Survey of Treemap Layout Algorithms

Willy Scheibel
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany

Daniel Limberger
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany

Jürgen Döllner
Hasso Plattner Institute,
Faculty of Digital Engineering,
University of Potsdam, Germany

ABSTRACT

This paper provides an overview of published treemap layout algorithms from 1991 to 2019 that were used for information visualization and computational geometry. First, a terminology is outlined for the precise communication of tree-structured data and layouting processes. Second, an overview and classification of layout algorithms is presented and application areas are discussed. Third, the use-case-specific adaption process is outlined and discussed. This overview targets practitioners and researchers by providing a starting point for own research, visualization design, and applications.

CCS CONCEPTS

• **Human-centered computing** → **Treemaps**; **Visualization design and evaluation methods**.

KEYWORDS

Treemap Layout Algorithms, Tree Spatialization, Graph Drawing

ACM Reference Format:
Willy Scheibel, Daniel Limberger, and Jürgen Döllner. 2020. Survey of Treemap Layout Algorithms. In *The 13th International Symposium on Visual Information Communication and Interaction (VINCI 2020), December 8–10, 2020, Eindhoven, Netherlands.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3430036.3430041

1 INTRODUCTION

For much of today's tree-structured data—either inherent or otherwise supplemented—the family of treemap visualization techniques provides a versatile and well established way for visualization [60]. As such, the domains of software visualization [54] (Figure 1), demographics [45], biology and medicine [19], and business data [69] deal with complex, tree-structured datasets, and have a growing need for visualization. Since the introduction of treemaps [47], related research is present at conferences and in journals every year, publishing advances in data processing, layouting techniques, visual variable mapping and encoding, and applications.

Eventually, this led to a large and even growing number of treemap layout algorithms. These number of algorithms and visualization techniques were consolidated in design spaces and surveys before. As such, Schulz et al. proposed both a design space for 3D treemaps [76] and an extension to this design space to cover implicit tree visualizations more generally [75]. A visual survey on tree visualization techniques in general is published at treevis.net [73].

VINCI 2020, December 8–10, 2020, Eindhoven, Netherlands

More specifically with focus on treemaps, Shneiderman provides a tool-centric view on treemap visualization techniques [78]. All of these publications focus on the visualization technique and the visual encoding. Focusing more on the layout algorithms, Baudel and Broeksema proposed a design space of sequential, space-filling layouts [8]. Schulz et al. later provided a design space for general rooted tree drawings [74].

On one hand, this availability leaves visualization designers to question which algorithm is suited for their intend. On the other hand, the selection of an appropriate algorithm may be driven by technical decisions. For example, the size of the dataset and the data that should be visually encoded needs to be considered as well. While the strict attribution of a visualization technique being either a treemap or a tree visualization technique [70], this survey focuses on layout algorithms that employ "the property of containment" [46], i.e., containment treemaps $\mathcal{T}_C$. As such, we consider algorithms that

- map a rooted tree into a kD layout space ($k \in \mathbb{N}^*$),
- account for multiple attributes and additional relations, and
- distribute child nodes *within* their parents' extent.

This paper presents a review of said algorithms by (1) recapitulating and consolidating basic terminology used, (2) presenting a classification of their characteristics, and (3) providing guidance for the choice of algorithm for applications of treemaps. Starting with the initial publication of treemaps in 1991, we considered 81 treemap layout algorithms published in conference proceedings and journal articles of major publishers—IEEE, ACM, Springer, EG, Elsevier, Palgrave, and SciTePress—as well as selected doctoral theses in the years from 1991 to 2019.

The remainder of this paper is structured as follows. Section 2 recapitulates terminology used for tree-structured data and treemap layouting techniques. Section 3 reviews available layout algorithms

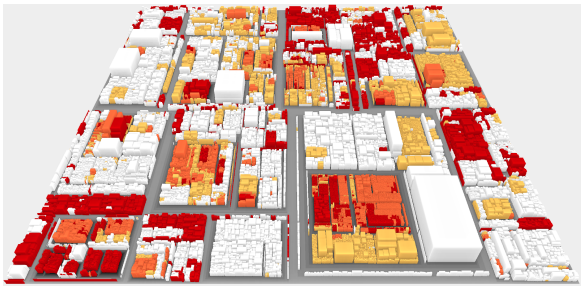

Figure 1: Example 2.5D treemap visualization of the source code files of the Firefox software project with 109 000 nodes using the Hilbert treemap layout algorithm and recursive nested depiction of child nodes.

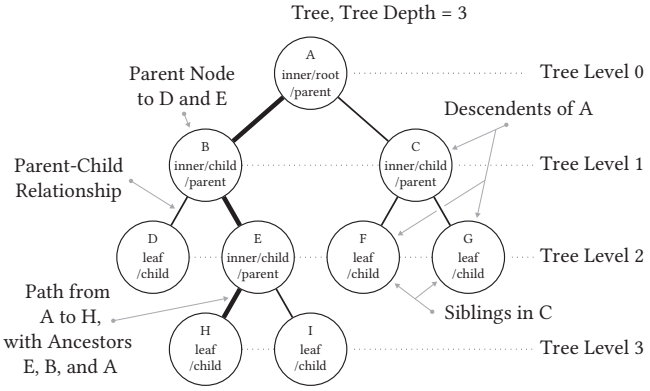

Figure 2: A tree-structure depiction with annotated terms.

for treemaps. Section 4 discusses adaptation and layout postprocessing. Section 5 concludes this paper.

2 TREEMAP TERMS

Depictions of tree-structured data are subject to research for at least a century [32]. The language used to discuss and communicate their making, intent, and effectiveness has changed and most likely will continue to change. Although the purpose of this section is to facilitate unambiguous and precise communication, we are aware of the general ambiguity of language and different approaches to modeling and communication. Nevertheless, we reiterate common terms for *tree structure* and *spatialization* using notions of graph theory [25] and visualization theory. As a general case, a tree-structured dataset can be modeled as a graph built up by vertices (data points) and at least one relationship among the vertices represented by edges.

Tree Structure Terms. For a dataset to be suitable for treemapping, the expected structure of the nodes and edges is expected to be a rooted tree. Often, we find the term *hierarchy* to be used synonymously, but we suggest to strictly distinguish. As an additional special case, a dataset with forest structure can be used for treemapping by employing a virtual root node.

Hierarchy. A special type of a graph with the characteristics that there is one type of directed relation (and, thus, one type of directed edges) encoding a parent-child relationship with no circles (synonym: directed, acyclic graph).

Tree. A special type of a hierarchy with the main characteristic that the hierarchy is *arborescent*, i.e., it has a single vertex with no parent relationship, each other vertex has exactly one *parent* vertex.

Forest. A special type of hierarchy that represents a set of trees, i.e., there may be multiple vertices with no parent relationship.

With respect to a tree-structured dataset, there are additional termini to precisely denote characteristics, assumptions, and algorithms for treemap visualizations (Figure 2):

Node. The representation of a data point in the semantics of a tree in information visualization (i.e., a vertex in graph theory).

Relation. Synonym to an edge in graph theory.

Parent-Child Relationship. A directed relation connecting two nodes, in which context one node has the *parent* role and the other has the *child* role.

Parent Node / Inner Node. A node that has child nodes, i.e., it has one or more relationships with other nodes where the node itself is a *parent*.

Root Node A node of a tree that has no parent node. For actual trees there is only a single node per tree; for forests there is one root node for each tree of the forest.

Child Node. A node that has a parent node, i.e., it has a relationship with another node where the node itself is a *child*.

Leaf Node. A node without associated child nodes.

For such a tree, additional properties and concepts can be applied that are used when deriving treemaps:

Path. The shortest list of nodes that connects two nodes using the parent-child relationship. Thereby, the first node is the source node, the last node is the target node and each two adjacent nodes in the list are connected through a parent-child relationship.

Ancestors. The set of nodes on the path from the parent node to the root node.

Descendants. The recursively collected set of all child nodes and their child nodes starting at one or more nodes.

Siblings. The set of child nodes with the same parent node.

Node Depth. The length of the path to the root node excluding the node itself.

Tree Depth. The maximum node depth across all nodes of a tree.

Tree Level. One level of a tree is the set of nodes with the same node depth. This is sometimes called a tree slice.

While the tree is the main data structure to derive treemap visualizations, actual datasets may be augmented with other structural information. To this end, a dataset that is used for treemaps is not limited to one type of edges to make it tree-structed, but there has to be one type of edge that allows for an arborescent view on the graph. Thus, some algorithms and techniques make use of the following concepts:

Compound Tree. A tree with additional relations between nodes that are not used as parent-child relationships.

Neighborhood Information. The availability of additional relations between sibling nodes that should be considered during node layouting and visualization.

Thereby, neighborhood information may be relevant for treemap layouting, but general additional relationships are more prominently used for superimposed edge visualization [43].

Spatialization Terms. The layout of a treemap is one of its unique features. Hence, the layouting process and its characteristics have specific termini. In graph theory, the layouting process is also known as *graph drawing* [28].

Treemap Layout. A set of polytopes within the dimensionality of the reference space, e.g., polygons for 2-dimensional layouts, that are associated with the nodes of a tree. The polytopes adhere to the property of containment.

Treemap Layout Element. A single polytope of a treemap layout, usually associated with the node it represents.

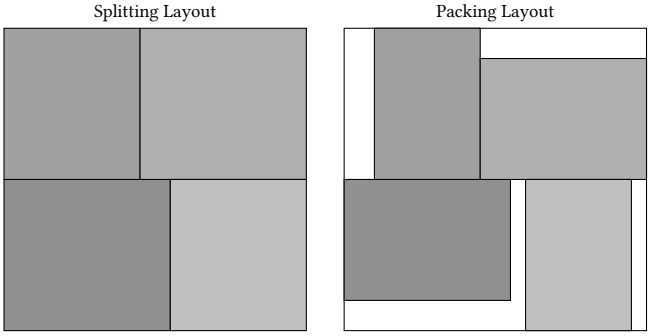

Figure 3: Conceptual display of splitting vs. packing layouts. Splitting layout algorithms utilize the whole space of a layout element for subdivision whereas packing layouts usually contain white space.

Treemap Layout Algorithm. An algorithm that maps tree-structured data to a treemap layout.

Space-filling. This property indicates that a layout algorithm generates layouts where the allocated space is fully used.

Geo-dependent. A layout algorithm is geo-dependent if the generated layouts adhere to given geo-spatial constraints, e.g., geo-referenced neighbors [16, 81]. A synonym for this characteristic is *spatially dependent.*

Packing Approach. A layout algorithm is a packing algorithm if the layout of an inner node is derived from the layout of its child nodes [75] (Figure 3).

Splitting Approach. A layout algorithm is a splitting algorithm if the layout of a node is derived from the layout of its parent node [75] (Figure 3). This process is also called dissection [62].

3 TREEMAP LAYOUT ALGORITHMS

Layouting is the mapping of a node into a reference space which typically is one-dimensional, two-dimensional or three-dimensional. This layouting is considered a recursive algorithm, whereby a treemap algorithm usually defines the layout for one level of the tree. For an algorithm to count as treemap layout algorithm, we require it to represent the parent-child relationship between nodes by containment, i.e., to layout a child node within the layout of the parent node. Besides the parent-child relationship, other information such as a size attribute or neighborhood information among child nodes can be used as well.

3.1 Definition of a Treemap Layout Algorithm

The task to identify a treemap layout algorithm depends on two definitions. First, it must be determinable if a resulting layout is in fact a treemap layout, i.e., if it adheres to a definition of a treemap layout. In order to provide a more specific yet minimal working definition, we consider a treemap layout as follows:

Definition 3.1. Treemap Layout
A treemap layout associates a treemap layout element for each node of a tree. For all layout elements and their associated nodes, the following conditions must be met.

(i) Layout elements of any sibling nodes p and q must not overlap.
(ii) If p is ancestor of q, then the layout element of q is fully enclosed by the layout element of p.

This ensures that the layout element of a child node is located within the layout element of its parent node, providing the containment property, and does not overlap with any sibling layout element. The latter property allows for an unambiguous visual mapping of the parent-child relationship. With respect to Definition 3.1, a treemap layout algorithm is an algorithm or function that maps a tree to a treemap layout. This results in a large set of algorithms to be considered treemap layout algorithms. In this paper, we focus on algorithms that derive such a layout from a rooted tree, while not using another treemap layout for the same tree as additional input.

Definition 3.2. Treemap Layout Algorithm
A treemap layout algorithm performs a graph drawing on a tree and results in an association of tree nodes to layout elements. For this, the input of the algorithm is a tree. As additional input, this algorithm does not use another treemap layout for the same input tree.

Algorithms that use such a pre-existing treemap layout for the rooted tree, we call *treemap layout postprocessing algorithms.*

3.2 Layout Algorithm Characteristics

With respect to the scope of this paper, we assess the layout algorithms by a small set of properties. As first classification, "one can discern two major layout methodologies: *subdivision* and *packing*" [75]. The basic approach for *packing layouts* is the arrangement of the polytopes of all child nodes in proximity and defining an enclosing polytope as own layout. This approach is then applied recursively from leaf nodes to the root node. Regarding *splitting layouts*, the basic approach is to define a polytope for the root node and perform a dissection on its layout for its child nodes, recursing down the tree. This basically means that all algorithms that perform a packing or subdivision of lines, areas, volumes, or even higher-dimensional shapes, are suitable for recursion and thus, for treemap layouting. Next to the layouting methodology, the dimensionality of the reference space for the layout is important, as it mainly restricts further visual variable mapping. For example, a 2.5D treemap, where the layout provides the space for 3D polytopes and the height of the polytopes is intended for height mapping, requires the layout to be one-dimensional or two-dimensional.

Another set of properties is the type of additional data an algorithm uses when layouting a tree. Here, we use four different types of input data: (1) to use node weights for layout sizes, (2) to use neighborhood information to place layout elements in adjecency, (3) to use a similarity measure to place layouts in vicinity, and (4) to use another treemap layout of another rooted tree as starting point. Especially for the node weights, we assess if the algorithms use the weight values to scale the sizes of leaf nodes. Typically, the size of parent nodes is adjusted to the spatial consumption of the child nodes. Next, the type of shapes an algorithm chooses for the resulting layout is often a key filter of choice. Typically, the class of shapes resembles either round shapes (points, circles, ellipsoids),

axis-aligned shapes (lines, rectangles, boxes), other convex shapes (triangles, polygons, polytopes), and general non-convex shapes.

There are more characteristics of treemap layout algorithms that may help to discern them and select one, e.g., the distinction between slicable and non-slicable layouts [108], or the algorithms' run-time complexity (note that "the generic problem of creating such layouts fall[s] in the category of NP-hard problems" [8]). However, slicable layouts as well as run-time complexity in general are scarcely discussed within the community of information visualization. For this paper, we deliberately omitted additional, external assessment of these layouts. Summarizing, we assess treemap layout algorithms along the following, non-exhaustive axes:

- Iteration approach: packing and splitting
- Additional attributes: weight, neighborhood-preserving, and similarity
- Layout shape: circular, rectangular, convex, and non-convex
- Reference space dimensionality: 1D, 2D, and 3D

3.3 Algorithm Overview

The collected treemap layout algorithms (as defined with Definition 3.2) and their properties are provided in Table 1. This introduction and classification is broadly categorized by (1) structure-focused layouts, (2) list-focused layout algorithms, (3) rectangular splitting layouts, (4) rectangular packing layouts, (5) rectangular 3D layouts, (6) circular layouts, (7) polygonal layouts, (8) similarity map layouts, (9) cartograms, and (10) generalized layout algorithms. For a broader perspective, we briefly include layout algorithms used outside of information visualization, too.

Structure-focused Layouts. By use of nesting in the layout, some algorithms can be considered treemap layout algorithms, although only the structure of the tree is depicted, i.e., no additional visual variable mapping of attributes is used. Examples for this are *Bubble Trees* [12], aesthetics-improved higraphs [39], and *horizontal-vertical inclusion tree* layouts [67]. Although most treemap layout algorithms allow for weight mapping of nodes to different surface areas, equal-weight mapping algorithms such as *Treeline* enable for basic depiction and straight-forward implementation [51].

List-focused Layout Algorithms. Although treemap layout algorithms are usually researched and discussed with recursion in mind, there is a number of algorithms that were initially designed to solve the layout of a degenerated case: the list. For the general layout methodology of *packing* layouts, bin packing [47] and specialized variations as guillotinable bin packing [50] can be used to generate treemap layouts. One subject for optimization is the aspect ratio and area of the resulting parent rectangle [14]. For the methodology of *splitting* layouts, the basic subdivision of a rectangle can be done using simple partitioning, as it is done within percentage bar charts, i.e., a one-dimensional partitioning using fractions. For more sophisticated layouts, mathematicians tries to find and prove optimal dissections of rectangles with associated weights [31, 62]. If the order and neighborhood information of nodes within a layout are important, the method of graph dualization can be used to layout them within a rectangle [107].

Rectangular Splitting Layouts. One of the first treemap layout algorithms is the formerly unnamed *Slice and Dice* algorithm [47].

Table 1: List of treemap layout algorithms (as defined with Definition 3.2) in approximated publication order.

Algorithm	Year	Packing	Splitting	Weight	Neighborhood	Similarity	Circular	Rectangular	Convex	Non-Convex	1D	2D	3D
Slice-and-Dice [47]	1991		•	•				•				•	
1-D Partitioning [46]	1993		•	•				•			•		
n-D Partitioning [46]	1993		•	•				•			•	•	•
Information Cube [68]	1993		•	•				•					•
Interactive Dynamic Map [110]	1994		•	•				•				•	
Guillotineable Bin Packing [50]	1995	•						•				•	
Hierarchical Clustering [44]	1998	•						•				•	
Cluster [100]	1999		•	•		•		•				•	
Modifiable [94]	2000		•	•				•				•	
Bubble Tree [12]	2000	•					•					•	
Squarified [15]	2000		•	•				•				•	
Pivot [79]	2001		•	•				•				•	
BinaryTree [78]	2001		•	•				•				•	
Blob Layout [39]	2002		•					•				•	
Plateau Placement [1]	2002	•						•				•	
Strip [9]	2002		•	•				•				•	
Quantum [9]	2002		•					•				•	
InfoSky [2]	2002		•	•					•			•	
(h-v) Inclusion Tree [67]	2003	•						•				•	
Strip TreeCube [88]	2003		•	•				•					•
Pivot TreeCube [88]	2003		•	•				•					•
Quantum TreeCube [88]	2003		•					•					•
Pebbles [103]	2003	•		•			•					•	
Data-Jewelry Box [105]	2003	•						•				•	
Template-based Packing [105]	2003	•						•				•	
Nested Circles [7]	2004		•				•					•	
RecMap V1 [42]	2004		•	•				•				•	
RecMap V2 [42]	2004	•		•	•			•				•	
Jigsaw [101]	2005		•	•						•		•	
Voronoi [6]	2005		•	•					•			•	
EncCon [63]	2005		•	•				•				•	
Circular Treemap [66]	2005	•					•					•	
ID-Map [37]	2005		•	•				•				•	
Treeline [51]	2005		•							•		•	
Circle Packing [98]	2006	•					•					•	
Matrix [95]	2006		•					•				•	
Grid [72]	2006		•							•		•	
Component Coupling [13]	2006	•				•		•				•	
Spiral [89]	2007		•	•				•				•	
Snake [89]	2007		•	•				•				•	
Approximation [62]	2007		•	•				•				•	
Multiresolution Grid Layout [38]	2007		•					•				•	
CodeCity [102]	2007	•		•				•				•	
Ellimaps [20]	2007		•	•			•					•	
Deterministic Hierarchical Clustering [96]	2007	•		•		•		•				•	
Density-guided [56]	2008		•	•				•				•	
Hierarchical Circular Partition [64]	2008		•	•					•			•	
Document Page Blocks [24]	2008		•					•				•	
Document Page Fit [24]	2008		•	•				•				•	
Fixed-outline Floorplanner [41]	2008		•	•				•				•	
CGD Ellimaps [65]	2009		•	•			•					•	
Evo-Streets [86]	2010	•		•				•				•	
Convex [23]	2011		•	•					•			•	
Ortho-Convex [23]	2011		•	•						•		•	
sqTM Burst Mapping [92]	2011		•	•				•				•	
Strip Packing [48]	2012	•		•				•				•	
Angular Treemap [52]	2012		•	•					•			•	
GosperMap [4]	2013		•	•						•		•	
Hilbert [87]	2013		•	•				•				•	
Moore [87]	2013		•	•				•				•	
Divide and Conquer Treemap [53]	2013		•	•						•		•	
iMap [97]	2013		•	•				•				•	
Fat Polygonal Partition [22]	2013		•	•					•			•	
Approximation by Impact [31]	2014		•	•				•				•	
Nmap [27]	2014		•	•	•			•				•	
Template-based Splitting [49]	2014		•	•				•				•	
Hexagon Tiling [11]	2014	•		•		•				•		•	
Variational Circular [109]	2015	•		•			•					•	
Enhanced Hexagon Tiling [106]	2015	•				•				•		•	
Weighted Map [33]	2015		•	•	•							•	
IsoMatch [30]	2015		•			•		•				•	
Code Map [40]	2015	•		•	•							•	
Squarified+ [18]	2016		•	•				•				•	
Golden Rectangle [58]	2017		•	•				•				•	
Bubble [34]	2017	•		•			•					•	
Box-connected Map [17]	2017		•	•	•					•		•	
EvoCells [71]	2018	•		•				•				•	
y-Soft Packing [14]	2018	•		•				•				•	
Greedy Insertion [93]	2018		•	•				•				•	
Orthogonal Voronoi [99]	2019		•	•						•		•	
Balanced Partitioning [29]	2019		•	•				•				•	

This algorithm uses a straight-forward dissection approach: "splitting the screen into rectangles in alternating horizontal and vertical directions as you traverse down the levels" [78]. The resulting long and thin rectangles are often-critiqued [36], and thus, a large number of splitting algorithms were proposed, each with advantages and disadvantages. The *Interactive Dynamic Map* integrates an algorithm that approximates rectangles with an aspect ratio of one [110]. The prolonged and hard-to-read rectangles were further handled using user-controlled aspect ratios by means of *modifiable* treemaps [94].

Extending on the interactive dynamic map, the approach of *Squarified* treemaps uses an up-front sorting of nodes by weight to improve the aspect ratios even further [15]. An alternative to the target aspect ratio one is the golden ratio—used by the *Golden Rectangle* layout [58]. Combined approaches for order-preservation and *good*—and this is still subject to research—aspect ratios are the *Strip* and *Pivot* layouts by Shneiderman and Wattenberg [79]. A layout algorithm can be adjusted to allow for quantized rectangle areas, allowing for grid layout of uniformly-sized leaf nodes [9]. The extension *Squarified+* provides an aspect-ratio optimizing combination of *Squarified* and *Strip* layouts [18].

Although not scientifically published, the *BinaryTree* layout is implemented in the Java library from the University of Maryland [78]. This divide-and-conquer approach splits the list of nodes in half and assigns rectangles in proportion to their summed weights, while switching split directions during recursion. With balanced partitioning treemaps, this design space of split position and node sorting was extended [29]. As some sort of binary tree, the treemap layout algorithm *Fractal Figures* can be used to highlight the number of nodes [21]. Further improving aspect ratios in treemap layouts, the *density-guided* approach of Liu et al. uses brute force to detect the best possible split [56].

The notion of strips to lay out rectangles for treemaps was continued by Tu and Shen with *S-shape* and *Spiral* layouts, both being strip layouts where the next strip continues in reverse direction and in a spiral pattern, respectively [89]. Both algorithms optimize an additional property that is desired with treemaps: *continuity*, i.e., preserved adjacency among sibling nodes. The approach with spiral and snake treemaps was further improved by more sophisticated space-filling curves (here: Hilbert and Moore curves) [87]. Originally published as layout algorithm for optimized node-link drawings, the *EncCon* [63] layout combines the squarified approach with *Spiral Treemaps*.

Using more regular layouts, the *ID-Map* [37], the *Matrix Layout* [95], and the *Multiresolution Grid Layout* [38] are primarily used for business charts. Likewise, the *iMap* layout allows for regular subdivision with quantized leaf rectangle sizes [97]. Kokash et al. proposed an approach that ensure uniform treemap layouts by applying an explicit splitting scheme [49]. Thereby, they propose that the layout template is either specified by the user or the result of a constraint solver. Improving on this, the *Greedy Insertion* approach operates on the derived layout tree and inserts new nodes next to the element with worst aspect ratio [93]. Although most algorithms target one specific dimensionality for the layout and use different approaches for dissection, Johnson presented a general *nD Partitioning* algorithm for arbitrary-dimensioned layout spaces in his dissertation [46]. The special case in one dimension, the *1D partitioning* [46], is used for other implicit edge-representation tree visualizations such as *icicle plots* and *sunburst views* [75].

Rectangular Packing Layouts. As an addition to basic adjacent placement (bin packing), one of the earliest algorithms uses two areas for its *plateau placement*: one area for additional inner nodes and one area for leaf nodes [1]. A specialized bin packing for equal-width leaf nodes is proposed with the *Strip Packing* algorithm [48]. Reusing a former layout, the *Data Jewelry Box* algorithm [105] uses computational geometry to place new nodes as it reuses previously layouted nodes. By applying the evolution of data to a rectangular treemap layout, the *EvoCells* algorithm adjusts the parent rectangles for each change on leaf nodes by displacement and packing [71]. A specialized combination of sorted nodes, quantized rectangle sizes and bin packing is the approach to generate *CodeCities* [102]. Although the depiction of the *Software Landscapes* are not a treemap visualization by means of splitting treemaps or containment treemaps—categories $\mathcal{T}_{\overline{S}}$ and $\mathcal{T}_{\overline{C}}$—, the underlying layout algorithm named EvoStreets is a hierarchical packing algorithm, and thus, a treemap layout algorithm [86].

Rectangular 3D Layouts. In contrast to 2D rectangular treemap layout algorithms, some algorithms compute 3D layouts. These algorithms are basically similar to their 2D counterparts. The packing approach is used for the *Information Cube* [68] and early splitting algorithms such as Slice'n'Dice, Ordered, Strip, and *Quantum* are proposed and discussed as well [88].

Circular Layouts. Next to rectangular treemap algorithms and their higher-dimensional complements, circles and ellipses can serve as basic shapes as well. One of the first approaches is the *Pebbles* treemap [103], that is a recursive packing of circles. This packing can be optimized, e.g., using a *front-line* approach [98], derived from a power diagram [109], or force-based contraction [34]. In contrast to more space-efficient packing, using more space to enhance the depiction of the tree structure is feasible for smaller datasets as well [66]. A splitting algorithm for circular treemap layouts was proposed by Balzer et al. [7]. Treemaps with a base layout shape of ellipses follows the concept of Venn diagrams and are hinted by Johnson and Shneiderman [47]. In contrast to circular layouts, they are suitable for layout creation using splitting algorithms, as proposed with *Ellimaps* [20] and optimized through combined geometrical distortions [65].

Polygonal Layouts. Polygonal layouts are characterized through use of more complex shapes for nodes, such as triangles, non-axis-aligned rectangles, or convex and non-convex polygons. An early example for the use of convex polygons is the *InfoSky* technique that uses "a modified version of the additively weighted power Voronoi diagram" [2]. These treemaps are termed *Voronoi treemap* [6]. Another approach—extending binary tree treemaps—uses scan lines to compute the polygon split [5]. When using arbitrary cutting angles, this approach results in *Angular Treemaps* [52] and their extension, the Divide and Conquer Treemaps [53]. Using space-filling curves with guaranteed locality are suitable for the creation of treemaps as well. A basic curve is a scanning of alternating rows and columns [72]. As more sophisticated examples, the Hilbert curve is used to create *Jigsaw* treemaps [101] and the Gosper curve is used for *GosperMaps* [4]. Allowing polygonal shapes improved

worst-case aspect ratios for *single layouts*. As such, the *hierarchical circular partition* is an approach for convex-polygonal treemaps [64]. Improving on this, de Berg et al. proposed *Convex* and *Ortho-Convex* treemaps [23] and *Fat Polygonal Partitioning* [22]. Buik-Aghai et al. proposed a map-like treemap layout algorithm that lays out nodes using adjacent hexagonal tiles [11]. This was later extended by the possibility of area-mapping [106]. More recently, the Voronoi layout approach is used to derive non-convex rectangular layouts [99].

Similarity-Map Layouts. Instead of subdividing by natural or given order, there are algorithms that arrange nodes based on their similarities. An early example is the *Cluster* treemap, proposed by Wattenberg [100], that uses a maximization on similarities for neighboring nodes. The most basic approach for this is brute force. More sophisticated approaches are energy-based approaches [13] such as spring embedders [11] and *IsoMatch* [30]. In contrast, dissimilarities are solved by large neighborhood search, resulting in a space-filling, box-connected map [17]. From the field of *artificial intelligence*, the self-optimizing maps can be used as base layout [80].

Cartograms. Originating from cartography, there are treemap layout algorithms that preserve underlying neighborhood information from geographic locations. Next to statistical cartograms from the early 19'th century, algorithmic descriptions for cartograms (both splitting and packing) have been proposed with the *RecMap* techniques [42]. Shortly after, algorithmic approximations on rectangular cartograms were introduced using constraint solver and linear programming [85]. For layouts that do not adhere to both tree-structure and neighborhood constraints, Buchin et al. proposed a transformation towards an *adjacency-preserving spatial treemap* [16]. A *slice and scale* approach improves the construction of neighborhood-preserving treemaps as it is used within *Nmap* treemaps [27]. The similiar approach *Weighted Maps* uses sorting by longitude and latitude coordinates to construct cartograms [33].

Generalized Layout Algorithms. Within recent years, the research community consolidated subgroups of layout algorithms and derived layout languages that cover design spaces of treemap layout algorithms. These generalized layout algorithms allow previously published algorithms to be expressed as special parameterization of a more generalized one. This was demonstrated for the subcategory of splitting treemap layout algorithms. As such, Baudel and Broeksema proposed an algorithm that captures "The Design Space of Sequential Space-Filling Layouts" [8]. Later, Schulz et al. proposed a "Generative Layout Approach for Rooted Tree Drawings" that supports layouts up to treemap category $\mathcal{T}_{\mathrm{MT}}$ [74].

Non-InfoVis Layouts. Loosely related to treemaps in information visualization, there are other layout techniques that perform some kind of subdivision of a surface. As they result from highly specialized domains, the applicability for information visualization may be restricted and we deliberately omitted these layout from the overview table. For example, the floorplanning community explores methods that optimize the placement of integrated circuits on a chip, whereby cuircits may have hierarchical nesting [41]. Following the quantum treemap approach to visualize multimedia items—that often comes with an fixed aspect ratio and view size of leaf node items—the layout of document pages is optimized using *Block* and *Fit* approaches [24]. In the domain of wireless submission via WiMAX, spatial subdivision of rectangles is important, too [82]. There, an adaption of *Squarified* layouts is used as an alternative layout algorithm to compute the burst windows [92], whereby their implementation and results resembles *Strip* layouts.

4 TREEMAP LAYOUT ADAPTATIONS

Although the treemap layout algorithm by itself is a basic building block in a tree visualization technique, the resulting layout is most often target to use-case-specific parameterization and further post-processing. This ranges from the selection of layout algorithms and the order of nodes for iteration to layout postprocessings for the transformation of geometry or the depiction of structure.

Choice of Layout Algorithm. Despite the variety in treemap layout algorithms and years of research and improvement, "no algorithm is superior in all [...] aspects and under all circumstances" [10]. Following this, the choice of the layout algorithm is subject to use case and task. This may even require visualization designers to select a general tree visualization technique [26]. For treemaps, typical considerations to take into account for layouting are additional relations and attributes. Use cases that rely on specific adjacency of child nodes are inclined to use cartograms or similarity maps, while others try to achieve general *readability* or other layout metrics. As the inherent property of a treemap is hierarchical recursion, "an algorithm could be chosen for each individual [inner] node" [95]. As of now, the used methods of layout algorithm selection are (1) equal for all nodes, (2) selected by a static scheme [59, 95], (3) data-dependent analysis [35] or machine learning [10], and (4) dynamic configuration [81].

Iteration Order. In addition to the selection of an algorithm, the order of node processing has influence on the result, too. There are a number of approaches for node iteration. For one, there is the inherent order given by the dataset [47], i.e., a technical order. There are semantic orders, such as "alphabetical order, chronological order, or other criteria" [3] or the order of nodes with respect to a specific attribute (e.g., sorted by node weight [15] or height [55]). Employing the neighborhood information can improve geo-dependent treemaps by sorting by either longitude and latitude coordinates [59] or by distance to an origin [104].

Layout Post-Processings. In contrast to layout algorithms that result in a subdivision of a surface there are techniques that build upon those techniques and perform operations on the resulting layout. As such, local and global transformations can be applied to a layout. Examples are the projection into other reference systems such as polar coordinates [46] and local parameterization to ensure uniform density among nodes [95]. Other local transformations include the resizing of layout nodes due to a degree-of-interest [77] or a degree of uncertainty [83]. Regarding the use of treemaps to visualize tree-structured data over a longer period of time, cognitive load can be decreased significantly using layouts that remain stable [61]. This resulted in approaches that re-use previously generated layouts as *template* for the computation of another one. One example is the computation of "the layout of the treemap using only local modifications" [84] by applying local swaps and approximate the layout of the previous instance.

Structure Depiction. A separate category of such layout post-processings is the introduction of additional space into the layout—especially used for splitting treemaps. One approach is the application of *nesting* to treemaps, which insets child nodes by an *offset* [47]. A special case of nesting is the *title bar* which introduces additional space, e.g., for labeling of inner nodes [90]. Another approach is the conversion of a *nested treemap* to a *cascaded treemap* [57]. Quite differently, a *burst* based on a node's depth can be applied to every node individually [72]. Typically, the area of leaf layouts is comparable and strictly correlates to the underlying weight values. It should be noted, however, that "visualizations of structure [...] invalidate this property" [57]. As it is usually necessary to "graphically portray the structure of the hierarchy" [46], nesting is applied nonetheless. One technique that allows for depiction of the structure while "maintaining the proportionality property of treemaps" is the *BeamTree* [91]. This technique converts a Slice'n'Dice treemap into a layout using implicit edge representations via overlap, converting a treemap layout of category $\mathcal{T}_{\mathrm{S}}$ into a $\mathcal{T}_{\mathrm{IE}}$ visualization.

5 CONCLUSIONS

In conclusion, the research community has published a multitude of treemap layout algorithms and adaptions for various use cases and tasks. Through a highlight of 81 of those algorithms we provide a broad overview. The majority of algorithms is designed for 2D layouts and rectangular shapes are most common. Among all algorithms, the number of packing and splitting approaches is mostly similar. Most notably, the majority of algorithms does neither account for similarity nor for neighborhood of nodes. This overview and introduction to the broad parameterization of treemap layout algorithms can be used by practitioners and researchers as a starting point for own research and visualization techniques and applications. We believe this review allows for more specific and more effective use of treemap visualization techniques. For example, we suggest that visualization APIs and libraries that target tree-structured data consider the reviewed algorithms for inclusion.

For future work, we intend to use this review as a starting point for more in-depth treemap algorithm evaluation and an assessment of treemap metrics. With this, we strive to extend the list of algorithms and evaluate them against further measurements and categories, such as run-time complexity, availability in visualization frameworks and libraries, and compatibility for combined usage with other layout algorithms. Another important direction is the collection of specific open questions in treemap layouts and deriving a research agenda. From our point of view, these are the next steps for an even broader use of the treemap visualization technique for researchers and visualization designers, as well as an improved comprehension of the visualization consumers, the users.

ACKNOWLEDGMENTS

We want to thank the anonymous reviewers for their valuable comments and suggestions to improve this article. This work is part of the "Software-DNA" project, which is funded by the European Regional Development Fund (ERDF – or EFRE in German) and the State of Brandenburg (ILB) as well as the "TASAM" project, which is funded by the German Federal Ministry for Economic Affairs and Energy (BMWi, ZIM).

REFERENCES

[1] Keith Andrews. 2002. Visual Exploration of Large Hierarchies with Information Pyramids. In *Proc. International Conference on Information Visualisation (iV '02)*. IEEE, 793–798. https://doi.org/10.1109/IV.2002.1028871

[2] Keith Andrews, Wolfgang Kienreich, Vedran Sabol, Jutta Becker, Georg Droschl, Frank Kappe, Michael Granitzer, Peter Auer, and Klaus Tochtermann. 2002. The InfoSky visual explorer: Exploiting Hierarchical Structure and Document Similarities. *Information Visualization* 1, 3-4 (2002), 166–181. https://doi.org/10.1057/PALGRAVE.IVS.9500023

[3] Keith Andrews, Josef Wolte, and Michael Pichler. 1997. Information Pyramids™: A new approach to visualizing large hierarchies. In *Proc. International Conference on Visualization (Vis '97)*. IEEE, 49–52.

[4] David Auber, C. Huet, A. Lambert, B. Renoust, A. Sallaberry, and A. Saulnier. 2013. GosperMap: Using a Gosper Curve for Laying Out Hierarchical Data. *TVCG* 19, 11 (2013), 1820–1832. https://doi.org/10.1109/TVCG.2013.91

[5] M. Balzer and Oliver Deussen. 2005. Exploring Relations within Software Systems Using Treemap Enhanced Hierarchical Graphs. In *Proc. 3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT '05)*. IEEE, 1–6. https://doi.org/10.1109/VISSOF.2005.1684312

[6] M. Balzer and Oliver Deussen. 2005. Voronoi treemaps. In *Proc. Symposium on Information Visualization (InfoVis '05)*. IEEE, 49–56. https://doi.org/10.1109/INFVIS.2005.1532128

[7] Michael Balzer, Andreas Noack, Oliver Deussen, and Claus Lewerentz. 2004. Software Landscapes: Visualizing the Structure of Large Software Systems. In *Proc. VGTC Symposium on Visualization (Vis '04)*. EG/IEEE. https://doi.org/10.2312/VisSym/VisSym04/261-266

[8] T. Baudel and B. Broeksema. 2012. Capturing the Design Space of Sequential Space-Filling Layouts. *TVCG* 18, 12 (2012), 2593–2602. https://doi.org/10.1109/TVCG.2012.205

[9] Benjamin B. Bederson, Ben Shneiderman, and Martin Wattenberg. 2002. Ordered and Quantum Treemaps: Making Effective Use of 2D Space to Display Hierarchies. *Transactions on Graphics* 21, 4 (2002), 833–854. https://doi.org/10.1145/571647.571649

[10] Joseph Bethge, Sebastian Hahn, and Jürgen Döllner. 2017. Improving Layout Quality by Mixing Treemap-Layouts Based on Data-Change Characteristics. In *Proc. Vision, Modeling & Visualization (VMV '17)*. EG. https://doi.org/10.2312/vmv.20171261

[11] Robert P. Biuk-Aghai, Cheong-Iao Pang, and Yain-Whar Si. 2014. Visualizing large-scale human collaboration in Wikipedia. *Future Generation Computer Systems* 31 (2014), 120–133. https://doi.org/10.1016/j.future.2013.04.001

[12] Richard Boardman. 2000. Bubble Trees the Visualization of Hierarchical Information Structures. In *Proc. Extended Abstracts on Human Factors in Computing Systems (CHI EA '00)*. ACM, 315–316. https://doi.org/10.1145/633292.633484

[13] Johannes Bohnet and Jürgen Döllner. 2006. Analyzing Feature Implementation by Visual Exploration of Architecturally-embedded Call-graphs. In *Proc. International Workshop on Dynamic Systems Analysis (WODA '06)*. ACM, 41–48. https://doi.org/10.1145/1138912.1138922

[14] Ulrich Brenner. 2018. y-Soft Packings of Rectangles. *Elsevier Computational Geometry* 70, C (2018), 49–64. https://doi.org/10.1016/j.comgeo.2018.01.005

[15] Mark Bruls, Kees Huizing, and Jarke J. van Wijk. 2000. Squarified Treemaps. In *Proc. TCVG Symposium on Visualization (Data Visualization '00)*. Springer, 33–42. https://doi.org/10.1007/978-3-7091-6783-0_4

[16] Kevin Buchin, David Eppstein, Maarten Löffler, Martin Nöllenburg, and Rodrigo I Silveira. 2011. Adjacency-Preserving Spatial Treemaps. In *Proc. Workshop on Algorithms and Data Structures (WADS '11)*. Springer, 159–170. https://doi.org/10.1007/978-3-642-22300-6_14

[17] Emilio Carrizosa, Vanesa Guerrero, and Dolores Romero Morales. 2017. Visualizing proportions and dissimilarities by Space-filling maps: A Large Neighborhood Search approach. *Computers & Operations Research* 78 (2017), 369–380. https://doi.org/10.1016/j.cor.2016.09.018

[18] Antonio Cesarano, Filomena Ferrucci, and Mario Torre. 2016. A heuristic extending the Squarified treemapping algorithm. *arXiv Computing Research Repository* abs/1609.00754 (2016).

[19] Emmanuel Chazard, Philippe Puech, Marc Gregoire, and Régis Beuscart. 2006. Using Treemaps to represent medical data. *Studies in Health Technology and Informatics* 124 (2006), 522–527.

[20] Pierre Collin, Benoit Otjacques, Xavier Gobert, Monique Noirhomme, and Fernand Feltz. 2007. Visualizing the activity of a web-based collaborative platform. In *Proc. 11th International Conference Information Visualization (iV '07)*. IEEE, 251–256. https://doi.org/10.1109/IV.2007.137

[21] M. D'Ambros, Michele Lanza, and H. Gall. 2005. Fractal Figures: Visualizing Development Effort for CVS Entities. In *Proc. 3rd International Workshop on Visualizing Software for Understanding and Analysis (VISSOFT '05)*. IEEE, 16. https://doi.org/10.1109/VISSOF.2005.1684303

[22] Mark de Berg, Krzysztof Onak, and Anastasios Sidiropoulos. 2013. Fat polygonal partitions with applications to visualization and embeddings. *Journal of Computational Geometry* 4, 1 (2013), 212–239.

[23] Mark de Berg, Bettina Speckmann, and Vincent van der Weele. 2011. Treemaps with Bounded Aspect Ratio. In *Proc. 22nd International Symposium on Algorithms and Computation (ISAAC '11)*. Springer, 260–270. https://doi.org/10.1007/978-3-642-25591-5_28

[24] João Batista S. de Oliveira. 2008. Two Algorithms for Automatic Document Page Layout. In *Proc. Symposium on Document Engineering (DocEng '08)*. ACM, 141–149. https://doi.org/10.1145/1410140.1410170

[25] Reinhard Diestel. 2010. *Graph Theory* (fourth ed.). Graduate Texts in Mathematics, Vol. 173. Springer.

[26] Y. Dong, A. Fauth, M. Huang, Y. Chen, and J. Liang. 2020. PansyTree: Merging Multiple Hierarchies. In *Proc. Pacific Visualization Symposium (PacificVis '20)*. IEEE, 131–135. https://doi.org/10.1109/PacificVis48177.2020.1007

[27] Felipe S. L. G. Duarte, Fabio Sikansi, Francisco M. Fatore, Samuel G. Fadel, and Fernando V. Paulovich. 2014. Nmap: A Novel Neighborhood Preservation Space-filling Algorithm. *TVCG* 20, 12 (2014), 2063–2071. https://doi.org/10.1109/TVCG.2014.2346276

[28] David Eppstein, Elena Mumford, Bettina Speckmann, and Kevin Verbeek. 2012. Area-Universal and Constrained Rectangular Layouts. *Journal on Computing* 41, 3 (2012), 537–564. https://doi.org/10.1137/110834032

[29] Cong Feng, Minglun Gong, Oliver Deussen, and Hui Huang. 2019. Treemapping via Balanced Partitioning. In *Proc. Computational Visual Media (CVM '19)*.

[30] O. Fried, S. DiVerdi, M. Halber, E. Sizikova, and A. Finkelstein. 2015. IsoMatch: Creating Informative Grid Layouts. *Computer Graphics Forum* 34, 2 (2015), 155–166. https://doi.org/10.1111/cgf.12549

[31] Armin Fügenschuh, Konstanty Junosza-Szaniawski, and Zbigniew Lonc. 2014. Exact and approximation algorithms for a soft rectangle packing problem. *Optimization* 63, 11 (2014), 1637–1663. https://doi.org/10.1080/02331934.2012.728217

[32] Henry Gannett. 1903. *Twelfth census of the United States, taken in the year 1900. Statistical atlas.* U.S. Census Office Washington. https://www.loc.gov/item/07019233/.

[33] Mohammad Ghoniem, Maël Cornil, Bertjan Broeksema, Mickaël Stefas, and Benoît Otjacques. 2015. Weighted maps: treemap visualization of geolocated quantitative data. In *Proc. Visualization and Data Analysis (SPIE, Vol. 9397)*. 1–15. https://doi.org/10.1117/12.2079420

[34] J. Görtler, C. Schulz, Daniel Weiskopf, and Oliver Deussen. 2018. Bubble Treemaps for Uncertainty Visualization. *TVCG* 24, 1 (2018), 719–728. https://doi.org/10.1109/TVCG.2017.2743959

[35] Sebastian Hahn and Jürgen Döllner. 2017. Hybrid-Treemap Layouting. In *Proc. EuroVis 2017 – Short Papers*. EG. https://doi.org/10.2312/eurovisshort.20171137

[36] Sebastian Hahn, Jonas Trümper, Dominik Moritz, and Jürgen Döllner. 2014. Visualization of varying hierarchies by stable layout of voronoi treemaps. In *Proc. International Conference on Information Visualization Theory and Applications (IVAPP '14)*. SciTePress, 50–58. https://doi.org/10.5220/0004686200500058

[37] M. C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. 2005. Importance-driven visualization layouts for large time series data. In *Proc. Symposium on Information Visualization (InfoVis '05)*. IEEE, 203–210. https://doi.org/10.1109/INFVIS.2005.1532148

[38] Ming C. Hao, Umeshwar Dayal, Daniel A. Keim, and Tobias Schreck. 2007. Multi-Resolution Techniques for Visual Exploration of Large Time-Series Data. In *Proc. VGTC Symposium on Visualization (EuroVis '07)*. EG, 27–34. https://doi.org/EG/IEEE

[39] David Harel and Gregory Yashchin. 2002. An algorithm for blob hierarchy layout. *The Visual Computer* 18, 3 (2002), 164–185. https://doi.org/10.1007/s003710100133

[40] N. Hawes, S. Marshall, and Craig Anslow. 2015. CodeSurveyor: Mapping large-scale software to aid in code comprehension. In *Proc. 3rd Working Conference on Software Visualization (VISSOFT '15)*. IEEE, 96–105. https://doi.org/10.1109/VISSOFT.2015.7332419

[41] Ou He, Sheqin Dong, Jinian Bian, Satoshi Goto, and Chung-Kuan Cheng. 2008. A Novel Fixed-outline Floorplanner with Zero Deadspace for Hierarchical Design. In *Proc. International Conference on Computer-Aided Design (ICCAD '08)*. IEEE/ACM, 16–23. https://doi.org/1509456.1509473

[42] R. Heilmann, Daniel A. Keim, C. Panse, and M. Sips. 2004. RecMap: Rectangular Map Approximations. In *Proc. Symposium on Information Visualization (InfoVis '04)*. IEEE, 33–40. https://doi.org/10.1109/INFVIS.2004.57

[43] Danny Holten. 2006. Hierarchical Edge Bundles: Visualization of Adjacency Relations in Hierarchical Data. *TVCG* 12, 5 (2006), 741–748. https://doi.org/10.1109/TVCG.2006.147

[44] Mao Lin Huang and Peter Eades. 1998. A Fully Animated Interactive System for Clustering and Navigating Huge Graphs. In *Proc. Graph Drawing (GD '98)*. Springer, 374–383. https://doi.org/10.1007/3-540-37623-2_29

[45] M. Jern, J. Rogstadius, and T. Åström. 2009. Treemaps and Choropleth Maps Applied to Regional Hierarchical Statistical Data. In *Proc. 13th International Conference on Information Visualisation (iV '09)*. IEEE, 403–410. https://doi.org/10.1109/IV.2009.97

[46] Brian Scott Johnson. 1993. *Treemaps: Visualizing Hierarchical and Categorical Data*. Ph.D. Dissertation. University of Maryland. UMI Order No. GAX94-25057.

[47] Brian Scott Johnson and Ben Shneiderman. 1991. Tree-Maps: A Space-filling Approach to the Visualization of Hierarchical Information Structures. In *Proc. 2nd Conference on Visualization (Vis '91)*. IEEE, 284–291. https://doi.org/10.5555/949607.949654

[48] A. Kobayashi, K. Misue, and J. Tanaka. 2012. Edge Equalized Treemaps. In *Proc. 16th International Conference on Information Visualisation (iV '12)*. IEEE, 7–12. https://doi.org/10.1109/IV.2012.12

[49] N. Kokash, B. de Bono, and J. Kok. 2014. Template-based treemaps to preserve spatial constraints. In *Proc. International Conference on Information Visualization Theory and Applications (IVAPP '14)*. SciTePress, 39–49. https://doi.org/10.5220/0004684900390049

[50] Berthold Kröger. 1995. Guillotineable bin packing: A genetic approach. *European Journal of Operational Research* 84, 3 (1995), 645–661. https://doi.org/10.1016/0377-2217(95)00029-P

[51] Guillaume Langelier, Houari Sahraoui, and Pierre Poulin. 2005. Visualization-based Analysis of Quality for Large-scale Software Systems. In *Proc. 20th International Conference on Automated Software Engineering (ASE '05)*. ACM, 214–223. https://doi.org/10.1145/1101908.1101941

[52] J. Liang, Quang Vinh Nguyen, S. Simoff, and Mao Lin Huang. 2012. Angular Treemaps – A New Technique for Visualizing and Emphasizing Hierarchical Structures. In *Proc. 16th International Conference on Information Visualisation (IV '12)*. IEEE, 74–80. https://doi.org/10.1109/IV.2012.23

[53] Jie Liang, Simeon Simoff, Quang Vinh Nguyen, and Mao Lin Huang. 2013. Visualizing Large Trees with Divide & Conquer Partition. In *Proc. 6th International Symposium on Visual Information Communication and Interaction (VINCI '13)*. ACM, 79–87. https://doi.org/10.1145/2493102.2493112

[54] Daniel Limberger, Willy Scheibel, Jürgen Döllner, and Matthias Trapp. 2019. Advanced Visual Metaphors and Techniques for Software Maps. In *Proc. 11th International Symposium on Visual Information Communication and Interaction (VINCI '19)*. ACM, 11:1–8. https://doi.org/10.1145/3356422.3356444

[55] Daniel Limberger, Willy Scheibel, Matthias Trapp, and Jürgen Döllner. 2017. Mixed-Projection Treemaps: A Novel Approach Mixing 2D and 2.5D Treemaps. In *Proc. 21st International Conference Information Visualisation (IV '17)*. IEEE, 164–169. https://doi.org/10.1109/iV.2017.67

[56] S. Liu, N. Cao, and H. Lv. 2008. Interactive Visual Analysis of the NSF Funding Information. In *Proc. Pacific Visualization Symposium (PacificVis '08)*. IEEE, 183–190. https://doi.org/10.1109/PACIFICVIS.2008.4475475

[57] Hao Lü and James Fogarty. 2008. Cascaded Treemaps: Examining the Visibility and Stability of Structure in Treemaps. In *Proc. Graphics Interface (GI '08)*. Canadian Information Processing Society, 259–266. https://doi.org/10.5555/1375714.1375758

[58] Liangfu Lu, Shiliang Fan, Mao Lin Huang, Weidong Huang, and Ruolan Yang. 2017. Golden Rectangle Treemap. *Journal of Physics: Conference Series* 787, 1 (2017), 012007. https://doi.org/10.1088/1742-6596/787/1/012007

[59] Florian Mansmann, Daniel A. Keim, Stephen C. North, Brian Rexroad, and Daniel Sheleheda. 2007. Visual Analysis of Network Traffic for Resource Planning, Interactive Monitoring, and Interpretation of Security Threats. *TVCG* 13, 6 (2007), 1105–1112. https://doi.org/10.1109/TVCG.2007.70522

[60] Liam McNabb and Robert S. Laramee. 2017. Survey of Surveys (SoS) – Mapping The Landscape of Survey Papers in Information Visualization. *Computer Graphics Forum* 36, 3 (2017), 589–617. https://doi.org/10.1111/cgf.13212

[61] Kazuo Misue, Peter Eades, Wei Lai, and Kozo Sugiyama. 1995. Layout Adjustment and the Mental Map. *Journal of Visual Languages & Computing* 6, 2 (1995), 183–210. https://doi.org/10.1006/jvlc.1995.1010

[62] Hiroshi Nagamochi and Yuusuke Abe. 2007. An approximation algorithm for dissecting a rectangle into rectangles with specified areas. *Discrete Applied Mathematics* 155, 4 (2007), 523–537. https://doi.org/10.1016/j.dam.2006.08.005

[63] Quang Vinh Nguyen and Mao Lin Huang. 2005. EncCon: An Approach to Constructing Interactive Visualization of Large Hierarchical Data. *Information Visualization* 4, 1 (2005), 1–21. https://doi.org/10.1057/palgrave.ivs.9500087

[64] Krzysztof Onak and Anastasios Sidiropoulos. 2008. Circular Partitions with Applications to Visualization and Embeddings. In *Proc. 24th Annual Symposium on Computational Geometry (SCG '08)*. ACM, 28–37. https://doi.org/10.1145/1377676.1377683

[65] Benoît Otjacques, Maël Cornil, Monique Noirhomme, and Fernand Feltz. 2009. ´. In *Proc. Conference on Human-Computer Interaction (INTERACT '09)*. Springer, 805–818. https://doi.org/10.1007/978-3-642-03658-3_84

[66] Bijan Parsia, Taowei Wang, and Jennifer Golbeck. 2005. Visualizing web ontologies with cropcircles. In *Proc. 4th International Semantic Web Conference (SWC '05)*.

[67] K. Pulo, P. Eades, and M. Takatsuka. 2003. Smooth structural zooming of h-v inclusion tree layouts. In *Proc. International Conference on Coordinated and Multiple Views in Exploratory Visualization (CMV '03)*. IEEE, 14–25. https://doi.org/10.1109/CMV.2003.1214999

[68] Jun Rekimoto and Mark Green. 1993. The information cube: Using transparency in 3d information visualization. In *Proc. 3rd Annual Workshop on Information Technologies & Systems (WITS '93)*. 125–132.

[69] Richard C. Roberts and Robert S. Laramee. 2018. Visualising Business Data: A Survey. *Information* 9, 11 (2018), 54. https://doi.org/10.3390/info9110285

[70] Willy Scheibel., Matthias Trapp, Daniel Limberger., and Jürgen Döllner. 2020. A Taxonomy of Treemap Visualization Techniques. In *Proc. 15th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (IVAPP '20)*. SciTePress, 273–280. https://doi.org/10.5220/0009153902730280

[71] Willy Scheibel, Christopher Weyand, and Jürgen Döllner. 2018. EvoCells – A Treemap Layout Algorithm for Evolving Tree Data. In *Proc. 9th International Conference on Information Visualization Theory and Applications (IVAPP '18)*. SciTePress, 273–280. https://doi.org/10.5220/0006617102730280

[72] Tobias Schreck, Daniel A. Keim, and Florian Mansmann. 2006. Regular TreeMap Layouts for Visual Analysis of Hierarchical Data. In *Proc. 22nd Spring Conference on Computer Graphics (SCCG '06)*. ACM, 183–190. https://doi.org/10.1145/2602161.2602183

[73] Hans-Jörg Schulz. 2011. Treevis. net: A tree visualization reference. *Computer Graphics and Applications* 6 (2011), 11–15. https://doi.org/10.1109/MCG.2011.103

[74] H. J. Schulz, Z. Akbar, and F. Maurer. 2013. A generative layout approach for rooted tree drawings. In *Proc. Pacific Visualization Symposium (PacificVis '13)*. IEEE, 225–232. https://doi.org/10.1109/PacificVis.2013.6596149

[75] Hans-Jorg Schulz, Steffen Hadlak, and Heidrun Schumann. 2011. The design space of implicit hierarchy visualization: A survey. *TVCG* 17, 4 (2011), 393–411. https://doi.org/10.1109/TVCG.2010.79

[76] Hans-Jörg Schulz, Martin Luboschik, and Heidrun Schumann. 2007. Interactive poster: Exploration of the 3d treemap design space. In *Symposium on Information Visualization '07*. IEEE.

[77] Kang Shi, Pourang Irani, and Pak Ching Li. 2007. Facilitating Visual Queries in the TreeMap Using Distortion Techniques. In *Proc. Symposium on Human Interface: Human Interface and the Management of Information. Methods, Techniques and Tools in Information Design*. Springer, 345–353. https://doi.org/10.1007/978-3-540-73345-4_39

[78] Ben Shneiderman. 2009. *Treemaps for space-constrained visualization of hierarchies*. Technical Report. Human-Computer Interaction Lab. http://www.cs.umd.edu/hcil/treemap-history.

[79] Ben Shneiderman and Martin Wattenberg. 2001. Ordered treemap layouts. In *Proc. Symposium on Information Visualization (InfoVis '01)*. IEEE, 73–78. https://doi.org/10.1109/INFVIS.2001.963283

[80] A. Skupin. 2002. A Cartographic Approach to Visualizing Conference Abstracts. *Computer Graphics and Applications* 22, 1 (2002), 50–58. https://doi.org/10.1109/38.974518

[81] Aidan Slingsby, Jason Dykes, and Jo Wood. 2009. Configuring Hierarchical Layouts to Address Research Questions. *TVCG* 15, 6 (2009), 977–984. https://doi.org/10.1109/TVCG.2009.128

[82] Chakchai So-In, Raj Jain, and Abdel-Karim Al Tamimi. 2009. OCSA: An algorithm for burst mapping in IEEE 802.16 e mobile WiMAX networks. In *Proc. 15th Asia-Pacific Conference on Communications (APCC '09)*. IEEE, 52–58. https://doi.org/10.1109/APCC.2009.5375688

[83] M. Sondag, W. Meulemans, C. Schulz, K. Verbeek, D. Weiskopf, and B. Speckmann. 2020. Uncertainty Treemaps. In *Proc. Pacific Visualization Symposium (PacificVis '20)*. IEEE, 111–120. https://doi.org/10.1109/PacificVis48177.2020.7614

[84] M. Sondag, Bettina Speckmann, and K. Verbeek. 2018. Stable Treemaps via Local Moves. *TVCG* 24, 1 (2018), 729–738. https://doi.org/10.1109/TVCG.2017.2745140

[85] Bettina Speckmann, Marc van Kreveld, and Sander Florisson. 2006. A Linear Programming Approach to Rectangular Cartograms. In *Proc. International Symposium on Spatial Data Handling*. Springer, 529–546. https://doi.org/10.1007/3-540-35589-8_34

[86] Frank Steinbrückner and Claus Lewerentz. 2010. Representing Development History in Software Cities. In *Proc. 5th International Symposium on Software Visualization (SoftVis '10)*. ACM, 193–202. https://doi.org/10.1145/1879211.1879239

[87] S. Tak and A. Cockburn. 2013. Enhanced Spatial Stability with Hilbert and Moore Treemaps. *TVCG* 19, 1 (2013), 141–148. https://doi.org/10.1109/TVCG.2012.108

[88] Y. Tanaka, Y. Okada, and K. Niijima. 2003. Treecube: visualization tool for browsing 3D multimedia data. In *Proc. 7th International Conference on Information Visualization (iV '03)*. IEEE, 427–432. https://doi.org/10.1109/IV.2003.1218020

[89] Y. Tu and H. W. Shen. 2007. Visualizing Changes of Hierarchical Data using Treemaps. *TVCG* 13, 6 (2007), 1286–1293. https://doi.org/10.1109/TVCG.2007.70529

[90] David Turo and Brian Scott Johnson. 1992. Improving the Visualization of Hierarchies with Treemaps: Design Issues and Experimentation. In *Proc. 3rd Conference on Visualization (Vis '92)*. IEEE, 124–131. https://doi.org/10.1109/VISUAL.1992.235217

[91] Frank van Ham and Jarke J. van Wijk. 2003. Beamtrees: Compact Visualization of Large Hierarchies. *Information Visualization* 2, 1 (2003), 31–39. https://doi.org/10.1057/palgrave.ivs.9500036

[92] J. Vanderpypen and L. Schumacher. 2011. Treemap-Based Burst Mapping Algorithm for Downlink Mobile WiMAX Systems. In *Proc. Vehicular Technology Conference (VTC '11)*. IEEE, 1–5. https://doi.org/10.1109/VETECF.2011.6093072

[93] Eduardo Faccin Vernier, Joao Comba, and Alexandru C. Telea. 2018. A Stable Greedy Insertion Treemap Algorithm for Software Evolution Visualization. In *Proc. Conference on Graphics, Patterns and Images (SIBGRAPI '18)*. IEEE. https://doi.org/10.1109/SIBGRAPI.2018.00027

[94] Frédéric Vernier and Laurence Nigay. 2000. Modifiable Treemaps Containing Variable-Shaped Units. In *Proc. Information Visualization – Extended Abstracts (InfoVis '00)*. IEEE.

[95] Roel Vliegen, Jarke J. van Wijk, and E. J. van der Linden. 2006. Visualizing Business Data with Generalized Treemaps. *TVCG* 12, 5 (2006), 789–796. https://doi.org/10.1109/TVCG.2006.200

[96] Richard Vuduc, Thomas Panas, Thomas Epperly, Andreas Saebjornsen, and Daniel Quinlan. 2007. Communicating Software Architecture using a Unified Single-View Visualization. In *Proc. International Conference on Engineering of Complex Computer Systems (ICECCS '07)*. IEEE, 217–228. https://doi.org/10.1109/ICECCS.2007.20

[97] Chaoli Wang, John P. Reese, Huan Zhang, Jun Tao, and Robert J. Nemiroff. 2013. iMap: a stable layout for navigating large image collections with embedded search. In *Proc. Visualization and Data Analysis (VDA '13)*. IS&T/SPIE, 8654:1–14. https://doi.org/10.1117/12.999313

[98] Weixin Wang, Hui Wang, Guozhong Dai, and Hongan Wang. 2006. Visualization of Large Hierarchical Data by Circle Packing. In *Proc. SIGCHI Conference on Human Factors in Computing Systems (CHI '06)*. ACM, 517–520. https://doi.org/10.1145/1124772.1124851

[99] Yan-Chao Wang, Feng Lin, and Hock-Soon Seah. 2019. Orthogonal Voronoi Diagram and Treemap. *arXiv Computing Research Repository* (2019). http://arxiv.org/abs/1904.02348

[100] Martin Wattenberg. 1999. Visualizing the Stock Market. In *Proc. CHI Extended Abstracts on Human Factors in Computing Systems (CHI EA '99)*. ACM, 188–189. https://doi.org/10.1145/632716.632834

[101] Martin Wattenberg. 2005. A note on space-filling visualizations and space-filling curves. In *Proc. Symposium on Information Visualization (InfoVis '05)*. IEEE, 181–186. https://doi.org/10.1109/INFVIS.2005.1532145

[102] Richard Wettel and Michele Lanza. 2007. Program Comprehension Through Software Habitability. In *Proc. 15th International Conference on Program Comprehension (ICPC '07)*. IEEE, 231–240. https://doi.org/10.1109/ICPC.2007.30

[103] Kai Wetzel. 2003. pebbles - using Circular Treemaps to visualize disk usage. Online. http://lip.sourceforge.net/ctreemap.html.

[104] Jo Wood and Jason Dykes. 2008. Spatially Ordered Treemaps. *TVCG* 14, 6 (2008), 1348–1355. https://doi.org/10.1109/TVCG.2008.165

[105] Yumi Yamaguchi and Takayuki Itoh. 2003. Visualization of distributed processes using "Data Jewelry Box" algorithm. In *Proc. Computer Graphics International (CGI '03)*. IEEE, 162–169. https://doi.org/10.1109/CGI.2003.1214461

[106] Muye Yang and Robert P. Biuk-Aghai. 2015. Enhanced Hexagon-Tiling Algorithm for Map-Like Information Visualisation. In *Proc. 8th International Symposium on Visual Information Communication and Interaction (VINCI '15)*. ACM, 137–142. https://doi.org/10.1145/2801040.2801056

[107] G. Yeap and M. Sarrafzadeh. 1995. Sliceable Floorplanning by Graph Dualization. *Discrete Mathematics* 8, 2 (1995), 258–280. https://doi.org/10.1137/S0895480191266700

[108] E. F. Y. Young, C. C. N. Chu, and Z. C. Shen. 2003. Twin binary sequences: a nonredundant representation for general nonslicing floorplan. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 22, 4 (2003), 457–469. https://doi.org/10.1109/TCAD.2003.809651

[109] Haisen Zhao and L. Lu. 2015. Variational circular treemaps for interactive visualization of hierarchical data. In *Proc. Pacific Visualization Symposium (PacificVis '15)*. IEEE, 81–85. https://doi.org/10.1109/PACIFICVIS.2015.7156360

[110] Mountaz Zizi and Michel Beaudouin-Lafon. 1994. Accessing Hyperdocuments Through Interactive Dynamic Maps. In *Proc. European Conference on Hypermedia Technology (ECHT '94)*. ACM, 126–135. https://doi.org/10.1145/192757.192786