


2D Embeddings of Multi-dimensional Partitionings

Marina Evers  and Lars Linsen 

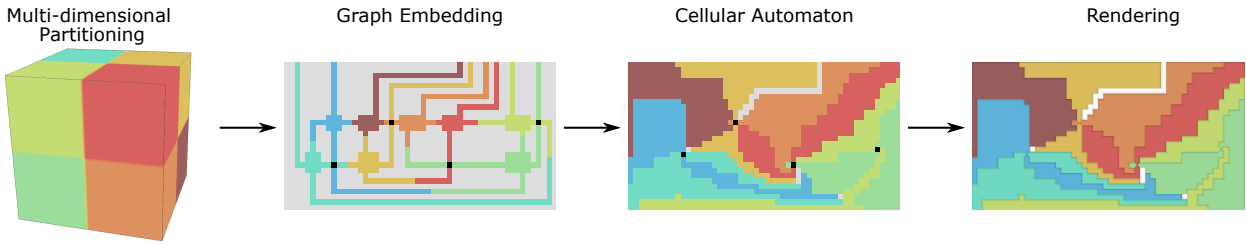


Fig. 1: A multi-dimensional partitioning is modeled as a graph that is embedded into a 2D plane. The graph embedding is used as a starting point for computing an area- and boundary-length-preserving layout of the partitioning using a cellular automaton approach. To its outcome, we apply a rendering that highlights relevant features.

Abstract—Partitionings (or segmentations) divide a given domain into disjoint connected regions whose union forms again the entire domain. Multi-dimensional partitionings occur, for example, when analyzing parameter spaces of simulation models, where each segment of the partitioning represents a region of similar model behavior. Having computed a partitioning, one is commonly interested in understanding how large the segments are and which segments lie next to each other. While visual representations of 2D domain partitionings that reveal sizes and neighborhoods are straightforward, this is no longer the case when considering multi-dimensional domains of three or more dimensions. We propose an algorithm for computing 2D embeddings of multi-dimensional partitionings. The embedding shall have the following properties: It shall maintain the topology of the partitioning and optimize the area sizes and joint boundary lengths of the embedded segments to match the respective sizes and lengths in the multi-dimensional domain. We demonstrate the effectiveness of our approach by applying it to different use cases, including the visual exploration of 3D spatial domain segmentations and multi-dimensional parameter space partitionings of simulation ensembles. We numerically evaluate our algorithm with respect to how well sizes and lengths are preserved depending on the dimensionality of the domain and the number of segments.

Index Terms—Multi-dimensional partitionings, segmentations, dimensionality reduction, parameter space visualization.

1 INTRODUCTION

Given a multi-dimensional domain, partitionings or segmentations split the domain into connected regions. The regions are disjoint, yet neighboring regions share a boundary. The union of the regions represents the entire domain again. Such multi-dimensional partitionings occur, for example, in parameter space analysis of simulation ensembles. Here, a common analysis goal is to identify regions in parameter space that share a common behavior in the simulation outcome. Especially for parameter spaces with more than three dimensions, it becomes challenging to understand the structure of the partitioning. Besides investigating the number of segments, it is of interest to understand how large the parameter-space region for each segment is, as this property provides information about the stability of the simulation output with respect to the input parameters. Additionally, the neighborhood relations between different segments provide information about possible transitions between different behaviors. Thus, a visualization of the partitioning should display all segments simultaneously, should reveal neighborhoods, and should visualize segment sizes and boundary lengths of neighboring segments to cover the most important structural information of the partitioning.

When analyzing image segmentations, the segmentation outputs are commonly visualized by assigning different colors to different segments, which works well for 2D domains, but not anymore for higher-dimensional domains, not even for 3D segmentations. Despite the fact that 3D images are commonly acquired and segmented in

different fields such as medicine, geology, or material science, there exists no approach, to our knowledge, that globally visualizes the segmentation output similar to 2D domains.

We present an approach for visualizing a multi-dimensional partitioning or segmentation in a 2D embedding. Similar to 2D segmentation visualizations, we encode segments in the 2D embedding by color. The embedding represents all segments, preserves the topology of the multi-dimensional segmentation, and optimizes for the preservation of segment sizes and boundary lengths. The computation of the embedding is based on a graph representation of the multi-dimensional segmentation and a respective graph embedding. The optimization is based on a cellular automaton approach. The output of the optimization is visualized by a shading technique for boundaries and a visual encoding of edge crossings, which cannot be avoided in case of non-planar graphs.

Our main contributions can be summarized by:

- An algorithm for computing a 2D embedding of a multi-dimensional segmentation that preserves the topology of the given multi-dimensional segmentation.
- An optimization of the embedding to generate segment areas and boundary lengths that match segment sizes and boundary sizes of the multi-dimensional segmentation.
- A rendering to highlight features of the embedding such as segment boundaries and edge crossings.
- Application of our approach to visualizing 3D spatial domain segmentations and n -dimensional parameter space partitionings.
- An in-depth evaluation of the algorithm's performance based on its optimization criteria.

2 RELATED WORK

Segmentation of 3D images are ubiquitous in medical image analysis, among others. Visualizing the segmentation output is typically performed by showing 2D slices of the volume or focusing on individual

• All authors are with the University of Münster, Germany.
E-mail: {marina.evers|linsen}@uni-muenster.de.

Manuscript received xx xxx. 201x; accepted xx xxx. 201x. Date of Publication xx xxx. 201x; date of current version xx xxx. 201x. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org.
Digital Object Identifier: xx.xxx/TVCG.201x.xxxxxx

segments of interest [36], i.e., not providing a global representation of the segmentation output. Similarly, in volume renderings, transfer functions are applied to assign colors and opacities to regions of interest to alleviate the inherent occlusion problem, but also do not provide a visual representation of the entire 3D segmentation.

Multi-dimensional spaces beyond 3D occur in simulation ensemble analysis [13, 51] when studying the parameter space of the ensembles. **Parameter space partitionings** represent the different behaviors that the simulated phenomenon may have. Sedlmair et al. [39] present a framework for visual parameter space analysis that identifies partitionings as one of the core analysis tasks. Paraglide [4] provides an interactive visual analysis of parameter space partitionings using scatterplots, but does not focus on understanding the partitioning’s structure in the multi-dimensional space. Other works [35, 41] use projections of parameter spaces, but operate on individual points instead of partitionings. Further techniques for the visualization of parameter spaces are radial layouts [7] and glyph layouts [5], which do not explicitly show the partitioning with its topology. This also holds for parallel coordinates, which are a common tool in visualizing multi-dimensional data and have been adapted for investigating parameter spaces [34, 52]. Closest to our work are the following two approaches for direct visualization of partitionings: Evers et al. [17] use an enriched hyper-slicer for a distortion-free visualization of multi-dimensional parameter space partitionings. However, their approach does not provide a high-level overview of all segments including topology and segment sizes. Fernandes et al. [18] study region transitions in partitionings by a glyph-based approach. They include areas of the segments and boundary sizes between them, but as the glyphs are relatively complex, the approach does not scale well to a higher number of segments. However, none of these approaches provides an overview of the multi-dimensional partitioning.

Our approach uses a graph representation of the multi-dimensional segmentation. Deriving a **graph from a segmentation** has been proposed by Ren et al. [37] to visualize joint layouts for segmented meshes. However, they do not aim at visualizing the properties of the segments. After deriving the graph from a segmentation, the graph needs to be visualized while preserving the segmentation’s properties. Various graph drawing techniques with different characteristics have been discussed [14, 20, 24, 30, 38, 44, 49, 50]. Common techniques include force-directed layouts [21, 27], layered graph drawing [42] and orthogonal layouts [3].

For **graph visualization**, besides classical node-link diagrams, different space-filling visualization techniques have been proposed. Treemaps [40] visualize weighted trees and provide a good sense of size-related tasks [19, 47], but our segmentation does not resemble a tree structure. Also, techniques that focus on visualizing clusterings in graphs are related to our problem. They include BubbleSets [12], LineSets [2], and KelpFusion [31], which add regions around previously placed vertices. GMap [22, 25] creates map-like visualizations of clustered graphs, but the layout might be highly fragmented. While this problem can be overcome by modifying the layout or the clustering [28], the resulting visualization does not ensure joint boundaries in the visualization in case of connections between the clusters. MapSets [16] targets a similar problem as GMap, but might lead to complex regional layouts.

Preserving areas in a given topology is strongly related to **cartograms** [33]. Layouts where shared boundaries present neighborhoods have been proposed [1, 55], but they only apply to planar graphs. Wu et al. [53] propose a visualization method for clustered graphs that deal with vertices belonging to more than one cluster by drawing connections on top of the layout. However, these connections are structurally different from the visualization of the clusters. Thus, in our case, a similar approach to dealing with non-planar graphs is not desirable.

3 OVERVIEW

We propose a visual encoding that represents a multi-dimensional segmentation and its most important features. As the driving application scenario for the development of our approach is the analysis of simulation ensembles, we derive tasks and corresponding optimization criteria based on analyzing multi-dimensional parameter spaces. While

the goal of analyzing multi-dimensional parameter spaces motivates our design choices, the developed approach is nevertheless generally applicable to the visualization of any domain segmentation.

Motivated by the application to the analysis of multi-dimensional parameter spaces, our overarching goal is to obtain an overview of the segmentation, identified as a relevant task in previous work [17]. To achieve this goal, we identified the following tasks:

T1: Identify neighborhood relations. The topological structure of a segmentation provides important information. In particular, it reveals which transitions between parameter space segments are possible.

T2: Understand the relative sizes of the segments. When analyzing a multi-dimensional parameter space, the users should be able to identify which portion of the parameter space corresponds to the respective segment, i.e., how likely the respective behavior is.

T3: Understand the relative sizes of the boundaries between segments. The sizes of the boundaries indicate how large the parameter ranges are that allow for direct transitions between the two considered segments. Our visualization should enable users to estimate whether there is a strong connection between the segments, which corresponds to a large boundary size, or whether the segments are only slightly connected.

Based on these tasks, we can define the following properties for our visual design:

Preserving Topology. The topology of a segmentation is defined by neighborhood relationships between the segments, thus, preserving it addresses Task T1.

Representing Segment Sizes. The sizes of multi-dimensional segments might vary significantly. Therefore, the sizes should also be represented in the lower-dimensional embedding, addressing Task T2.

Representing Segment Boundary Sizes. To address Task T3, the embedding should be optimized for the boundary sizes. Note that in this paper we refer to the boundary sizes as boundary lengths to avoid confusion with the segment sizes, even though only in 2D they are actually lengths.

Note that we do not consider the shapes of the segments for our design, as it is infeasible to represent multi-dimensional shapes in a single 2D embedding.

Approach. We propose an embedding algorithm for visualizing a multi-dimensional segmentation in two dimensions. The algorithm consists of several steps shown schematically in Fig. 1. As input, we consider a segmented multi-dimensional volume of dimensionality n , where each segment represents a connected component and is labeled with a unique ID. The segments partition the n -dimensional domain such that each point in the n -dimensional domain is assigned to exactly one segment. Without loss of generality, we assume that the domain is given in a discrete setting in the form of a regular grid, i.e., each grid point stores the assigned segment’s ID. This assumption is only made for implementation purposes. If the multi-dimensional segmentation is given in another format, it can be easily resampled to a regular grid.

The computation of the embedding (see Sec. 4) is based on a graph representation of the multi-dimensional segmentation (see Sec. 4.1). The graph’s vertices represent the segments, its edges represent neighborhood information, segment sizes are stored as vertex weights, and boundary lengths are stored as edge weights. We, then, compute a 2D embedding of the graph that minimizes edge crossings. Afterwards, the graph embedding is used as input to a cellular automaton approach, which operates on a 2D grid of cells that is initialized with a drawing of the graph embedding and eventually labels each cell with a segment ID. The cellular automaton approach iteratively optimizes the area size of the segments and the boundary lengths between segments to match the segment sizes and boundary sizes of the multi-dimensional segmentation while preserving the topological structure (see Sec. 4.2). We choose a cellular automaton over other optimization algorithms because it can guarantee topology preservation and can be applied to the discrete, non-differentiable segmentation data. Moreover, other algorithm choices that would be applicable, such as genetic algorithms or simulated annealing, are usually not more efficient. Finally, we present a rendering approach for the output of the cellular automaton that highlights its important features (see Sec. 5).

4 EMBEDDING ALGORITHM

4.1 Graph Representation

The topological structure of the segmentation S can be represented by a weighted, undirected graph $G = (V, E)$. Each segment $s_i \in S$ corresponds to a vertex $v_i \in V$, and a shared boundary of two segments $s_i, s_j \in S$ is represented by an edge $(v_i, v_j) \in E$. In addition, there are also segments that lie at the border of the given domain. This property should also be preserved. Therefore, we add an additional vertex v_b representing the outer border of the domain. For each vertex v_i representing a segment connected to this outer border, we add a new edge (v_i, v_b) to the graph. For our implementation that assumes a segmentation over a regular grid, we assume two segments to be neighbors, if and only if two grid cells of the two segments share a common face along one of the dimensions, i.e. in 3D, cells that only share a vertex or an edge are not considered to be neighbors. In the context of cellular automata, this is referred to as von Neumann neighborhoods. For higher-dimensional data, the neighborhood structure of von Neumann neighbors is very sparse as only the neighbors along a single dimension are considered [56]. For the applications presented in this work, this does not impose problems since the dimensionality of none of the presented examples is very high. Additionally, the neighborhood criterium for creating the graph and computing the boundary lengths can be easily exchanged without influencing the later stages of the algorithm. We further enhance our graph representation with weights for vertices and edges. The size A_i of a segment s_i is stored as a weight of vertex v_i , which later will be used to preserve the corresponding area in the 2D embedding. The size A_i is computed by counting the number of n -dimensional cells that the segment consists of. The size of the boundary between two segments s_i and s_j is stored as a weight B_{ij} of edge (v_i, v_j) and is computed by counting the number of cells that are von Neumann neighbors.

Given a graph G that represents the multi-dimensional segmentation, we next want to embed this graph into two dimensions, where the embedding shall minimize edge crossings. Edge crossings make a graph layout more complex and, thus, harder to understand. Additionally, they represent a structure that does not appear in the multi-dimensional space, but is an artifact of the 2D embedding. However, the problem of minimizing the number of edge crossings is NP-hard [23]. Therefore, we opt for a suitable approximation algorithm using a planarization method [44]. For simplicity of the rasterization in the next steps, we use an orthogonal graph drawing algorithm that assures that the boundary vertex v_b is placed close to the boundary. We then create a rasterization of the graph embedding to create an initial configuration for the cellular automaton. An example of such an initial configuration is shown as the graph embedding in Fig. 1. More details on the graph embedding and creating the initial configuration can be found in the supplementary material. While the graph drawing already provides a 2D embedding of the segmentation's topology, it does not yet reflect the segment sizes and boundary sizes, which we will add next.

4.2 Cellular Automaton

The graph embedding places the graph vertices in a 2D space and draws connecting lines for all graph edges. We aim for a visualization that resembles a 2D segmentation visualization such that the mental abstraction to other representations, such as graph drawings, is not necessary. Using the graph embedding as an initial configuration, we want to transform it to a representation where (1) the graph vertices $v_i \in V$ become 2D segments with an area according to the vertex weights A_i and (2) the graph edges $(v_i, v_j) \in E$ become shared segment boundaries between the 2D segments belonging to v_i and v_j with a boundary length according to the edge weight B_{ij} (cf. Fig. 1). The first goal is related to that of generating cartograms. We, therefore, build upon an approach by Dorling [15] using a cellular automaton. While this approach was criticized for not preserving shapes [33], this is not an issue for us, as we cannot, generally, preserve the shape of n -dimensional segments in 2D. Moreover, we extend the approach by Dorling to also fulfill the second goal of boundary length optimization listed above.

The cellular automaton consists of cells arranged in a regular 2D

grid. Each cell of the cellular automaton has a state that can change based on a set of rules that depends on the state of the neighboring cells. An iterative process allows for the optimization of a target function. For our application, each cell's state corresponds to one of the segment IDs. If the cell is not assigned to any segment (in the following referred to as background cells), we assign the ID -1 . If the cell represents an edge crossing (black dots in Fig. 1), we assign the ID -2 . Given the initial configuration, the cellular automaton iteratively changes the state of the cells facilitating the local optimization according to some optimization criteria.

Optimization Criteria. When applying the cellular automaton approach to the initial configuration, each cell can change its state (i.e., its label ID) based on the states of the surrounding cells in each iteration. During the iterative process, we want to optimize the area of the segments and the shared boundary lengths between segments while preserving the topology.

To optimize the *segments' areas*, we compute for each cell the deviation of the size of the 2D segment it belongs to from the size of the n -dimensional segment that it corresponds to. The deviation for segment s is computed by

$$d_{A,s} = \frac{A_{s,nD}}{A_{nD}} - \frac{A_{s,2D}}{A_{2D}}, \quad (1)$$

where $A_{s,nD}$ denotes the size of the n -dimensional segment that corresponds to s , A_{nD} the overall size of the partitioned multi-dimensional domain, $A_{s,2D}$ the current size of segment s in the 2D embedding (computed as the number of cells assigned to segment s), and A_{2D} the size of the 2D embedding (computed as the total number of cells of the cellular automaton). Here, $d_{A,s} > 0$ denotes that the segment s needs to expand further to represent the original partitioning accurately. Having computed the deviation $d_{A,s}$ for all segments s , we determine whether a cell should change its state by investigating, if at least one of the neighboring cells has a higher deviation. A cell with a neighbor with higher area deviation will be marked for a state change from its current segment ID to the neighboring segment ID.

To optimize for shared *boundary lengths*, we employ a similar approach. However, the boundary deviation is not computed per segment, but per pair of segments. In analogy to the segment size deviation, we compute the boundary length deviation as the difference of the relative boundary length in high-dimensional space to the relative boundary length in the 2D embedding. Thus, the boundary length deviation between segments s_i and s_j is computed by

$$d_{L,(s_i,s_j)} = \frac{L_{(s_i,s_j),nD}}{L_{nD}} - \frac{L_{(s_i,s_j),2D}}{L_{2D}}, \quad (2)$$

where $L_{(s_i,s_j),nD}$ denotes the boundary length between the n -dimensional segments corresponding to segments s_i and s_j , L_{nD} the sum of the lengths of all boundaries in the n -dimensional space, $L_{(s_i,s_j),2D}$ the boundary length between segments s_i and s_j in the 2D embedding, and L_{2D} the sum of the lengths of all boundaries in the 2D embedding. Here, $d_{L,(s_i,s_j)} > 0$ denotes that the length of the boundary needs to increase. Note that for each cell, we can compute a boundary deviation for each neighbor leading to four boundary deviations per cell. However, just changing the segment to which the cell belongs does not necessarily change the length of the boundary. For example, the boundary length between the blue and the green segment in Fig. 2a is identical even though one cell has a different state. Therefore, we compute the change in boundary length $L_{(s_i,s_j),2D}$ when changing the label of the current cell from segment s_i to segment s_j by $\Delta L_{(s_i,s_j)} = N_{s_i} - N_{s_j}$, where N_{s_i} is the number of neighbors of the current cell that belong to segment s_i and N_{s_j} the number of neighbors of the current cell that belong to segment s_j . Then, the cell is marked for a state change, if and only if $d_{L,(s_i,s_j)} \Delta L_{(s_i,s_j)} > 0$. In this case, the boundary of the 2D segment is proportionally larger than in the multi-dimensional space ($d_{L,(s_i,s_j)} < 0$) and the boundary length would be reduced ($\Delta L_{(s_i,s_j)} < 0$), or the boundary is too short ($d_{L,(s_i,s_j)} > 0$) while the length of the boundary can be expanded ($\Delta L_{(s_i,s_j)} > 0$).

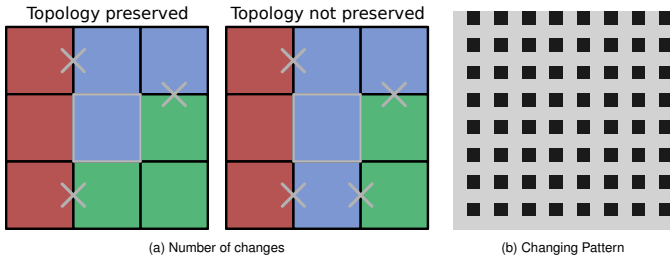


Fig. 2: (a) Topology is preserved, if the number of segment changes (grey crosses) when clockwise traversing the Moore neighborhood of the current cell (grey boundary) is ≤ 3 . Otherwise, topology might change. (b) To prevent topology violations by simultaneously changing neighboring cells, only non-adjacent cells may change simultaneously.

If a cell was marked for a state change, the *target segment* to which the cell should change needs to be defined. Here, we check the area as well as the boundaries to the four neighbors of the cell and choose the criterion with the most significant deviation to determine the target segment of the cell. Note that the initial configuration contains background cells that we want to vanish. Thus, we set their area deviation to -1 independent of the background’s size. This corresponds to the minimum value and would push the cellular automaton towards changing the background cells’ state to other segments, leading to a vanishing background.

While these optimization criteria do not consider the shape of the segments, more compact shapes are, generally, easier to interpret than line-like structures. To achieve compact shapes, we use the *security factor* introduced by Dorling [15]. The security factor considers the cell’s neighborhood and measures how exposed the cell is to the corresponding segments. Each von Neumann neighbor that belongs to the same segment adds a value of 3 to the security factor. Here, we also consider the neighbors on the diagonal, which add a value of 1. Thus, we obtain a value between 0 for an isolated cell label and 16 for a cell completely surrounded by cells of the same segment. We only allow cells to change their state, if the security factor lies below the threshold of 11. A detailed investigation of the choice of the security factor and its influence on the result can be found in Sec. 6.2.

Topology Preservation. Besides optimizing the areas and boundary sizes, the cellular automaton should preserve the topology, for which we follow the approach by Dorling [15] with some adaptations. It can be tested if a cell is critical to preserve the topology by counting the number of segment changes that occur when traversing the cell’s Moore neighborhood, i.e., its 8-connected neighbors, in clockwise order. If the number of segment changes is larger than 3, the cell is critical for the topology and should not be changed, see Fig. 2a. To preserve the topology on the grid’s border, we also count the transition to the border as a segment change. Additionally, isolated cells should not be allowed to disappear. As we initially start with a background that should disappear over time, we need to adapt the behavior for the background cells. Note that only changing the state of background cells cannot change the topology because of the topology preservation criterion: If the state change of a background cell leads to a new shared boundary of two segments, the number of segment changes when traversing the background cell’s neighborhood must be at least 4 and is therefore marked as critical. This restriction could lead to background cells not vanishing during the iterative process. Therefore, we separately check for the background cells to investigate, if a change would destroy the topology based on the original graph. In some cases, especially for complex, non-planar graphs, some of the background cells might remain to separate two segments. As removing them would lead to topological changes, we keep them as separating boundaries and treat them differently in the visualization to distinguish them from the segments (see Sec. 5). Note that the cells that denote an edge crossing are never considered for change because, by definition, they contain at least 4 changes when traversing the cell’s neighborhood.

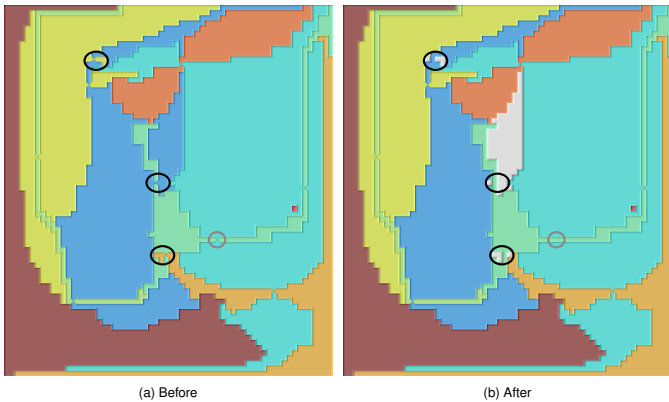


Fig. 3: To improve the embedding, crossings between segments that become unnecessary can be removed. These crossings are either duplicates (grey circle) or crossings where one side is not required anymore to preserve the topology (black circles).

Iterative Optimization. Following the criteria described above, the 2D embedding is generated iteratively. To avoid conflicts by simultaneously changing neighboring cells, Dorling [15] proposed a *checkerboard pattern* where diagonal neighbors change simultaneously. However, applying a standard checkerboard pattern might lead to topology violations in the Moore neighborhood. Even though we compute the topology for the von Neumann neighborhood, we want to avoid diagonal neighbors in the embedding to avoid confusions. Thus, we apply a changing pattern that does not allow direct or diagonal neighbors to change simultaneously, which leads to a pattern as shown in Fig. 2b. This pattern is iteratively shifted such that each cell may be changed in every fourth iteration.

Using an iterative algorithm, we need to define some *convergence criterion* to determine when the algorithm should terminate. In general, the user can set a maximum number of iterations. However, we want the cellular automaton to terminate early, if no further improvements can be made. For this purpose, we check after each iteration, if the states changed. If no further changes occur for four iterations, the algorithm has converged. However, one may observe an oscillation between different states without improvement. Therefore, we apply a damping that gets stronger with increased accuracy. For the damping, we define a probability that a state switches. This probability is defined as the absolute value of the maximal deviation of either the areas or the boundaries multiplied by a user-defined scaling factor g . A higher value of g indicates a higher tolerance to the deviation and, thus, a weaker damping. Then, the cell switches its state with the calculated probability. However, the stochastic nature of this approach might cause the algorithm to stop early. To prevent this, we change the threshold for stopping the algorithm to ten iterations without changes, which is also a parameter that can be adjusted by the user.

As we are using a graph layout for node-link diagrams but then generate a dense visualization of the segmentation embedding instead, it may occur that some *edge crossings* created during the initial graph layout are no longer necessary. Here, we can differentiate between two cases: First, two segments could cross twice, where one of the crossings could be removed (see grey circles in Fig. 3), but we need to ensure that none of the segments is split. Second, an edge crossing was necessary to connect to another segment or the border, but due to the disappearance of the background, this connection is now occurring in another part of the segment. In this case, the part of the segment whose topological information can be fully replicated by the other part can be removed together with the corresponding edge crossing (see black circles in Fig. 3). This is implemented by setting the states of the corresponding cells back to the background, which creates additional space for the surrounding segments to spread further. As testing the conditions for removal is computationally expensive, we do not perform this step every iteration but in a user-defined frequency. For our results,

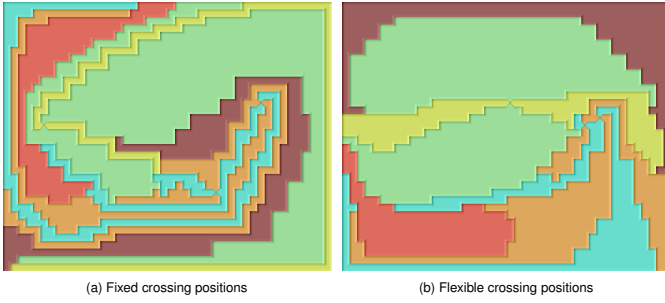


Fig. 4: Keeping the edge crossing position fixed might result in unnecessarily complex embeddings (a), while allowing them to move leads to more compact segments (b). Both examples have been created based on the same initialization.

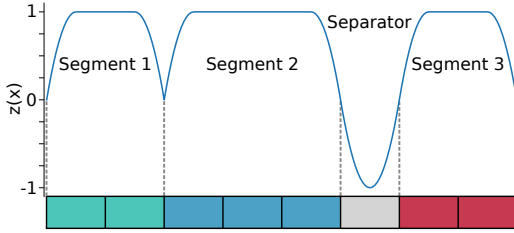


Fig. 5: We create a height profile with plateaus in the center of the segments and valleys for separators between segments. Here, we set the height $h = 1$ and the width for height changes $w = 0.5r$, where r is the number of pixels for each cell of the cellular automaton.

we compute the check every 300th iteration.

The *positions of the edge crossings* are fixed in the layout. However, allowing them to change their positions can significantly improve the layout as shown in Fig. 4. Keeping the line crossings fix provides less flexibility for the relaxation and, thus, might induce more line-like structures. Thus, we allow them to move by switching the state of the cell containing the edge crossing with the state of a neighboring cell. The movement is only allowed, if the topology is preserved. If a vertical or horizontal movement is possible, we compute the barycenter of the involved segment and move the edge crossing in the direction towards the segment’s barycenter, as moving towards the barycenter generally allows for more compact segments.

The evolution of our algorithm over time, including the different additional improvements discussed in this section, is shown in an animation in the supplementary material.

5 RENDERING

Inspired by the renderings of 2D segmentations, we visualize the 2D embedding by assigning to each segment one *color*. The colors may be assigned according to some categorical color map (i.e., trying to use distinguishable colors) but may also encode some other (meta-)information about the data, for example, indicating which segments lead to a similar simulation output when analyzing multi-dimensional parameter space partitionings. To still clearly observe the segments even in case of adjacent segments with similar colors, we emphasize boundaries by applying a *shading* inspired by cushion treemaps [46, 48]. The shading should allow us to identify the unique segments independent of the color coding to provide a large flexibility for using different colors. Additionally, the shading should allow for a better differentiation of the separators (regions of background cells that remain to separate different segments) between segments and shared segment boundaries. Unlike cushion treemaps, we do not want to visualize a hierarchy, but aim at a relatively simple visualization without adding too much visual complexity. We follow the idea of cushion treemaps and use quadratic functions to model the height, but use a piecewise definition to keep the inner part of the segments flat. Our piecewise

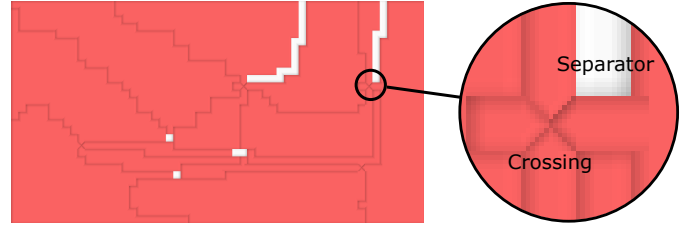


Fig. 6: Shading the segments and visually encoding special features like edge crossings and background areas allows for a visualization of segments independent of the color coding.

height function $z_i(x)$ in one dimension is shown in Fig. 5. We use an increase in height from the shared boundaries of the different segments, while we use a decrease for the separators. We also choose a color for the separators that is clearly distinguishable from all segment colors.

In the following, we will describe our shading based on the x -direction of the image. The y -direction is treated identically. To model the desired behavior, we apply the constraints $z_i(x_{i,1}) = 0$, $z_i(x_{i,1} + w) = h$ and $\frac{dz_i}{dx}(x_{i,1} + w) = 0$ where w is the width of the region where the quadratic increase in height should occur, $x_{i,1}$ is the position where the segment i starts, and h denotes the final height of the segments. Analogous constraints are applied at the end of the segment, where the height decreases. For the separators, which are modeled as valleys with a negative height, we set the constraints $z_v(x_{v,2} - w) = -h$, $\frac{dz_v}{dx}(x_{v,2}) = 0$ and $\frac{dz_v}{dx}(x_{v,2}) = \frac{dz_i}{dx}(x_{i,1})$ where $x_{v,2}$ denotes the position where the separator v ends. Again, analogous constraints are applied at the start of the separators. These constraints can be used to compute the normals in these regions as $(2(ax + b), 2(ay + c), 1)$, where a , b , and c are the coefficients that can be computed between the boundaries of the segments or separators x_1 and x_2 in x -direction and y_1 and y_2 in the y -direction as follows:

$$a = \begin{cases} 0 & x_1 + w < x < x_2 - w \\ -\frac{sh}{w^2} & \text{otherwise} \end{cases},$$

$$b = \begin{cases} \frac{sh}{w^2}(x_1 + w) & x_1 \leq x \leq x_1 + w \\ 0 & x_1 + w < x < x_2 - w \\ \frac{sh}{w^2}(x_2 - w) & x_2 - w \leq x \leq x_2 \end{cases},$$

$$c = \begin{cases} \frac{sh}{w^2}(y_1 + w) & y_1 \leq y \leq y_1 + w \\ 0 & y_1 + w < y < y_2 - w \\ \frac{sh}{w^2}(y_2 - w) & y_2 - w \leq y \leq y_2 \end{cases},$$

where $s = 1$, if the corresponding region is a segment, and $s = -1$, if the region corresponds to a segment separator.

By applying this kind of rendering, a wide range of color coding options can be used. Fig. 6 shows a visualization where all segments contain the same color. Still, the different segments and the edge crossings are visible. The *edge crossings* are visually encoded as a cross where we apply a similar shading as for the segment boundaries. Thus, the crossings are visible but are not highlighted.

6 ALGORITHMIC EVALUATION

In the following, we will investigate the influences of the different optimization criteria of our approach as well as its scalability with the number of segments and the dimensionality of the input data. Two synthetic datasets allow us to tune the characteristics as desired.

As a first dataset D_1 , we create a 2D or 3D image segmentation with a user-defined resolution, an adaptable number of segments, and varying segment shapes. For each segment, we randomly choose a seed point and a random growth rate. Then, each segment is expanded by adding surrounding, still unassigned cells using the growth rate as a probability. As a second dataset D_2 , we use a simple 3D cube that is divided once in all dimensions leading to eight, equally sized segments

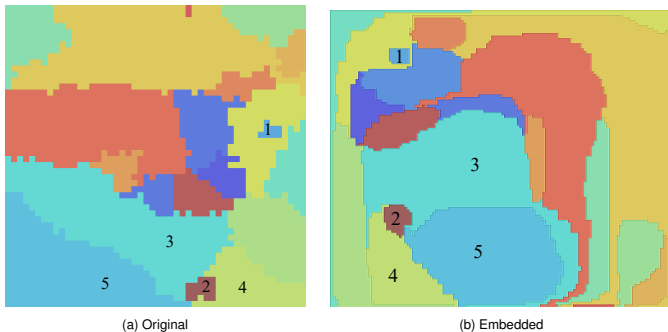


Fig. 7: A 2D embedding (b) of a 2D segmentation (a) allows for a visual comparison of input and output of our algorithm. We observe that the topology is maintained (including the light blue segment (label 1) fully surrounded by the yellow one) and that area sizes and shared boundary lengths are approximately the same. Shapes and locations are not preserved, but this was not a desired goal of our approach.

where each segment has a joint boundary with three other segments and the outer boundary of the domain, see Fig. 1.

To evaluate our approach with respect to different influencing factors, we define a set of quality metrics that we want to investigate:

Number of edge crossings: We investigate the absolute number of edge crossings, as fewer edge crossings mean a less complex embedding.

Mean area deviation: We measure how well our area optimization works by computing the mean area deviation for all segments $\bar{d}_A = \frac{1}{|S|} \sum_{s \in S} |d_{A,s}|$, where $d_{A,s}$ is computed as in Eq. (1) and $|S|$ denotes the cardinality of S .

Mean boundary length deviation: We measure how well our boundary length optimization works by computing the mean boundary length deviation for all adjacent segments s_i and s_j by $\bar{d}_L = \frac{1}{|E|} \sum_{(s_i, s_j) \in S \times S} |d_{L,(s_i, s_j)}|$, where $d_{L,(s_i, s_j)}$ is the boundary length deviation as in Eq. (2) and $|E|$ denotes the number of edges in the graph representing the segmentation.

Note that the topology preservation is not explicitly evaluated because the topology is perfectly preserved based on the construction of the algorithm.

6.1 Visual Inspection and Numerical Evaluations

We first apply our algorithm to the 2D version of dataset D_1 with 20 segments. Using a 2D input is obviously not a meaningful application scenario, but it allows for a visual inspection and comparison of input and output, which are shown in Fig. 7. The output is obtained after 5,000 iterations with a damping factor of 7 and a security factor of 11. We observe that topological structures are preserved. Moreover, area sizes in both images are similar, which is confirmed by a mean area deviation of $\bar{d}_A = 0.016\%$. Also, boundary lengths are similar, which is confirmed by a mean boundary length deviation of $\bar{d}_L = 0.96\%$. For example, when investigating the brown segment in the lower left part of the embedding (label 2), most of its boundary is shared with the lighter blue (label 3) and the green segment (label 4), while only a small shared boundary appears with the darker blue color (label 5), which agrees well with the original segmentation. Location of segments and shapes are not preserved, but this was not a desired goal of our approach.

Next, we apply our approach to dataset D_2 using the same parameters as above, see Fig. 1. Here, we obtain a mean area deviation of $\bar{d}_A = 0.18\%$ and a mean boundary length deviation of $\bar{d}_L = 4.49\%$. In general, the deviations are significantly higher than for the 2D case. However, this was to be expected, as in this case, a dimensionality reduction is involved and, additionally, the graph cannot be embedded in 2D as a planar graph. Therefore, we obtain four edge crossings and some separating regions of segments without a shared boundary.

The quality measures for all datasets discussed in this paper are presented in Table 1 of the supplementary material. Here, we can observe that the results are rather dataset-dependent.

6.2 Influence of Parameters

Two main parameters for our algorithm are the choice of the damping factor and the security factor. To identify their influence on the outcome, we study damping factors between 1 and 9 and security factors between 9 and 12. We use dataset D_1 with 5, 10, and 15 segments in 2D and 3D settings. For all cases, we set the number of iterations to the number of pixels as the image resolution scales with the segmentation’s complexity to visualize. The results are shown in Fig. 8. We can observe that the accuracy slightly increases for damping factors > 3 . This observation can be explained by the fact that too strong damping leads to early termination of the algorithm, i.e., the result is not fully optimized. Therefore, we recommend a larger damping factor, even though it depends on the underlying dataset. When observing the results for the different security factors, we can observe that 10 and 11 lead to better results in mean area deviation. The variation in mean boundary length deviation is less pronounced. Here, the security factor of 12 outperforms the others, which is expected as it allows for more flexibility in forming boundaries. However, as it performs worse in area deviation, we still recommend security factors of 10 or 11 based on the numerical results.

The visual differences between the security factors are shown in Fig. 9. The results for security factors 9 to 11 are quite similar and the visual differences are negligible. However, for a security factor of 12, we see that the boundaries between the segments are significantly more fine-grained. As this adds additional complexity to the visualization and, as seen in Fig. 8, does not improve the numerical results in general, we recommend choosing a smaller security factor. For the following results, we use a security factor of 11 as a default.

6.3 Optimization Criteria

Next, we want to evaluate the impact of area and boundary length preservation. Only optimizing boundary length does not lead to desirable results, as the boundary length optimization does not allow for a spreading towards the background voxels of the initial graph embedding. Hence, we compare using area optimization only to the combined optimization of area and boundary length with respect to the quality metrics defined above. We choose dataset D_1 with 5 to 15 segments in a 3D setting and a security factor of 11. Note that the area deviations are significantly smaller than the boundary length deviations leading to a smaller mean error when considering only areas. To avoid early stopping in this case, we increase the damping factor to 100 in this evaluation (while choosing 7 in the other cases). The results are shown in Fig. 10. We observe that adding boundary length optimization leads to a larger area deviation, but to a significant decrease in boundary deviation as expected. However, the difference for the dataset variants investigated here is relatively small. At the same time, the computations take much longer if the boundary deviation is included. Thus, for larger graphs with more complex embeddings when slightly larger boundary deviations may be tolerable, we recommend only optimizing for areas.

6.4 Scalability

The computational scalability of our approach depends on different factors. The first step is the graph computation. Computing the vertices including their weights scales linearly with the number of cells N in the multi-dimensional domain, leading to a complexity of $\mathcal{O}(N)$. For determining shared boundaries, we obtain a complexity of $\mathcal{O}(Nn)$ where n is the number of dimensions, because, for each voxel, both neighbors in each dimension need to be checked. Note that the number of cells increases exponentially with the number of dimensions (when assuming the same sampling). The subsequent steps for creating the layout only depend on the weighted graph. Thus, the remainder of the algorithm is insensitive to the resolution and dimensionality of the input data, but instead scales with its internal complexity, which is characterized by the number of vertices and edges. A more complex graph is not only more expensive to draw, but also requires a larger number of cells for the cellular automaton, which leads to larger computation times in the cellular automaton. Additionally, edge crossings increase the computational cost as additional optimization steps are executed. This can be confirmed experimentally. We observe that the computational

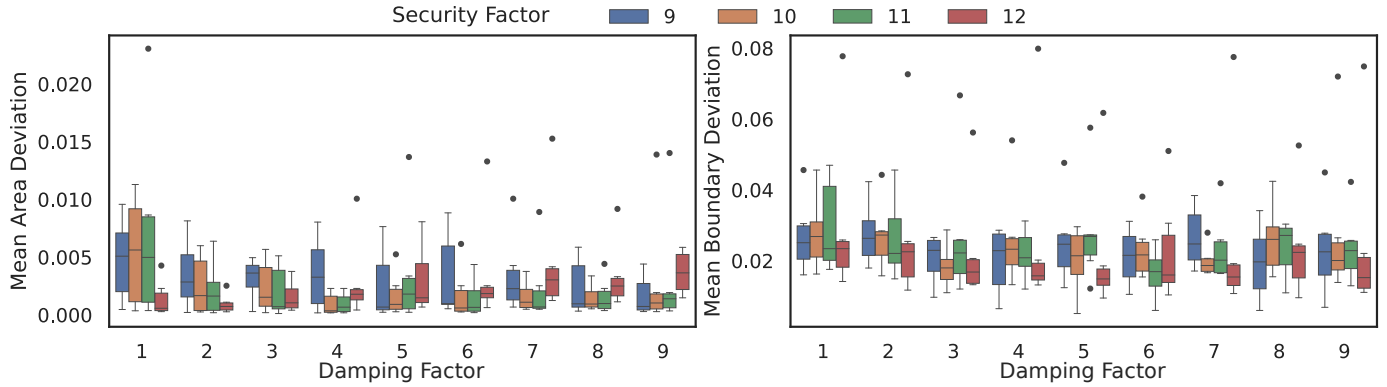


Fig. 8: Mean area deviation and mean boundary length deviation are evaluated depending on the damping factor as well as the security factor.

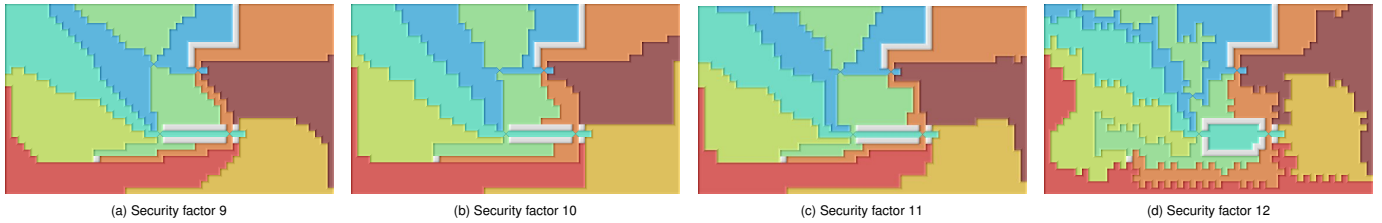


Fig. 9: Cellular automaton with different security factors applied to the same initial state: The result for a security factor of 12 differs significantly from the results with lower values, introducing unnecessary visual complexity.

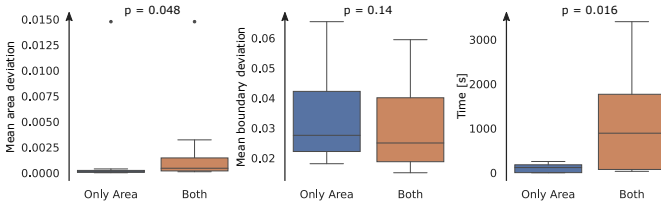


Fig. 10: The comparison of different evaluation criteria shows the trade-offs between only optimizing for area and optimizing for both area and boundary length. The p-values are computed using a paired t-test.

costs increase significantly with the number of segments (see Fig. A2 in the supplementary material). The increasing number of cells most likely also causes the decrease in the mean deviation.

Table 2 of the supplementary material provides an overview of the run times for the different steps of the algorithm of different real-world and artificial datasets. Which step is the computationally most expensive one strongly depends on the dataset. The computation time for the graph is mainly determined by the number of voxels in the input space, which confirms the theoretical considerations. The complexity of the graph, for which the number of edge crossings and the number of segments provide an indicator, mainly determines the computation time for the embedding. The number of cells of the cellular automaton is the main influencing factor for the computational cost of running the automaton. However, early termination of the automaton significantly reduces the computation times for this step. The respective quantitative analysis is detailed in Section C of the supplementary material.

7 APPLICATION TO REAL-WORLD SCENARIOS

We demonstrate the applicability of our approach to real-world problems by applying it to the visualization of 3D image segmentations as well as to 4D and 5D parameter space partitionings of simulation ensembles. We also discuss our approach in comparison to visualizing the segmentation as a graph drawing. A second 3D segmentation use case can be found in the supplementary material.

7.1 3D Image Segmentations

As a first use case, we apply our algorithm to a 3D volume segmentation used as an input for a radiofrequency ablation simulation. The 3D domain covers the human body around the liver and contains different organ tags as well as tags for a tumor and the needle that is used for radiofrequency ablation. The dataset has a resolution of $92 \times 92 \times 92$ and contains 11 different segments. As the regions with unique organ tags are not all connected, we set a unique ID for each connected component, which leads to 59 segments. A volume visualization showing only a subset of the tissue types is shown in Fig. 11a. While a 3D segmentation could also be visualized in a volume rendering, showing a growing number of segments quickly leads to occlusion. Without interaction, rendering 11 segments at the same time is already challenging. However, we do not consider our visualization to be a replacement for common volume visualization techniques but instead propose to use the segmentation embedding to obtain an overview of all segments and investigate their sizes and topology. For example, it could be used in an interactive setting, where segments of interest can be selected in the embedding and shown in a 3D surface or volume visualization.

The resulting embedding is shown in Fig. 11b. Here, we apply a color coding based on the original segments. When observing the embedding, we can directly see that the majority of the given domain is covered by the two segments that are labeled as liver and other tissue (tissue without specific organ tags). In the center, we see a group of segments that represent the vessels. The vessels have joint boundaries. While the hepatic artery (HA) and the portal vein (PV) are connected to the unlabeled tissue, the hepatic vein (HV) contains no connection to this kind of tissue. We also spot very small segments belonging to the hepatic artery (HA) that are isolated inside of a region with no organ tags. As artery parts that are neither connected to the border of the domain nor to other vessels are not plausible, this indicates either a mistake in the segmentation or an undersampling artifact. The needle used for ablating the tumor is surrounded only by liver tissue, unlabeled tissue, and tumor. Therefore, we can deduce that the placement of the needle does not damage any of the other tissue types.

A straightforward alternative design choice for our visualization fulfilling all optimization criteria would be a representation of the segmentation properties in a node-link diagram as shown in Fig. 11c.

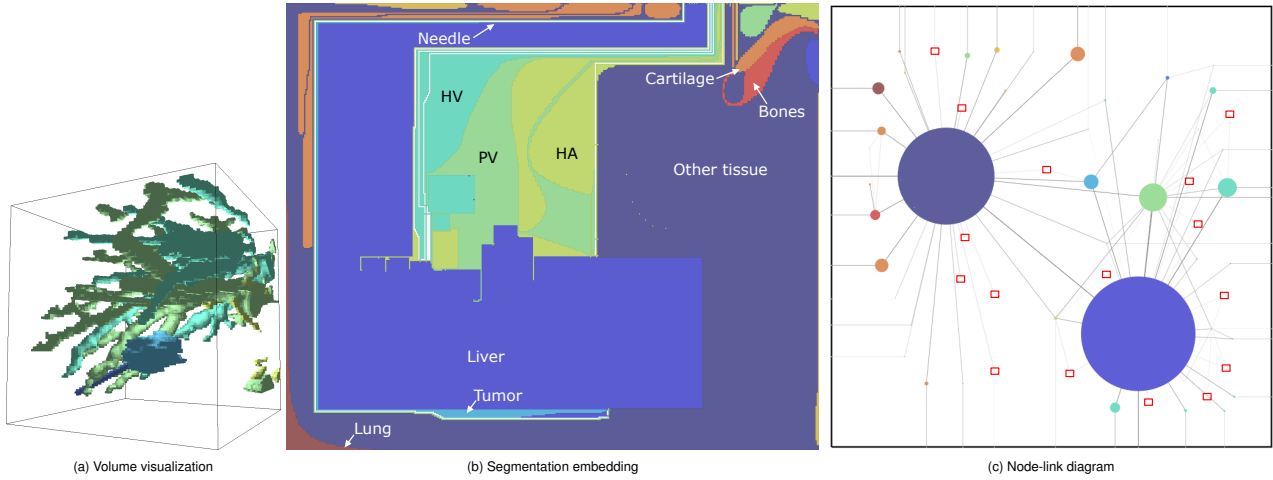


Fig. 11: Different visualizations of 3D medical image segmentation data. The volume visualization (a) only shows selected segments (here vessel, needle, and tumor). The 2D embedding (b) provides an overview of sizes and topological information of all segments. The given organ tags are used to label the embedding’s segments. (c) Encoding the same information in a node-link diagram using the same color map as in (b). The red boxes mark very small nodes.

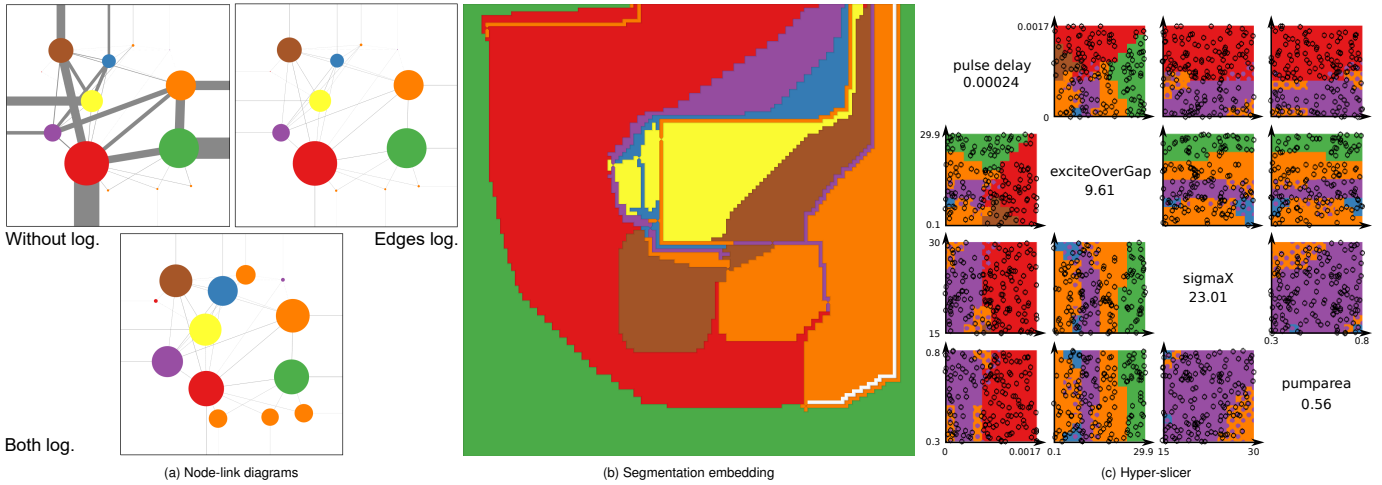


Fig. 12: Different encodings of the node-link diagram (a), 2D embedding (b) and hyper-slicer (c) of a 4D parameter space partitioning of a semiconductor simulation ensemble. The embedding allows the user to investigate the structures globally, while the hyper-slicer only provides a local view.

Details on how we create the node-link diagram can be found in the supplementary material, Section D. The topological information can be encoded by the edges of the nodes, the sizes of the segments by the nodes’s sizes and the boundary lengths by the edge width. Therefore, we explicitly compare such a node-link diagram to our segmentation embedding. As the boundary sizes vary significantly, we had to choose a logarithmic scale for the edge widths to keep most of the edges in a visible range. A segment’s joint boundary with the border of the domain is encoded by an edge connecting to the black bounding box.

Node-link diagrams have been shown to not scale well with the number of nodes and an increasing node-link density [32]. We also observe this issue when comparing our embedding to the node-link diagram. In general, the dense pixel visualization of our embedding makes better use of the available screen space. This becomes especially clear when encoding the segment sizes and edges. Some of the nodes representing smaller segments are barely visible such as the ones marked in Fig. 11c. Additionally, the logarithmically scaled edge widths are hard to interpret with respect to what portion of a segment’s boundary is shared with another. This problem is also shown in Fig. 12a, where we compare the different options for a less complex node-link diagram. While the logarithmic scaling improves the visibil-

ity of edges and nodes such as the small red one, sizes are much harder to interpret and compare. For example, the size difference between the yellow and the large red node is hard to estimate (Tasks T2 and T3). In this example, we also observe occlusion of the yellow node and some edges, which complicates the understanding of the topological structure (T1). As the segmentation embedding resembles segmentation visualizations, segments completely enclosed by other segments in the high-dimensional domain are visualized as completely enclosed segments in the embedding, which is quite intuitive. In the graph drawing, instead, such segments are visualized as nodes with a single edge, where its interpretation requires an additional cognitive effort.

7.2 Multi-dimensional Parameter Space Partitioning

In this section, we analyze multi-dimensional parameter space partitionings of simulation ensembles. We first consider the analysis of a blood flow simulation ensemble with a 5D parameter space. This simulation ensemble was created to study the blood flow through a brain aneurysm and find biomarkers for certain diseases [29]. The simulation is driven by five different input parameters: a characteristic length, a characteristic velocity, the viscosity of the fluid, its density, and a parameter defining the boundary conditions. The simulation

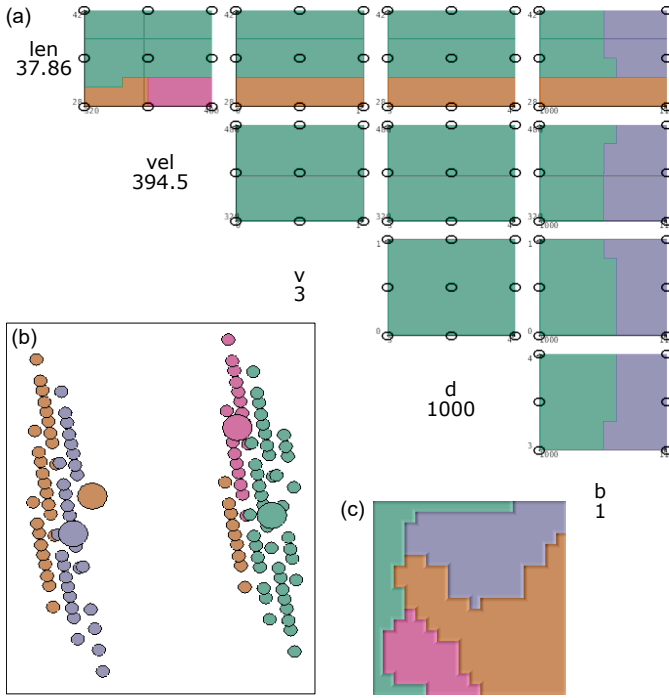


Fig. 13: Parameter space visualizations of a 5D parameter space partitioning of a blood flow simulation ensemble, where the partitioning is based on a clustering of the ensemble members. Our segmentation embedding (c) provides a global overview that is not provided by the hyperslicer (a), yet the hyperslicer can provide some local details. The sample embedding (b) does not capture the topology of the multidimensional segmentation.

output can be used to partition the parameter space into regions leading to similar behavior. In this paper, we work on the parameter space partitioning as determined by Evers et al. [17]. They presented an approach for partitioning the parameter space and analyzing it using a hyper-slicer as shown in Fig. 13a. Their analysis approach also contains a parameter space overview visualization for which they apply multidimensional scaling to the parameter space samples which are the parameter settings for the simulation runs. Then, they color code the points in low-dimensional space based on the cluster they belong to and additionally insert a larger point representing the cluster for navigation (see Fig. 13b). However, this visualization does not support the investigation of segment sizes and topological properties. For the parameter space sampling, we choose a resolution of 10 in each dimension. Our 2D embedding of the parameter space partitioning (Fig. 13c) together with the parameter space visualizations proposed by Evers and Linsen [17] is shown in Fig. 13. The 2D embedding obtained a mean area deviation of 0.62% and a mean boundary deviation of 0.38%, which indicates a very accurate representation. Our 2D embedding reveals that the green segment is connected to all other segments. This can be confirmed when viewing those 2D slices where we can see the shared boundaries. However, the 2D slices only show a subset of information. For example, we cannot immediately see that the largest part of the parameter space belongs to the orange segment or that the purple and the pink segments do not share a joint boundary. The parameter sample embedding shown in Fig. 13b also does not allow us to obtain this information. For this use case, it does not become clear from the parameter sample embedding that the green and the purple segments share a joint boundary in parameter space. Thus, the topology information is not pertained. Additionally, the sizes could only be represented by the number of points belonging to each segment. However, if several points are placed close to each other in the parameter sample embedding, the size information is difficult to obtain. Hence, we conclude that our 2D segmentation embedding allows for a better global overview of the

entire structure of the multi-dimensional partitioning.

Next, we analyze the 4D parameter space partitioning for a semiconductor simulation ensemble [17]. The simulation ensemble was created to study transport properties in a quantum wire. As in the previous use case, the partitioning was created by clustering the ensemble members and transferring this clustering to the 4-dimensional parameter space of the simulation. We choose the same clustering as Evers and Linsen [17], which results in 7 clusters. The parameter space is sampled with a resolution of 10 in each dimension. Our 2D embedding of the parameter space partitioning is shown in Fig. 12b, where we color the segments according to the clusters. The embedding shows that the parameter space partitions of the clusters are not necessarily connected. For example, the red and purple segments are split into two disconnected regions. The same partitioning is shown in the hyper-slicer in Fig. 12c. Obtaining the same structural information would not only require a significant amount of interaction with the hyper-slicer but also a high amount of mental effort.

8 DISCUSSION AND CONCLUSION

We presented an algorithm for embedding multi-dimensional partitionings into 2D, where we optimize the area and boundary sizes to represent those of the multi-dimensional space while preserving the partitioning's topology. The 2D embedding was computed by first creating a graph representation of the partitioning, whose graph drawing was used as an initial configuration for cellular automaton optimization. Our rendering of the embedded segments allows for a large flexibility in the choice of color codings.

Our approach maintains the desired criteria well and allows us to represent several structures that could occur in segmentations including segments that are completely surrounded by other segments and connections to the boundary. Besides an evaluation on synthetic datasets, an application to 3D segmentations as well as to multi-dimensional parameter space partitionings shows its utility on real-world data. In particular, we demonstrated that our approach provides an overview of the entire segmentation structure, which other approaches, such as a hyper-slicer or dimensionality reductions, cannot provide. We also compared the segmentation embedding to representations based on a node-link diagram and showed that our approach presents certain features of segmentations, such as enclosed segments, more clearly and uses the available space more efficiently. Interpreting our visualization requires no knowledge about graphs as the underlying graph representation is not visible in the final visualization. Additionally, we showed scaling problems of the node-link diagram. However, a deeper investigation of the strengths and weaknesses of both approaches would require a user study, which we plan for future work.

Our embedding provides an overview of the segmentation structure but loses information about the positions and shapes of the segments. However, linking our 2D embedding to coordinated views of volume renderings of 3D segments or multi-dimensional data visualizations (like the hyper-slicer for parameter space partitionings or parallel coordinates) allows for accessing this information for selected segments, where the selection can easily be performed in our 2D embedding. A limitation of the current implementation are relatively high computation times. However, the implementation is not yet optimized and parallelized, which could lead to a significant speed-up. On the other hand, the computation of the 2D embedding can be a pre-processing step for an interactive analysis and only needs to be computed once. If only small changes in the sizes of the segments and boundaries occur, it would also be possible to update the embedding instead of requiring a complete recomputation. Thin segments might occur in order to preserve the topology and in the attempt to optimize the boundary, which may hamper area estimation. We plan to address this limitation by adapting the cellular automaton in future work. Incorporating our 2D embedding into more powerful interactive analyses such as analyzing the simulation outcomes of a parameter-space segment is another potential future research direction, but was clearly beyond the scope of this paper. Moreover, the quality in preserving the boundary length is lower than that of the area preservation, which provides room for further improvements.

SUPPLEMENTAL MATERIALS

The implementation of the algorithm is available at <https://github.com/marinaevers/segmentation-projection>. The supplemental material published together with this paper contains an appendix with further details for reproducibility and a video that shows the iterative optimization of the algorithm.

ACKNOWLEDGMENTS

This work was funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant 260446826 (LI 1530/21-2).

REFERENCES

- [1] M. J. Alam, T. Biedl, S. Felsner, M. Kaufmann, S. G. Kobourov, and T. Ueckerdt. Computing cartograms with optimal complexity. In *Proceedings of the twenty-eighth annual symposium on Computational geometry*, pp. 21–30, 2012. doi: 10.1145/2261250.2261254 2
- [2] B. Alper, N. Riche, G. Ramos, and M. Czerwinski. Design study of linesets, a novel set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 17(12):2259–2267, 2011. doi: 10.1109/TVCG.2011.186 2
- [3] C. Batini, E. Nardelli, and R. Tamassia. A layout algorithm for data flow diagrams. *IEEE Transactions on Software Engineering*, SE-12(4):538–546, 1986. 2
- [4] S. Bergner, M. Sedlmair, T. Moller, S. N. Abdolyousefi, and A. Saad. Paraglide: Interactive parameter space partitioning for computer simulations. *IEEE Transactions on Visualization and Computer Graphics*, 19(9):1499–1512, 2013. doi: 10.1109/tvcg.2013.61 2
- [5] A. Bock, A. Pembroke, M. L. Mays, L. Rastaetter, T. Ropinski, and A. Ynnerman. Visual verification of space weather ensemble simulations. In *2015 IEEE Scientific Visualization Conference (SciVis)*, pp. 17–24. IEEE, 2015. doi: 10.1109/SciVis.2015.7429487 2
- [6] J. M. Boyer and W. J. Myrvold. Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications*, 5:241–273, 2006. doi: 10.7155/jgaa.00091 12
- [7] S. Bruckner and T. Möller. Result-driven exploration of simulation parameter spaces for visual effects design. *IEEE Transactions on Visualization and Computer Graphics*, 16(6):1468–1476, 2010. doi: 10.1109/tvcg.2010.190 2
- [8] P. Chalermsook and A. Schmid. Finding triangles for maximum planar subgraphs. In *International Workshop on Algorithms and Computation*, pp. 373–384. Springer, 2017. doi: 10.1007/978-3-319-53925-6_29 12
- [9] M. Chimani, C. Gutwenger, M. Jünger, G. W. Klau, K. Klein, and P. Mutzel. The open graph drawing framework (OGDF). *Handbook of graph drawing and visualization*, 2011:543–569, 2013. 12
- [10] M. Chimani, C. Gutwenger, M. Jünger, K. Klein, P. Mutzel, and M. Schulz. The open graph drawing framework. In *15th International Symposium on Graph Drawing*, pp. 23–26, 2007. 12
- [11] M. Chimani, M. Ilsen, and T. Wiedera. Star-struck by fixed embeddings: Modern crossing number heuristics. In *International Symposium on Graph Drawing and Network Visualization*, pp. 41–56. Springer, 2021. doi: 10.1007/978-3-030-92931-2_3 12
- [12] C. Collins, G. Penn, and S. Carpendale. Bubble Sets: Revealing set relations with isocontours over existing visualizations. *IEEE Transactions on Visualization and Computer Graphics*, 15(6):1009–1016, 2009. doi: 10.1109/TVCG.2009.122 2
- [13] P. Crossno. Challenges in visual analysis of ensembles. *IEEE Computer Graphics and Applications*, 38(2):122–131, 2018. doi: 10.1109/mcg.2018.021951640 2
- [14] W. Didimo, G. Liotta, and F. Montecchiani. A survey on graph drawing beyond planarity. *ACM Computing Surveys (CSUR)*, 52(1):1–37, 2019. doi: 10.1145/3301281 2
- [15] D. Dorling. Area cartograms: their use and creation. In *Concepts and techniques in modern geography series*. Environmental Publications, University of East Anglia, 1996. doi: 10.1002/9780470979587.ch33 3, 4
- [16] A. Efrat, Y. Hu, S. G. Kobourov, and S. Pupyrev. Mapsets: Visualizing embedded and clustered graphs. In *International Symposium on Graph Drawing*, pp. 452–463. Springer, 2014. doi: 10.1007/978-3-662-45803-7_38 2
- [17] M. Evers and L. Linsen. Multi-dimensional parameter-space partitioning of spatio-temporal simulation ensembles. *Computers & Graphics*, 104:140–151, 2022. doi: 10.1016/j.cag.2022.04.005 2, 9
- [18] O. Fernandes, S. Frey, G. Reina, and T. Ertl. Visual representation of region transitions in multi-dimensional parameter spaces. *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, 2019. doi: 10.2312/STAG.20191367 2
- [19] C. Fiedler, W. Scheibel, D. Limberger, M. Trapp, and J. Döllner. Survey on user studies on the effectiveness of treemaps. In *Proceedings of the 13th international symposium on visual information communication and interaction*, pp. 1–10, 2020. doi: 10.1145/3430036.3430054 2
- [20] M. T. Fischer, A. Frings, D. A. Keim, and D. Seebacher. Towards a survey on static and dynamic hypergraph visualizations. In *2021 IEEE Visualization Conference (VIS)*, pp. 81–85. IEEE, 2021. doi: 10.1109/VIS49827.2021.9623305 2
- [21] T. M. Fruchterman and E. M. Reingold. Graph drawing by force-directed placement. *Software: Practice and experience*, 21(11):1129–1164, 1991. doi: 10.1002/spe.4380211102 2, 13
- [22] E. R. Gansner, Y. Hu, and S. Kobourov. Gmap: Visualizing graphs and clusters as maps. In *2010 IEEE Pacific Visualization Symposium (PacificVis)*, pp. 201–208. IEEE, 2010. doi: 10.1109/PACIFICVIS.2010.5429590 2
- [23] M. R. Garey and D. S. Johnson. Crossing number is NP-complete. *SIAM Journal on Algebraic Discrete Methods*, 4(3):312–316, 1983. doi: 10.1137/0604033 3
- [24] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 6(1):24–43, 2000. doi: 10.1109/2945.841119 2
- [25] Y. Hu, E. R. Gansner, and S. Kobourov. Visualizing graphs and clusters as maps. *IEEE Computer Graphics and Applications*, 30(6):54–66, 2010. doi: 10.1109/MCG.2010.101 2
- [26] R. Jayakumar, K. Thulasiraman, and M. Swamy. $O(n^2)$ algorithms for graph planarization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 8(3):257–267, 1989. doi: 10.1109/43.21845 12
- [27] T. Kamada, S. Kawai, et al. An algorithm for drawing general undirected graphs. *Information processing letters*, 31(1):7–15, 1989. doi: 10.1016/0020-0190(89)90102-6 2
- [28] S. G. Kobourov, S. Pupyrev, and P. Simonetto. Visualizing graphs as maps with contiguous regions. In *EuroVis (Short Papers)*, 2014. doi: 10.2312/eurovisshort.20141153 2
- [29] S. Leistikow, A. Nahardani, V. Hoerr, and L. Linsen. Interactive visual similarity analysis of measured and simulated multi-field tubular flow ensembles. In *Eurographics Workshop on Visual Computing for Biology and Medicine*, pp. 139–150. The Eurographics Association, 2020. doi: 10.2312/vcbm.20201180 8
- [30] F. McGee, M. Ghoniem, G. Melançon, B. Otjacques, and B. Pinaud. The state of the art in multilayer network visualization. *Computer Graphics Forum*, 38(6):125–149, 2019. doi: 10.1111/cgf.13610 2
- [31] W. Meulemans, N. H. Riche, B. Speckmann, B. Alper, and T. Dwyer. Kelpfusion: A hybrid set visualization technique. *IEEE Transactions on Visualization and Computer Graphics*, 19(11):1846–1858, 2013. doi: 10.1109/TVCG.2013.76 2
- [32] T. Munzner. *Visualization analysis and design*. CRC Press, 2014. 8
- [33] S. Nusrat and S. Kobourov. The state of the art in cartograms. *Computer Graphics Forum*, 35(3):619–642, 2016. doi: 10.1111/cgf.12932 2, 3
- [34] H. Obermaier, K. Bensema, and K. I. Joy. Visual trends analysis in time-varying ensembles. *IEEE Transactions on Visualization and Computer Graphics*, 22(10):2331–2342, 2016. doi: 10.1109/tvcg.2015.2507592 2
- [35] D. Orban, D. F. Keefe, A. Biswas, J. Ahrens, and D. Rogers. Drag and track: A direct manipulation interface for contextualizing data instances within a continuous parameter space. *IEEE Transactions on Visualization and Computer Graphics*, 25(1):256–266, 2019. doi: 10.1109/tvcg.2018.2865051 2
- [36] B. Preim and D. Bartz. *Visualization in Medicine: Theory, Algorithms, and Applications*. Morgan Kaufmann Publishers Inc., 2007. 2
- [37] J. Ren, J. Schneider, M. Ovsjanikov, and P. Wonka. Joint graph layouts for visualizing collections of segmented meshes. *IEEE Transactions on Visualization and Computer Graphics*, 24(9):2546–2558, 2018. doi: 10.1109/TVCG.2017.2751473 2
- [38] H.-J. Schulz and H. Schumann. Visualizing graphs—a generalized view. In *Tenth International Conference on Information Visualisation (IV'06)*, pp. 166–173. IEEE, 2006. doi: 10.1109/IV.2006.130 2
- [39] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Möller. Visual parameter space analysis: A conceptual framework. *IEEE Transactions*

- on *Visualization and Computer Graphics*, 20(12):2161–2170, 2014. doi: [10.1109/tvcg.2014.2346321](https://doi.org/10.1109/tvcg.2014.2346321) 2
- [40] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Transactions on Graphics (TOG)*, 11(1):92–99, 1992. doi: [10.1145/102377.115768](https://doi.org/10.1145/102377.115768) 2
- [41] B. Spence, L. Tweedie, H. Dawkes, and H. Su. Visualization for functional design. In *Proceedings of the 1995 IEEE Symposium on Information Visualization*, INFOVIS '95, pp. 4–10. IEEE Computer Society, 1995. doi: [10.1109/infvis.1995.528680](https://doi.org/10.1109/infvis.1995.528680) 2
- [42] K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical system structures. *IEEE Transactions on Systems, Man, and Cybernetics*, 11(2):109–125, 1981. doi: [10.1109/TSMC.1981.4308636](https://doi.org/10.1109/TSMC.1981.4308636) 2
- [43] R. Tamassia. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing*, 16(3):421–444, 1987. doi: [10.1137/0216030](https://doi.org/10.1137/0216030) 12
- [44] R. Tamassia. *Handbook of graph drawing and visualization*. CRC press, 2013. 2, 3
- [45] R. Tamassia, G. Di Battista, and C. Batini. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics*, 18(1):61–79, 1988. doi: [10.1109/21.87055](https://doi.org/10.1109/21.87055) 12
- [46] A. C. Telea. *Data visualization: principles and practice*. CRC Press, 2014. 5
- [47] S. T. Teoh. A study on multiple views for tree visualization. In *Visualization and Data Analysis 2007*, vol. 6495, pp. 99–110. SPIE, 2007. doi: [10.1117/12.703076](https://doi.org/10.1117/12.703076) 2
- [48] J. J. van Wijk and H. van de Wetering. Cushion treemaps: Visualization of hierarchical information. In *Proceedings 1999 IEEE Symposium on Information Visualization (InfoVis '99)*, pp. 73–78. IEEE, 1999. doi: [10.1109/INFVIS.1999.801860](https://doi.org/10.1109/INFVIS.1999.801860) 5
- [49] C. Vehlou, F. Beck, and D. Weiskopf. Visualizing group structures in graphs: A survey. *Computer Graphics Forum*, 36(6):201–225, 2017. doi: [10.1111/cgf.12872](https://doi.org/10.1111/cgf.12872) 2
- [50] T. von Landesberger, A. Kuijper, T. Schreck, J. Kohlhammer, J. J. van Wijk, J.-D. Fekete, and D. W. Fellner. Visual analysis of large graphs: State-of-the-art and future research challenges. *Computer Graphics Forum*, 30(6):1719–1749, 2011. doi: [10.1111/j.1467-8659.2011.01898.x](https://doi.org/10.1111/j.1467-8659.2011.01898.x) 2
- [51] J. Wang, S. Hazarika, C. Li, and H.-W. Shen. Visualization and visual analysis of ensemble data: A survey. *IEEE Transactions on Visualization and Computer Graphics*, 25(9):2853–2872, 2019. doi: [10.1109/tvcg.2018.2853721](https://doi.org/10.1109/tvcg.2018.2853721) 2
- [52] J. Wang, X. Liu, H.-W. Shen, and G. Lin. Multi-resolution climate ensemble parameter analysis with nested parallel coordinates plots. *IEEE Transactions on Visualization and Computer Graphics*, 23(1):81–90, 2017. doi: [10.1109/tvcg.2016.2598830](https://doi.org/10.1109/tvcg.2016.2598830) 2
- [53] H.-Y. Wu, M. Nöllenburg, and I. Viola. Multi-level area balancing of clustered graphs. *IEEE Transactions on Visualization and Computer Graphics*, 28(7):2682–2696, 2022. doi: [10.1109/TVCG.2020.3038154](https://doi.org/10.1109/TVCG.2020.3038154) 2
- [54] M. Yannakakis. Node-and edge-deletion NP-complete problems. In *Proceedings of the tenth annual ACM symposium on Theory of computing*, pp. 253–264, 1978. doi: [10.1145/800133.804355](https://doi.org/10.1145/800133.804355) 12
- [55] K.-H. Yeap and M. Sarrafzadeh. Floor-planning by graph dualization: 2-concave rectilinear modules. *SIAM Journal on Computing*, 22(3):500–526, 1993. doi: [10.1137/0222035](https://doi.org/10.1137/0222035) 2
- [56] D. A. Zaitsev. A generalized neighborhood for cellular automata. *Theoretical Computer Science*, 666:21–35, 2017. doi: [10.1016/j.tcs.2016.11.002](https://doi.org/10.1016/j.tcs.2016.11.002) 3