 4 An illustration of dynamic hierarchical data visualization, where the topological structure of visual layout is taken into account. ① Initialization & external sites projection; ② visual layouts computing by Algorithm 1

to move these external sites to the interior of their visualization region. Subsequently, the subsites of these external sites are moved to the interior of the optimized visualization region with a relative position preserved. An explanation of the updating scheme is given as follows.

(1) *Update, insert and delete* Based on the categories of dynamic hierarchical data in Sect. 4.3.1 modifying sites (or regions) in the visual layouts can also be classified into three types: update, insert and delete. More specifically, updating the relative region in visual layouts corresponds to changing the size of a digital document in Case 1. In Cases 2 and 3, transferring a digital document to another directory refers to deleting the relevant region from the primal layout and inserting it into the new layout.

Firstly, as the values of the data nodes in the hierarchy change, the visualization layout should be updated to reflect the data information in the hierarchy, as shown on the left in Fig. 4. As only the values of the hierarchical data change, the current Voronoi treemap can be initialized using the previous layout (sites and weights) and further optimized using Algorithm 1 to satisfy the area constraint. Secondly, when the data tree structure in the hierarchy changes, the relative area is removed from the previous layout, which is considered to be inserted into the new layout, and the relative position is preserved, as shown on the right in Fig. 4. Similarly, the Voronoi treemap is optimized by Algorithm 1 to meet the area constraint.

A particular scenario exists when the data tree structure is changed. That is, the dynamic hierarchical data involves new data node insertions. These newly inserted data nodes have no relative position or weight information in the previous visualization layouts. Therefore, randomly initialized sites representing these newly inserted data nodes are inserted into the corresponding visualization region with their weights set to the minimum of their neighbors. This initialization strategy, circumventing the possibility of small or empty cells, improves the stability of power diagram construction.

(2) *External site projection and topology preservation* The above initialization strategy utilizes the previous visual results as inputs to initialize the current visual layouts. The

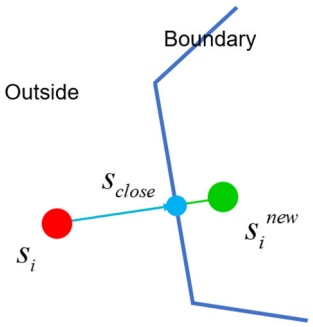


Fig. 5 Exemplification of the external site projection, where the external site s_i is projected onto the visual boundary and then moved into the visualization region

Breadth-First traversing strategy in Algorithm 1 generates the visualization layout of each data node. However, due to changes in hierarchical data in adjacent frames, the visualization layout of the same data element may differ, resulting in some sites (called "external sites") outside the current visual region.

For these external sites, additional processing is vital to preserving the topology of visual layouts during in-between changes. To be more specific, two-stage processing is carried out sequentially: (1) projecting them onto the visual boundary, and then (2) moving them inside the visual region. As shown in Fig. 5, let s_i represent an external site, and blue lines denote the visual boundary. We first project the external site s_i onto the visual boundary, and the projected point is marked as s_{close}. That is, s_{close} is the nearest point to s_i from inside its visual region (despite being located on the boundary). Then, the internal site s_i^{new}, as the site of s_i inside the visual region, could be calculated as: $s_i^{\text{new}} = s_{\text{close}} + \lambda(s_{\text{close}} - s_i)$, where the parameter λ is the offset of the projected point. To circumvent "jump" in the position of the same data element in the visualization layout between adjacent frames, the parameter λ is set to 0.01 so that the internal site s_i^{new} is as close as possible to the external site s_i. Following the two-stage processing, visualization layouts are generated by applying the optimization approach in Algorithm 1 to satisfy the area and centroid constraints.

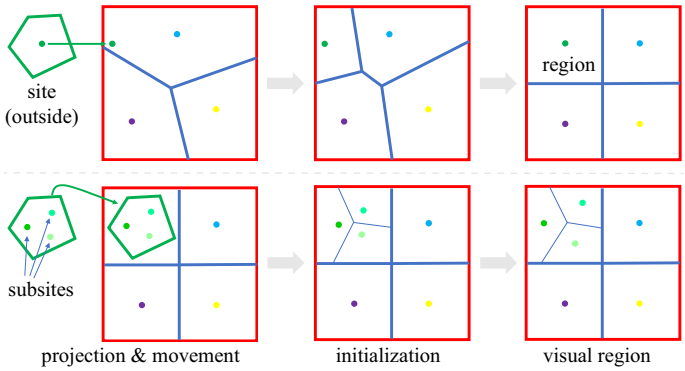


Fig. 6 Exemplification of the topology preservation of layouts with an external site and its subsites. Top row: the projection of an external site to its visualization region (red squared region); and bottom row: the movement of these subsites with their relative position preserved (top left region). From left to right: projection and movement exemplification; initialization with the previous weights and the positions of the moved sites; optimized visualization layouts

As shown in the top row of Fig. 6, sites outside the visualization region are first moved to the inside of their relevant region (red squared region). Thus, the current Voronoi treemap is initialized by combining the positions and weights of other sites. Based on this, combining the weight optimization and position optimization, the visualization layout is optimized to satisfy the area and centroid constraints, as shown in the last figure in the top row of Fig. 6 (assuming the same area constraint for all sites here).

In contrast to the projection strategy described above, the movement of the subsites of these external sites relies on the visual layout calculation. Owing to the Breadth-First traversing strategy in Sect. 4.2, the visual layout of their parent site is determined when computing the visual layouts of these subsites. These subsites are scaled and moved directly into the calculated visual region while retaining their relative positions, as shown in the first figure in the bottom row of Fig. 6. Then, the positions and weights are used as initialization (middle figure in the bottom row of Fig. 6), combined with weight optimization and position optimization to produce the visual layout, as shown in the last figure in the bottom row of Fig. 6.

Therefore, the external site projection and topology preservation could be summarized as (1) these external sites are projected onto the visual boundary and then moved into the relevant visual region (top in Fig. 6), and (2) their subsites are scaled and moved into the optimized visual region with their relative position in previous layout preserved (bottom in Fig. 6). Consequently, our method could effectively preserve the visualization layouts during each frame for visualizing dynamic hierarchical data.

5 Performance evaluation

We implement and experiment with PowerHierarchy on a computer with a 64-bit version of Win10 system, a 3.6 GHz Intel (R) Core (TM) i7–9700K CPU, and 16 GB memory. The coding language is C++, and the platform is Microsoft Visual C++ 2012. It should be mentioned that the compared methods are reimplemented in C++, and the power diagrams are constructed by Computational Geometry Algorithms Library (CGAL−4.7).

5.1 Datasets

We perform the test on a variety of datasets, including the Dataset of China Plant (DCP), the Flare class hierarchy dataset (Flare) [28], the KEGG orthology dataset (KEGG) [29], the global GDP dataset (GDP) [28], and the Consumer Price Index Dataset (CPI), to confirm the effectiveness of PowerHierarchy.

(1) *DCP* The DCP is a scientific dataset of plant species in China established by the Institute of Botany, Chinese Academy of Sciences. With over 30 000 species in roughly 300 families, the dataset primarily provides basic information concerning the systematic categorization of plant species. In the experiment, we select a portion of the species data consisting of five layers with 157 nodes, where the value of each data node indicates the number of species under the corresponding family category.
(2) *Flare* The Flare project [28] is an ActionScript framework for developing visualizations that run in the Adobe Flash Player, which has a hierarchical software structure consisting of four layers with 220 nodes. In this dataset, the value of each data node represents the size of the corresponding source file in the Flare project.

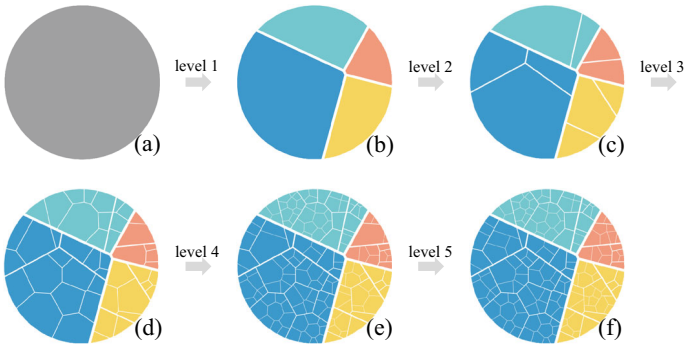


Fig. 7 Visualization illustration of five revisions of a hierarchical data from DCP, where the area size reflects the value of its relative data node

(3) *KEGG* The KEGG [29] is a dataset of molecular activities expressed as function orthologs, composed of four layers and 17600 nodes. In our tests, the top four KEGG Pathway map categories are chosen, where the value of each data node shows the number of subordinate categories.
(4) *GDP* The GDP dataset [28] separates the global GDP into seven categories: Asia, North America, South America, Europe, Australia, Africa, and the Rest of the World. Note that the GDP dataset consists of two layers with 42 nodes since these countries or regions with a GDP of less than 300 billion are included in the last category. Additionally, our tests use the global GDP from 2010 to 2019 as dynamic hierarchical data.
(5) *CPI* The CPI dataset is a dynamic hierarchical dataset that counts all Chinese provinces in the first quarter of 2019 to 2022. We collect detailed data from the National Bureau of Statistics of China. All areas in China are classified into six modules based on the geographical location of the province: North China, Northeast China, East China, South Central China, Southwest China, and Northwest China. The value of each data node indicates the per capita consumer expenditure of residents in the corresponding province.

5.2 Evaluation for static hierarchical data

5.2.1 Visualization results

To demonstrate the feasibility of our approach, several tests are performed on these datasets provided in Sect. 5.1. Our method's fundamental principle is recursively computing each data node's visual layout. Thus, we first compute the visualization results of hierarchical data in the DCP dataset on a circular region to illustrate the computational flow of our method. The generated visualization layouts of five layers are presented in Fig. 7, and the original visual region is shown in Fig. 7a. Furthermore, to verify the robustness and effectiveness of our method, we conduct more experiments on DCP, Flare, and KEGG datasets to visualize more complex hierarchical data. The primal visual regions are set to the shapes of a flower, an elliptical, and a squared region. Figure 8 presents the corresponding visualization results. The results in Fig. 7 demonstrate the feasibility of our method, and those in Fig. 8 further indicate the effectiveness and robustness of visualizing more complex hierarchies, even with the flower visual boundary.

5.2.2 Comparison

To evaluate the superiority of our method, we compare the proposed method with three hierarchical data visualization methods, including the Slice & Dice treemap [12], the Squarified treemap [13], and the Voronoi treemap [3]. Due to the restrictive nature of the primal visualization region in the Slice & Dice treemap [12] and the Squarified treemap [13], we set a square visualization region as the input visual region. Figure 9 presents the visualization results of various methods in three datasets. Figure 10 demonstrates the associated violin plots of the aspect ratio of produced visualization layouts. The aspect ratio of a power cell is calculated as $\text{ratio}(P(s_i)) = \min\{w/h, h/w\}$, where w and h are the width and height of the power cell $P(s_i)$. Notably, a power cell that has an aspect ratio close to 1 implies greater visibility.

According to the results in Figs. 9 and 10, we can observe that the Slice & Dice treemap [12] typically generates cells with a high aspect ratio, leading to the results being unclear and difficult to understand. To circumvent these high aspect ratio cells, the Squarified treemap [13] utilizes the greedy algorithm to produce the visualization layouts of hierarchical data. Nevertheless, despite the attractive aspect ratio of the visualization results, it is a challenge to determine the relationships between the data nodes in the hierarchy. In view of visualizing hierarchical data using Voronoi treemaps, the results produced by PowerHierarchy perform better than those in [3]. In particular, PowerHierarchy yields more regular cells with a better aspect ratio, as shown in the red wireframes of Fig. 9.

5.2.3 Computational efficiency

Computational efficiency is a vital indicator of a visualization approach. A good visualization approach should be

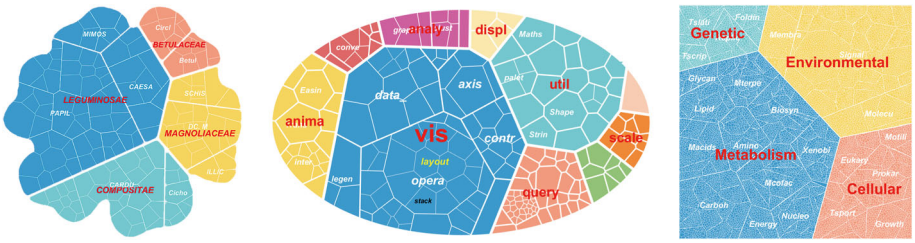


Fig. 8 Visualization results of hierarchical data by PowerHierarchy in three datasets, where each cell area corresponds to the value of data element in the hierarchies. From left to right: visualization result in the Dataset of China Plant (DCP, five layers with 157 nodes), the Flare class dataset (Flare, four layers with 220 nodes), and the KEGG dataset (KEGG, four layers with 17600 nodes). Taking the Flare class dataset as an example, the hierarchical relationship of data elements is shown as: *vis* (first layer), *opera* (second layer), *layout* (third layer), and *stack* (fourth layer)

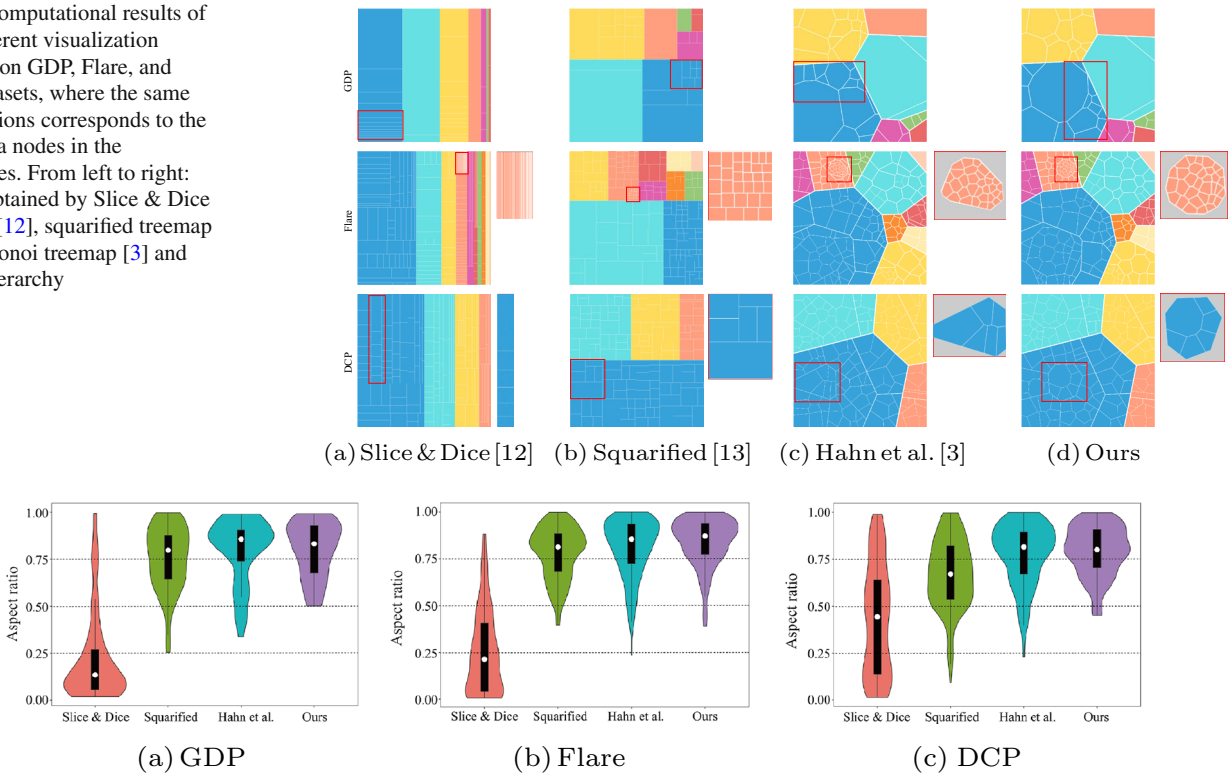


Fig. 9 Computational results of four different visualization methods on GDP, Flare, and DCP datasets, where the same color regions corresponds to the same data nodes in the hierarchies. From left to right: results obtained by Slice & Dice treemap [12], squarified treemap [13], Voronoi treemap [3] and PowerHierarchy

Fig. 10 The violin plots of the aspect ratio of the layouts obtained using various methods, corresponding to these results in Fig. 9

able to compute the visualization layouts of hierarchical data relatively quickly, and the area error of the generated visualization results should be as small as possible. Treemap recursively computes the layouts of data elements for each layer in the hierarchy. Thus, the timing of the layout computation for single-layer data elements is closely related to the computational efficiency of the visualization approach. To compare the performance of PowerHierarchy with other methods, we design an experiment with the various sites and count the corresponding computational time of the visualization layouts. Besides, we also calculate the area error of the visualization layouts of different methods with the same iteration, where the area error ϵ could be calculated by $\max\{|A_i^{\text{current}} - A_i^{\text{target}}|\}$. Table 2 presents the computational time and the area error of different methods. Notably, to keep things consistent, the area accuracy is the same when comparing computational efficiency of different methods (Table 2).

The results in Table 2 show that the Slice & Dice treemap [12] and Squarified treemap [13] achieve better computational performance. They directly compute the rectangular visualization layout corresponding to each data element with-

Table 2 Computational time and area accuracy comparison of PowerHierarchy and other methods

Nodes (n)	Methods	Timing (s)	Area error of same iterations		
			#Iter = 20	#Iter = 30	#Iter = 40
10	Slice & Dice [12]	0.083	0.000	0.000	0.000
	Squarified [13]	0.132	0.000	0.000	0.000
	Hahn et al. [3]	0.957	8.941E-2	5.472E-3	6.459E-4
	Ours	0.393	6.861E-5	3.186E-7	6.521E-9
50	Slice & Dice [12]	0.382	0.000	0.000	0.000
	Squarified [13]	0.517	0.000	0.000	0.000
	Hahn et al. [3]	6.485	8.149E-2	6.617E-3	2.752E-4
	Ours	3.316	5.793E-4	7.836E-6	8.711E-9

out iterative optimization. Thus, these two methods produce visualization layouts with an area error of 0. However, it is well-known that the primal visual region of these two methods can only be rectangular, and the generated visual layouts have an unsatisfied aspect ratio of the Slice & Dice. In view of the visualization approach via Voronoi treemaps, the method provided by Hahn et al. [3] optimizes the weights proportionally while optimizing the sites using Lloyd's method to generate visualization results. This method, however, takes a lot longer. Unlike [3], using numerical optimization, PowerHierarchy combines the L-BFGS method for sites with Newton's for weights to yield the visualization layouts. Additionally, our method computes a CVT as input to prevent some corner situations (e.g., empty cells, etc.), improving the computational efficiency while enhancing the stability of power diagram construction. The results in Fig. 2 provide evidence that PowerHierarchy could achieve more accurate visualization layouts with less time consumption. Thus, it is possible to apply PowerHierarchy for visualizing time-dependent hierarchical data.

5.3 Evaluation for dynamic hierarchical data

5.3.1 Visualization results

To demonstrate our method's effectiveness and topology preservation, we perform our method in the 2010–2019 global GDP dataset. The 2010–2019 global GDP dataset includes statistics on the GDP of each nation and area from 2010 to 2019. All countries and regions are classified into seven groups based on their geographic location and GDP values: Asia, North America, Europe, South America, Australia, Africa, and the Rest of the World. The GDP value of every nation and region is tallied online, and those with too low GDP (less than $300 billion) are included in the Rest of the Word section.

Figure 11 provides the visualization results of the yearly global GDP, using the original GDP data as the sole input in our system. Additionally, Fig. 12 presents our visualization system, and more dynamic results are provided in the supplementary video, e.g., the addition, deletion, and modification of data nodes, etc. These experimental results demonstrate the effectiveness and feasibility of our method. Thanks to the animated visualization results, these economists could more easily track changes in the global GDP. Taking the GDP of Asia as an example, the region with the label "CN" stands for the GDP of China, while the region with the label "JP" denotes the GDP of Japan. The results in Fig. 11 demonstrate that China's GDP is steadily growing in relation to Japan's GDP, indicating that China's economy is of significance in Asia.

5.3.2 Comparison & computational efficiency

In view of visualizing dynamic hierarchical data, we compare PowerHierarchy with the one put forward by Sud et al. [7] on a three-level DCP dataset. Three situations are considered: (1) raising the values of data nodes; (2) changing the structure of the data tree; and (3) altering both (1) and (2). Specifically, the first situation involves raising the value of a node named "Ostrya Scop.", and the second situation corresponds to switching the structure of a subtree named "CAES. TAUB." from a node "LEGUMINOSAE" to "BETAULACEAE". Notably, the method presented by Sud et al. [7] employs the additively weighted Voronoi diagram to compute the visualization layouts. To maintain experimental consistency, the method in [7] is reimplemented in C++ using a CPU, and power diagrams are utilized as the foundation for visualization layout computation. The results of visualizing dynamic hierarchical data are shown in Fig. 13, where the green star indicates the nested regions of the node with value change, and the regions with red borderlines designate the node with structural change in the hierarchy. Table 3 reports the computational time of these results.

Furthermore, to validate the superiority of our method, we conduct an additional comparative experiment with the method proposed by Sud et al. [7] on another dynamically hierarchical dataset, namely the CPI dataset. The CPI dataset

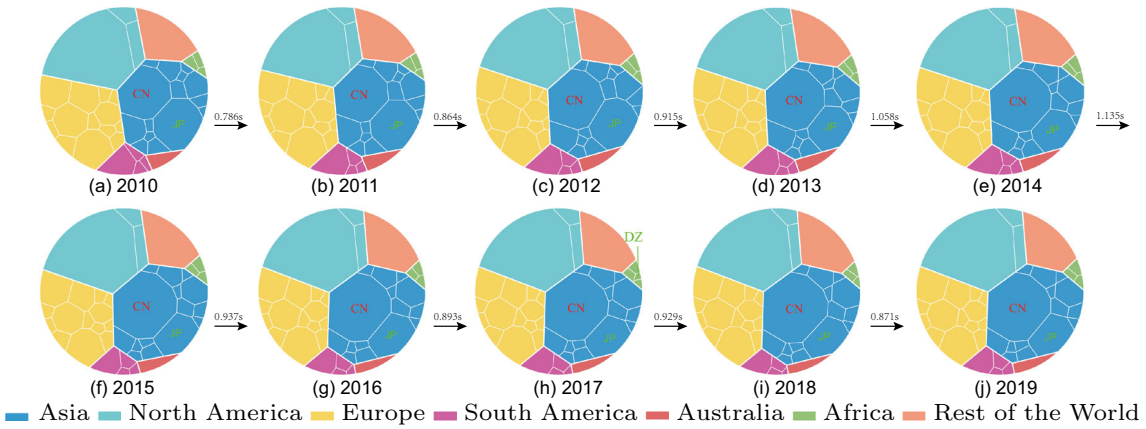


Fig. 11 Visualization layouts and the computational timings (in seconds) of the global GDP from 2010 to 2019, where the color regions represent different continental plates in the world (colored by the TreeColor [30]). The subregion labeled "CN" corresponds to the GDP of China, "JP" represents the GDP of Japan, and "DZ" denotes the GDP of Algeria

Fig. 12 A screen capture of our system showing the layouts of our proposed visualization approach in use to view the global GDP from 2010 to 2019

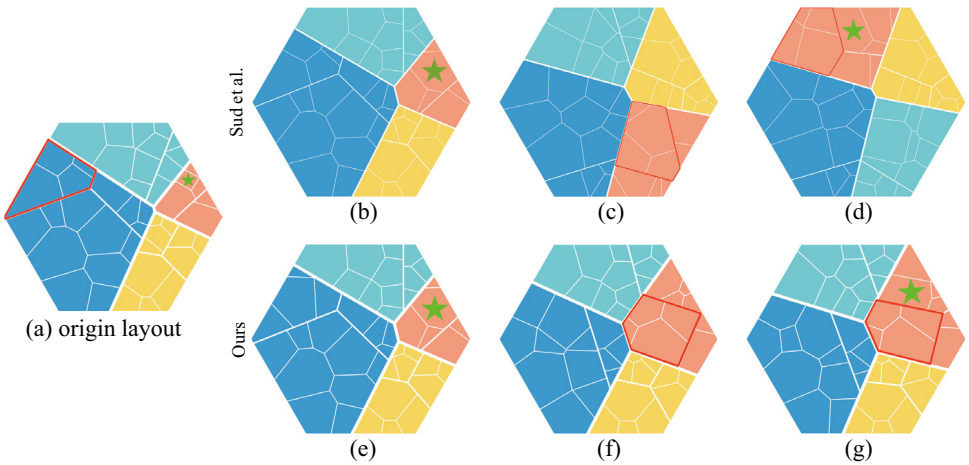


Fig. 13 Experimental results of visualizing dynamic hierarchical data by PowerHierarchy and the method in [7]. Top row: results generated by the method in [7], and bottom row: results generated by PowerHierarchy. The previous visualization result is shown in **a**; **b** and **e** are results of situation (1); **c** and **f** represent results of situation (2); and **d** and **f** denote results of situation (3)

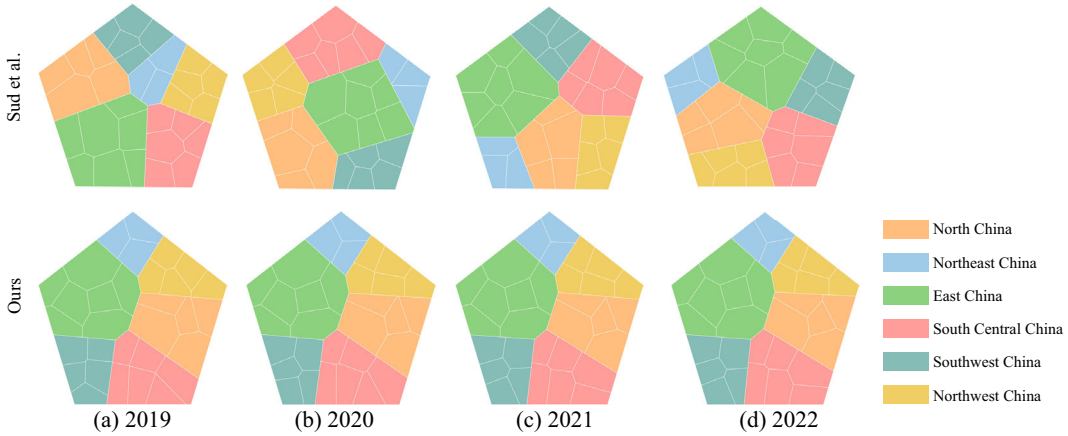


Fig. 14 Visualization layouts of different methods on the 2019–2022 CPI dataset from China, where the color regions show the detailed information of the relevant province

Table 3 Computational time of the visualization layouts generated by the method in [7] and PowerHierarchy (corresponding to these results in Fig. 13)

Methods	Layouts	#Iter	Time (s)
Sud et al. [7]	Fig. 13b	59	1.079
Ours	Fig. 13e	53	0.958
Sud et al. [7]	Fig. 13c	91	1.519
Ours	Fig. 13f	64	1.154
Sud et al. [7]	Fig. 13d	98	1.637
Ours	Fig. 13g	71	1.247

consists of an essential macroeconomic indicator reflecting the changes in the price level of consumer goods and services related to people's life. We collect this detailed information from the National Bureau of Statistics of China, specifically for each province in the first quarter of each year from 2019 to 2022. These provinces are divided into six parts based on their regional location, as shown in Sect. 5.1. Figure 14 presents the visualization results of different methods on the CPI dataset.

From the results in Fig. 13 and Fig. 14, we can observe that our proposed method can effectively compute the visualization layouts of dynamic hierarchical data. The previous visualization layouts are fed as inputs to initialize the current visualization layouts in the method proposed by Sud et al. [7]. For simple dynamic hierarchical data (e.g., Fig. 13b), the topology structures of visualization layouts can be preserved, but it cannot be guaranteed for those complex dynamic hierarchical data (e.g., Fig. 13c, d), which may cause the visualization results unclear and confusing. However, the updating scheme projects those sites outside the visualization region and their subsites into the corresponding region (described in Sect. 4.3.2). Consequently, the topology of visualization layouts of two adjacent temporal data sequences generated by PowerHierarchy can be guaranteed to preserve, as shown in Fig. 13e–g. Furthermore, the computational time reported in Table 3 indicates that the computational efficiency of our proposed updating scheme is similar to the method in [7] when visualizing simple dynamic hierarchical data (e.g., only value changes), but it has better performance in visualizing complex dynamic hierarchical data (e.g., data structure changes).

6 Conclusion

In this paper, we propose an efficient and topology-preserved visualization approach, called PowerHierarchy, for visualizing hierarchical data. An improved version of the power diagram computing algorithm from [8] is presented to achieve better efficiency and accuracy, which computes a CVT after random initialization. Then it utilizes Newton's and L-BFGS methods to calculate the visualization layouts. On this basis, an updating scheme is introduced, where the previous results are fed as inputs to initialize current layouts. Those external sites and their subsites are iteratively projected into the visual boundary and then moved into the visual region. Experimental results on several datasets demonstrate the effectiveness and accuracy of PowerHierarchy for visualizing hierarchical data. Besides, the topological structures of visualization layouts generated by the updating scheme could be preserved during in-between frames.

Limitations and future work However, there is a common issue in the hierarchical data visualization technique via Voronoi treemaps. Some irregular cells may appear in the visualization layouts when the difference between two

nodes with the same parent is too large, as shown in Fig. 12 (the cell corresponds to the GDP of Australia). In addition, our proposed visualization approach could be extended to simple non-convex visualization region shapes (such as the flower shape in Fig. 8a). Still, it is not suitable for these shapes with more complex boundaries. In the future, we would like to extend our work with other treemap techniques to improve visualizing complex hierarchical data with complex boundaries.

Supplementary Information The online version contains supplementary material available at https://doi.org/10.1007/s00371-023-02864-4.

Acknowledgements This study was supported in part by a grant from the National Natural Science Foundation of China (61972128).

Data Availability Statement The datasets generated during and/or analysed during the current study are available from the corresponding author on reasonable request.

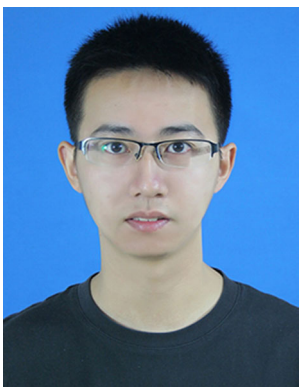
Declarations

Conflict of interest The authors have no conflicts of interest/competing interests to declare that are relevant to the content of this article.

References

1. Vernier, E.F., Telea, A.C., Comba, J.: Quantitative comparison of dynamic treemaps for software evolution visualization. In: 2018 IEEE Working Conference on Software Visualization (VISSOFT), pp. 96–106 (2018). https://doi.org/10.1109/vissoft.2018.00018. IEEE
2. Vernier, E., Sondag, M., Comba, J., Speckmann, B., Telea, A., Verbeek, K.: Quantitative comparison of time-dependent treemaps. Comput. Graph. Forum **39**(3), 393–404 (2020). https://doi.org/10.1111/cgf.13989
3. Hahn, S., Trümper, J., Moritz, D., Döllner, J.: Visualization of varying hierarchies by stable layout of voronoi treemaps. In: 2014 International Conference on Information Visualization Theory and Applications (IVAPP), pp. 50–58 (2014). https://doi.org/10.5220/0004686200500058. IEEE
4. Balzer, M., Deussen, O.: Voronoi treemaps. In: IEEE Symposium on Information Visualization, 2005. INFOVIS 2005., pp. 49–56 (2005). https://doi.org/10.1109/INFVIS.2005.1532128. IEEE
5. Nocaj, A., Brandes, U.: Computing Voronoi treemaps: Faster, simpler, and resolution-independent. Comput. Graph. Forum **31**(3), 855–864 (2012)
6. Gotz, D.: Dynamic voronoi treemaps: a visualization technique for time-varying hierarchical data. Phys. Rev. A **30**(2), 150–156 (2011). https://doi.org/10.1109/TADVP.2007.896008
7. Sud, A., Fisher, D., Lee, H.-P.: Fast dynamic voronoi treemaps. In: 2010 International Symposium on Voronoi Diagrams in Science and Engineering, pp. 85–94 (2010). https://doi.org/10.1109/isvd.2010.16. IEEE
8. Xin, S.-Q., Lévy, B., Chen, Z., Chu, L., Yu, Y., Tu, C., Wang, W.: Centroidal power diagrams with capacity constraints: computation, applications, and extension. ACM Trans. Graph. (TOG) **35**(6), 1–12 (2016). https://doi.org/10.1145/2980179.2982428
9. Scheibel, W., Trapp, M., Limberger, D., Döllner, J.: A taxonomy of treemap visualization techniques. In: VISIGRAPP (3: IVAPP), pp. 273–280 (2020). https://doi.org/10.5220/0009153902730280
10. Khalid, Z.M., Zeebaree, S.R.: Big data analysis for data visualization: a review. Int. J. Sci. Bus. **5**(2), 64–75 (2021). https://doi.org/10.5281/zenodo.4462042
11. Scheibel, W., Limberger, D., Döllner, J.: Survey of treemap layout algorithms. In: Proceedings of the 13th International Symposium on Visual Information Communication and Interaction, pp. 1–9 (2020). https://doi.org/10.1145/3430036.3430041
12. Johnson, B., Shneiderman, B.: Tree-maps: a space-filling approach to the visualization of hierarchical information structures. Read. Inf. Visualiz. Using Vision Think (1999). https://doi.org/10.1109/visual.1991.175815
13. Knauthe, V., Ballweg, K., Wunderlich, M., Landesberger, T., Guthe, S.: Influence of container resolutions on the layout stability of squarified and slice-and-dice treemaps. In: Eurographics/IEEE VGTC Conference on Visualization, pp. 97–101 (2020). https://doi.org/10.2312/evs20201055
14. Ahmed, A.G.: Voronoi tree maps with circular boundaries. In: Proceedings of the Conference on Computer Graphics & Visual Computing, pp. 115–116 (2018). https://doi.org/10.2312/cgvc.20181214
15. Yang, B., Cao, W.: The ordered treemap of weight divided layout algorithm. J. Comput. **30**(5), 31–45 (2019). https://doi.org/10.3966/199115992019103005003
16. Chen, Y., Du, X., Yuan, X.: Ordered small multiple treemaps for visualizing time-varying hierarchical pesticide residue data. Vis. Comput. **33**, 1073–1084 (2017). https://doi.org/10.1007/s00371-017-1373-x
17. Görtler, J., Schulz, C., Weiskopf, D., Deussen, O.: Bubble treemaps for uncertainty visualization. IEEE Trans. Visual Comput. Graph. **24**(1), 719–728 (2017). https://doi.org/10.1109/tvcg.2017.2743959
18. Wang, Y.-C., Xing, Y., Lin, F., Seah, H.-S., Zhang, J.: Ost: a heuristic-based orthogonal partitioning algorithm for dynamic hierarchical data visualization. J. Visual. (2022). https://doi.org/10.1007/s12650-022-00830-1
19. Sondag, M., Speckmann, B., Verbeek, K.: Stable treemaps via local moves. IEEE Trans. Visual Comput. Graph. **24**(1), 729–738 (2017). https://doi.org/10.1109/tvcg.2017.2745140
20. Balzer, M.: Capacity-constrained voronoi diagrams in continuous spaces. In: 2009 Sixth International Symposium on Voronoi Diagrams, pp. 79–88 (2009). https://doi.org/10.1109/ISVD.2009.28. IEEE
21. De Goes, F., Breeden, K., Ostromoukhov, V., Desbrun, M.: Blue noise through optimal transport. ACM Trans. Graph. (TOG) **31**(6), 1–11 (2012). https://doi.org/10.1145/2366145.2366190
22. Zheng, L., Yao, Y., Wu, W., Xu, B., Zhang, G.: A novel computation method of hybrid capacity constrained centroidal power diagram. Comput. Graph. **97**, 108–116 (2021). https://doi.org/10.1016/j.cag.2021.04.007
23. Zheng, L., Gui, Z., Cai, R., Fei, Y., Zhang, G., Xu, B.: GPU-based efficient computation of power diagram. Comput. Graph. **80**, 29–36 (2019). https://doi.org/10.1016/j.cag.2019.03.011
24. Du, Q., Faber, V., Gunzburger, M.: Centroidal Voronoi tessellations: applications and algorithms. SIAM Rev. **41**(4), 637–676 (1999). https://doi.org/10.1137/S0036144599352836
25. Aurenhammer, F.: Power diagrams: properties, algorithms and applications. SIAM J. Comput. **16**(1), 78–96 (1987). https://doi.org/10.1137/0216006
26. Aurenhammer, F., Hoffmann, F., Aronov, B.: Minkowski-type theorems and least-squares clustering. Algorithmica **20**(1), 61–76 (1998). https://doi.org/10.1007/pl00009187
27. Liu, Y., Wang, W., Lévy, B., Sun, F., Yan, D.-M., Lu, L., Yang, C.: On centroidal Voronoi tessellation-energy smoothness and

fast computation. ACM Trans. Graph. (ToG) **28**(4), 1–17 (2009). https://doi.org/10.1145/1559755.1559758
28. Wang, Y.-C., Liu, J., Lin, F., Seah, H.-S.: Generating orthogonal Voronoi treemap for visualization of hierarchical data. Comput. Graph. Int. Conf. (2020). https://doi.org/10.1007/978-3-030-61864-3_33
29. Bernhardt, J., Funke, S., Hecker, M., Siebourg, J.: Visualizing gene expression data via voronoi treemaps. In: 2009 Sixth International Symposium on Voronoi Diagrams, pp. 233–241 (2009). https://doi.org/10.1109/ISVD.2009.33. IEEE Computer Society
30. Li, B., Zhang, X.: Tree-coloring problems of bounded treewidth graphs. J. Comb. Optim. **39**(1), 156–169 (2020). https://doi.org/10.1007/s10878-019-00461-7



Yuyou Yao received the bachelor's degree in software engineering from Hefei University of Technology, Hefei, China, in 2018. He is currently pursuing the Ph.D. degree in software engineering with School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China. His research interests include computer graphics, visualization and geometric processing.

Tao Li received the bachelor's degree in computer science from Huaibei Normal University, Huaibei, China, in 2020. He is currently pursuing the master's degree in computer science with School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China. His research interests include computer graphics and image processing.

Wenming Wu received the bachelor's degree in statistical mathematics from Anhui University, Hefei, China, in 2015, and the Ph.D. degree in computational mathematics from University of Science and Technology of China, Hefei, China, in 2020. He is currently a lecturer in the School of Computer Science and Information Engineering, Hefei University of Technology, Hefei, China. His research interests include computer graphics and computer vision.

Gaofeng Zhang received the bachelor's and master's degrees in computer science from Hefei University in 2005 and 2008, respectively, and the Ph.D. degree from Swinburne University of Technology, Australia, in 2013. He is currently an assistant professor with the School of Software, Hefei University of Technology. His research interests include computer graphics, service computing and software security.

Liping Zheng received the bachelor's and Ph.D. degrees in computer science from Hefei University of Technology, Hefei, China, in 2004 and 2008, respectively. He is currently a professor with the School of Computer Science and Information Engineering, Hefei University of Technology. His research interests include computer graphics, visualization and computer simulation.

Terms and Conditions

Springer Nature journal content, brought to you courtesy of Springer Nature Customer Service Center GmbH (“Springer Nature”).
Springer Nature supports a reasonable amount of sharing of research papers by authors, subscribers and authorised users (“Users”), for small-scale personal, non-commercial use provided that all copyright, trade and service marks and other proprietary notices are maintained. By accessing, sharing, receiving or otherwise using the Springer Nature journal content you agree to these terms of use (“Terms”). For these purposes, Springer Nature considers academic use (by researchers and students) to be non-commercial.
These Terms are supplementary and will apply in addition to any applicable website terms and conditions, a relevant site licence or a personal subscription. These Terms will prevail over any conflict or ambiguity with regards to the relevant terms, a site licence or a personal subscription (to the extent of the conflict or ambiguity only). For Creative Commons-licensed articles, the terms of the Creative Commons license used will apply.
We collect and use personal data to provide access to the Springer Nature journal content. We may also use these personal data internally within ResearchGate and Springer Nature and as agreed share it, in an anonymised way, for purposes of tracking, analysis and reporting. We will not otherwise disclose your personal data outside the ResearchGate or the Springer Nature group of companies unless we have your permission as detailed in the Privacy Policy.
While Users may use the Springer Nature journal content for small scale, personal non-commercial use, it is important to note that Users may not:

1. use such content for the purpose of providing other users with access on a regular or large scale basis or as a means to circumvent access control;
2. use such content where to do so would be considered a criminal or statutory offence in any jurisdiction, or gives rise to civil liability, or is otherwise unlawful;
3. falsely or misleadingly imply or suggest endorsement, approval , sponsorship, or association unless explicitly agreed to by Springer Nature in writing;
4. use bots or other automated methods to access the content or redirect messages
5. override any security feature or exclusionary protocol; or
6. share the content in order to create substitute for Springer Nature products or services or a systematic database of Springer Nature journal content.

In line with the restriction against commercial use, Springer Nature does not permit the creation of a product or service that creates revenue, royalties, rent or income from our content or its inclusion as part of a paid for service or for other commercial gain. Springer Nature journal content cannot be used for inter-library loans and librarians may not upload Springer Nature journal content on a large scale into their, or any other, institutional repository.
These terms of use are reviewed regularly and may be amended at any time. Springer Nature is not obligated to publish any information or content on this website and may remove it or features or functionality at our sole discretion, at any time with or without notice. Springer Nature may revoke this licence to you at any time and remove access to any copies of the Springer Nature journal content which have been saved.
To the fullest extent permitted by law, Springer Nature makes no warranties, representations or guarantees to Users, either express or implied with respect to the Springer nature journal content and all parties disclaim and waive any implied warranties or warranties imposed by law, including merchantability or fitness for any particular purpose.
Please note that these rights do not automatically extend to content, data or other material published by Springer Nature that may be licensed from third parties.
If you would like to use or distribute our Springer Nature journal content to a wider audience or on a regular basis or in any other manner not expressly permitted by these Terms, please contact Springer Nature at

onlineservice@springernature.com