ADJACENCY-PRESERVING SPATIAL TREEMAPS*

Kevin Buchin,† *David Eppstein,*‡ *Maarten Löffler,*§ *Martin Nöllenburg,*¶ *and Rodrigo I. Silveira*‖

Abstract. Rectangular layouts, subdivisions of an outer rectangle into smaller rectangles, have many applications in visualizing spatial information, for instance in rectangular cartograms in which the rectangles represent geographic or political regions. A *spatial treemap* is a rectangular layout with a hierarchical structure: the outer rectangle is subdivided into rectangles that are in turn subdivided into smaller rectangles. We describe algorithms for transforming a rectangular layout that does not have this hierarchical structure, together with a clustering of the rectangles of the layout, into a spatial treemap that respects the clustering and also respects to the extent possible the adjacencies of the input layout.

1 Introduction

Spatial treemaps are an effective technique to visualize two-dimensional hierarchical information. They display hierarchical data by using nested rectangles in a space-filling layout. Each rectangle represents a geometric or geographic region, which in turn can be subdivided recursively into smaller regions. On lower levels of the recursion, rectangles can also be subdivided based on non-spatial attributes. Typically, at the lowest level some attribute of interest of the region is summarized by using properties like area or color. Treemaps were originally proposed to represent one-dimensional information in two dimensions [20] and remain a popular visualization technique for hierarchical data in general [21]. Yet, they are particularly well suited to represent spatial—two-dimensional—data because the containment metaphor of the nested rectangles has a natural geographic meaning, and two-dimensional data makes an efficient use of space. This has led to the concept of spatially ordered treemaps [26] which remain an active research topic in geovisualization [11]. In fact, treemaps with spatial constraints on the positions of rectangles are also interesting for non-geographic data (e.g., for visualizing human anatomy [6]) and there are treemap algorithms that take certain positional constraints into account [15]. Another treemap application, where spatial constraints are important is the visualization of dynamic data while maintaining spatial stability of the treemap [24].

Spatial treemaps are closely related to rectangular cartograms [19]: distorted maps where each region is represented by a rectangle whose area corresponds to a numerical

*A preliminary version of this paper appeared at the Algorithms and Data Structures Symposium (WADS) 2011 [4].

†*Dept. of Mathematics and Computer Science, TU Eindhoven, the Netherlands*, k.a.buchin@tue.nl
‡*Dept. of Computer Science, University of California, Irvine, USA*, eppstein@ics.uci.edu
§*Dept. of Computing and Information Sciences, Utrecht University, the Netherlands*, m.loffler@uu.nl
¶*Algorithms and Complexity Group, TU Wien, Vienna, Austria*, noellenburg@ac.tuwien.ac.at
‖*Dept. de Matemàtiques, Universitat Politècnica de Catalunya, Spain*, rodrigo.silveira@upc.edu

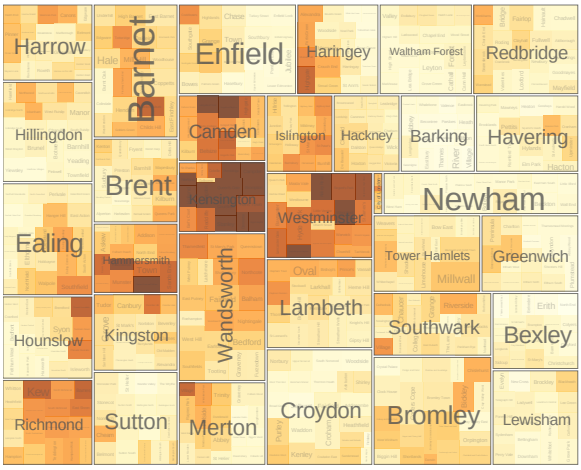
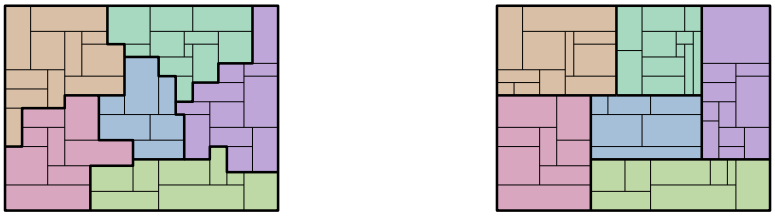


Figure 1: A 2-level spatial treemap from [22]; used with permission.

attribute such as population. Rectangular cartograms can be seen as spatial treemaps with only one level; multi-level spatial treemaps in which every rectangle corresponds to a region are also known as *rectangular hierarchical cartograms* [22, 23]. Spatial treemaps and rectangular cartograms have in common that it is essential to preserve the recognizability of the regions shown [25]. Most previous work on spatial treemaps reflects this by focusing on the preservation of distances between the rectangular regions and their geographic counterparts (that is, they minimize the displacement of the regions). However, often small displacement does not imply recognizability (swapping the position of two small neighboring countries can result in small displacement, but a big loss of recognizability). In the case of cartograms, most emphasis has been put on preserving adjacencies between the geographic regions. It has also been shown that while preserving the topology it is possible to keep the displacement error small [5, 25]. Another special type of spatial treemaps are grid maps [9], which arrange equally-sized rectangles in a regular grid-like fashion while seeking to optimize criteria such as correct adjacencies, displacement, and directional relations.

In this paper we are interested in constructing high-quality spatial treemaps by prioritizing the preservation of topology, following a principle already used for rectangular cartograms. Previous work on treemaps has recognized that preserving neighborhood relationships and relative positions between the regions were important criteria [7, 11, 12, 15, 17, 26], but we are not aware of treemap algorithms that put the emphasis on preserving topology.

The importance of preserving adjacencies in spatial treemaps can be appreciated by viewing a concrete example. Figure 1, from [22], shows a spatial treemap of property transactions in London between 2000 and 2008, with two levels formed by the boroughs and wards of London and colors representing average prices. To see whether housing prices of neighboring wards are correlated, it is important to preserve adjacencies: otherwise it is easy to draw incorrect conclusions, like seeing clusters that do not actually exist, or missing existing ones.



(a) (b)

Figure 2: (a) An example input: a two-level subdivision, where the regions at the top level in the hierarchy are not rectangles. (b) The desired output: a spatial treemap, in which as many external bottom-level adjacencies as possible have been kept while reshaping the regions at the top level into rectangles.

Preserving topology in spatial treemaps poses different challenges than in (non-hierarchical) rectangular cartograms. Topology-preserving rectangular cartograms exist under very mild conditions and can be constructed efficiently [5, 25]. As we show in this paper, this is not the case when a hierarchy is added to the picture.

In this paper we consider spatial treemaps with a two-level hierarchy. While the concept can easily be generalized to m levels for $m > 2$, the restriction to $m = 2$ is interesting on its own and already much more complex than the case of non-hierarchical spatial treemaps. The practical relevance of this case is supported by applications that use data with a two-level hierarchy [22].

For the purpose of this paper, building a spatial treemap is considered a two-phase process. In the first phase, a non-hierarchical rectangular cartogram, i.e., a rectangular subdivision, is produced from the original geographic regions. This can be done with one of the many algorithms for rectangular cartograms [5]. The result will contain all the bottom-level regions as rectangles, but the top-level regions will not be rectangular yet, thus will not represent the hierarchical structure. In the second phase, we convert such a cartogram into a two-level spatial treemap by making the top-level regions rectangles. It is at this stage where our algorithms come in and where we intend to preserve the topology of the underlying cartogram as much as possible. See Figure 2 for an example.

The advantage of this two-phase approach is that it allows for customization and user interaction. Interactive exploration of the data is essential when visualizing large amounts of data. The freedom to use an arbitrary rectangular cartogram algorithm in the first phase of the construction allows the user to prioritize the adjacencies that he or she considers most essential. In the second phase, our algorithm will produce a spatial treemap that aims to preserve as many of the adjacencies in the base cartogram as possible.

In addition, we go one step further and consider preserving the *orientations* of the adjacencies in the base cartogram (that is, whether two neighboring regions share a vertical or horizontal edge, and which one is on which side). This additional constraint is justified by the fact that the regions represent geographic or political regions, and relative positions between regions are an important factor when visualizing this type of data [5, 25]. The preservation of orientations has been studied for cartograms [8] and for (not necessarily adjacent) pairs of regions in treemaps [15], but to our knowledge, this is the first time oriented adjacencies are

explicitly considered for spatial treemaps.

We can distinguish three types of adjacency-relations: 1) *top-level adjacencies*, 2) *internal bottom-level adjacencies* (adjacencies between two rectangles that belong to the same top-level region), and 3) *external bottom-level adjacencies* (adjacencies between two rectangles that belong to different top-level regions). As we argue in the next section, we can always preserve all adjacencies of types 1 and 2 under a mild assumption. Hence the objective of our algorithms is to construct spatial treemaps that preserve as many adjacencies of type 3 as possible. We consider several variants of the problem, based on whether the orientations of the adjacencies have to be preserved, and whether the top-level subdivision is given in advance. In order to give efficient algorithms, we restrict ourselves to top-level regions that are orthogonally convex. This is a technical limitation that seems difficult to overcome, but that we expect does not limit the applicability of our results too much: our algorithms should still be useful for many practical instances, for example, by subdividing non-convex top-level regions into few convex pieces.

Results In the most constrained case in which adjacencies and their orientations need to be preserved and the top-level subdivision is given (Section 3.1), we solve the optimization problem to maximize the number of preserved external bottom-level adjacencies in $O(n)$ time, where n is the total number of (bottom-level) rectangles. The case in which the top-level subdivision is not fixed (Section 3.2) is much more challenging: it takes a combination of several techniques based on regular edge labelings to obtain an algorithm that solves the problem optimally in $O(k^4 \log k + n)$ time, for k being the number of top-level regions; we expect k to be much smaller than n. Finally, we prove that the case in which the orientations of adjacencies do not need to be preserved (Section 4) is NP-hard; we give worst-case bounds and an approximation algorithm.

2 Preliminaries

Rectangles and Subdivisions We work in the Euclidean plane $\mathbb{R}^2$ and consider all geometric objects as axis-aligned, rectilinear objects. So a *rectangle* in this text is always an axis-aligned rectangle. A set of rectangles $\mathcal{R}$ is called a *rectangle complex* if the interiors of none of the rectangles overlap, and each pair of rectangles is either completely disjoint or shares part of an edge; no two rectangles may meet in a single point. Each rectangle of a rectangle complex is called a *cell* of that complex. We represent rectangle complexes using a structure that has bidirectional pointers between neighboring cells.

Let $\mathcal{R}$ be a rectangle complex. The *boundary* of $\mathcal{R}$ is the boundary of the union of the rectangles in $\mathcal{R}$. Note that this is always a proper polygon, but it could have multiple components and holes. We say that $\mathcal{R}$ is *simple* if its boundary is a simple polygon, i.e., it is connected and has no holes. We say that $\mathcal{R}$ is *convex* if its boundary is orthogonally convex, i.e., the intersection of any horizontal or vertical line with $\mathcal{R}$ is either empty or a single line segment. We say that $\mathcal{R}$ is *rectangular* if its boundary is a rectangle.

Let $\mathcal{R}'$ be another rectangle complex. We say that $\mathcal{R}'$ is an *extension* of $\mathcal{R}$ if there is a bijective mapping between the cells in $\mathcal{R}$ and $\mathcal{R}'$ that preserves the adjacencies and their

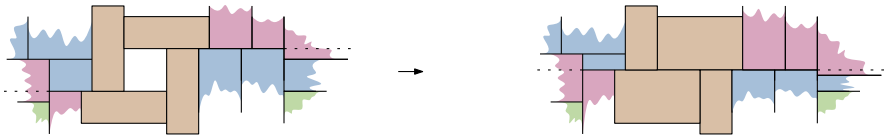


Figure 3: Removing a windmill hole (beige rectangles). Red rectangles get expanded; blue rectangles get compressed.

orientations. Note that $\mathcal{R}'$ could have adjacencies not present in $\mathcal{R}$ though. We say that $\mathcal{R}'$ is a *simple extension* of $\mathcal{R}$ if $\mathcal{R}$ is not simple but $\mathcal{R}'$ is; similarly we may call it a *convex extension* or a *rectangular extension*.

We show that every rectangle complex has a rectangular extension.

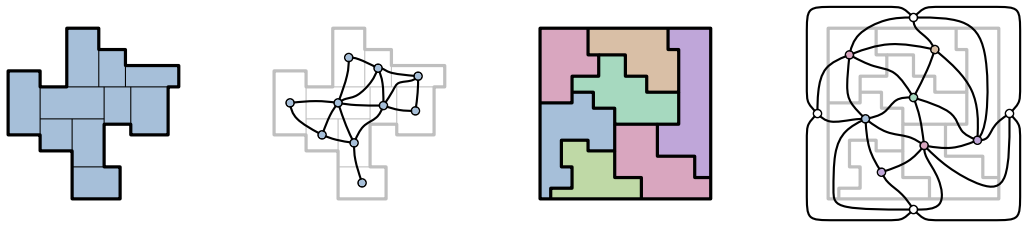
Lemma 1. *Let $\mathcal{R}$ be a rectangle complex. There always exists a rectangular extension of $\mathcal{R}$.*

Proof. We first augment $\mathcal{R}$ by four rectangles forming a bounding box of $\mathcal{R}$. Our goal is to extend the complex so that no holes inside the bounding box remain, while all existing adjacencies are preserved with their orientation. Obviously each hole is formed by at least four adjacent rectangles. Let H be a hole of the augmented complex. If there is a rectangle R adjacent to H with a full rectangle edge, we can extend R into the hole until it touches another rectangle. This either closes the hole or splits it into holes of lower complexity. Now let's assume that there is a hole without a rectangle adjacent to it along a full edge. Then each edge of the hole is a partial rectangle edge blocked by another rectangle. This is only possible in a "windmill" configuration of four rectangles cyclically blocking each other. But such a hole can be removed by moving two opposite edges e and f toward each other while expanding their own rectangles and shrinking the other two rectangles. Any rectangle outside the windmill intersected by the supporting lines of e and f also gets expanded or compressed. See Figure 3. None of the operations removes adjacencies or changes their orientations. □

We define $\mathbb{D} = \{\text{left}, \text{right}, \text{top}, \text{bottom}\}$ to be the set of the four cardinal directions. For a direction $d \in \mathbb{D}$ we use the notation $-d$ to refer to the direction opposite from d. We define an object $O \subset \mathbb{R}^2$ to be *extreme* in direction d with respect to a rectangle complex $\mathcal{R}$ if there is a point in O that is at least as far in direction d as any point in $\mathcal{R}$. Let $R \in \mathcal{R}$ be a cell, and $d \in \mathbb{D}$ a direction. We say R is *d-extensible* if there exists a rectangular extension $\mathcal{R}'$ of $\mathcal{R}$ in which R is extreme in direction d with respect to $\mathcal{R}'$ (or in other words, if its d-side is part of the boundary of $\mathcal{R}'$).

A set of simple rectangle complexes $\mathcal{L}$ is called a (rectilinear) *layout* if the boundary of the union of all complexes is a rectangle, the interiors of the complexes are disjoint, and no point in $\mathcal{L}$ belongs to more than three cells. If all complexes are rectangular we say that $\mathcal{L}$ is a *rectangular layout* or a *spatial treemap*. We call the rectangle bounding $\mathcal{L}$ the *root box*.

Let $\mathcal{L}$ be a rectilinear layout. We define the *top-level subdivision* $\mathcal{L}'$ of $\mathcal{L}$ as the subdivision of the root box of $\mathcal{L}$, in which the *top-level regions* are defined by the boundaries of the complexes in $\mathcal{L}$, which are assumed to be simple. We say $\mathcal{L}'$ is *rectangular* if all regions in $\mathcal{L}'$ are rectangles.



(a) (b) (c) (d)

Figure 4: (a) A rectangle complex. (b) The dual graph of the complex. (c) A top-level subdivision. (d) The extended dual graph of the top-level subdivision.

Dual Graphs of Rectangle Complexes The *dual graph* of a rectangle complex is an embedded planar graph with one vertex for each rectangle in the complex, and an edge between two vertices if the corresponding rectangles touch (have overlapping edge pieces). The *extended dual graph* of a rectangular rectangle complex has four additional vertices for the four sides of its rectangular boundary, and an edge between an original vertex and an additional vertex if the corresponding rectangle touches the corresponding side of the bounding box. We will be using dual graphs of entire spatial treemaps, of individual rectangle complexes, and of the top-level subdivision (ignoring the bottom-level rectangles); Figure 4 shows some examples. Extended dual graphs of rectangular rectangle complexes are fully triangulated (except for the outer face which is a quadrilateral), and the graphs that can arise in this way are characterized by the following lemma [13, 16, 25]:

Lemma 2. *A triangulated plane graph G with a quadrilateral outer face is the dual graph of a rectangular rectangle complex if and only if G has no separating triangles.*

Now, consider the three types of adjacencies we wish to preserve: 1) (top-level) adjacencies between top-level regions, 2) internal (bottom-level) adjacencies between the cells in one rectangle complex, and 3) external (bottom-level) adjacencies between cells of adjacent rectangle complexes.

Observation 1. *It is always possible to keep all internal bottom-level adjacencies.*

Observation 2. *It is possible to keep all top-level adjacencies if and only if the extended dual graph of the input top-level subdivision has no separating triangles.*

Observation 1 follows by applying Lemma 1 to all regions, and Observation 2 follows from Lemma 2, since the extended dual graph of the top-level regions is a triangulated plane graph with a quadrilateral outer face.

From now on we assume that the dual graph of the top-level subdivision has no separating triangles, and we will preserve all adjacencies of types 1 and 2. Unfortunately, it is not always possible to keep adjacencies of type 3—see Figure 5—and for every adjacency of type 3 that we fail to preserve, another adjacency that was not present in the input will appear. Therefore, our aim is to preserve as many of these adjacencies as possible.

We can now phrase our generic problem statement and its variations.

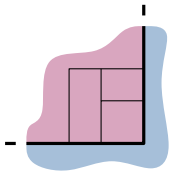


Figure 5: Not all external adjacencies can be kept.

Problem 1. Given a rectilinear input layout $\mathcal{L}$ as a two-level subdivision of the root box of $\mathcal{L}$ into (top-level) convex rectangle complexes, each of which is subdivided into bottom-level rectangles, create a spatial treemap $\mathcal{L}'$ of $\mathcal{L}$ that preserves all top-level and internal bottom-level adjacencies (types 1 and 2) and maximizes the number of preserved external bottom-level adjacencies (type 3).

The next section considers the case that the orientations of the bottom-level adjacencies (types 2 and 3) must be preserved, either for a given rectangular top-level subdivision, the *target subdivision* $\mathcal{T}$, or optimized over all possible rectangular top-level subdivisions. Section 4 considers all bottom-level adjacencies without their original orientations.

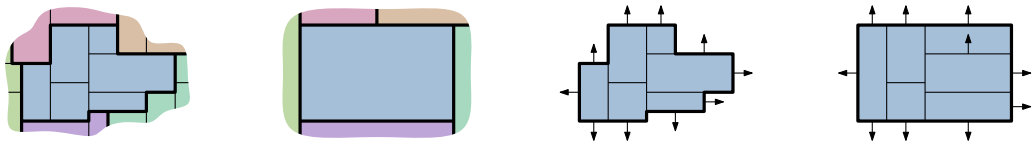
3 Preserving orientations

We begin studying the version of the problem where all internal adjacencies have to be preserved respecting their original orientations. Additionally, we want to maximize the number of preserved and correctly oriented (bottom-level) external adjacencies. We consider two scenarios: first we assume that the target top-level subdivision $\mathcal{T}$ is part of the input, and then we study the case in which we optimize over all possible top-level subdivisions. The former situation is particularly interesting in certain applications [15, 22], in which the user specifies a top-level layout that needs to be filled with the bottom-level cells. If, however, the bottom-level adjacencies are more important, then optimizing over top-level subdivisions allows to preserve more external adjacencies.

3.1 Given the top-level subdivision

In this section we are given, in addition to the initial two-level input layout $\mathcal{L}$, a target top-level subdivision $\mathcal{T}$. The goal is to find a two-level treemap that has $\mathcal{T}$ as its top-level subdivision that preserves all oriented bottom-level internal adjacencies and that maximizes the number of preserved oriented bottom-level external adjacencies in the output.

First observe that in $\mathcal{T}$ any two neighboring top-level regions have a single orientation for their adjacency. Hence we can only keep those bottom-level external adjacencies that have the same orientation in the input as their corresponding top-level regions have in the output layout. Secondly, consider a rectangle R in a complex $\mathcal{R}$, and a rectangle B in another complex $\mathcal{B}$. Observe that if R and B are adjacent in the input, for example with R to the left of B, then their adjacency can be preserved only if R is right-extensible in $\mathcal{R}$ and B is left-extensible in $\mathcal{B}$.



(a) (b) (c) (d)

Figure 6: (a) A top-level region in the input. (b) The same region in the target top-level subdivision. (c) Edges of rectangles that want to become part of a boundary have been marked with arrows. Note that one rectangle wants to become part of the top boundary but cannot, because it is not extensible in that direction. (d) All arrows that are not blocked can be satisfied.

The main result in this section is that the previous two conditions are enough to describe all adjacencies that cannot be preserved, whereas all the other ones can be kept. Furthermore, we will show how to decide extensibility for convex complexes, and how to construct a final solution that preserves all possible adjacencies, leading to an algorithm for the optimal solution.

Recall that we assume all top-level regions are orthogonally convex. Consider each rectangle complex $\mathcal{R}$ of $\mathcal{L}$ separately. Since we know the target subdivision $\mathcal{T}$ and since all cells externally adjacent to $\mathcal{R}$ are consecutive along its boundary, we can immediately determine the cells on each of the four sides of the output region (see Figure 6). The reason is that for a rectangle R that is exterior to its region $\mathcal{R}$, and that is adjacent to another rectangle $B \in \mathcal{B}$, their adjacency is relevant only if $\mathcal{R}$ and $\mathcal{B}$ are adjacent with the same orientation in the target top-level subdivision. We can easily categorize the extensible rectangles of a convex rectangle complex.

Lemma 3. *Let $\mathcal{R}$ be a convex rectangle complex, let $R \in \mathcal{R}$ be a rectangle, and $d \in \mathbb{D}$ a direction. R is d-extensible if and only if there is no rectangle $R' \in \mathcal{R}$ directly adjacent to R on the d-side of R.*

Proof. For the 'only if' part, simply note that if there is such a rectangle $R' \in \mathcal{R}$, then the adjacency between R and R' must be preserved, with its original orientation. Hence there is always a point in R' that is further in direction d than any point in R. So R is not d-extensible.

For the 'if' part, consider the complex obtained by extending R in direction d until it becomes extreme in that direction. This is always possible because $\mathcal{R}$ is convex; the resulting complex is still simple. Now we add a temporary bounding box consisting of four rectangles around $\mathcal{R}$, one in each direction, such that R is adjacent to the one on the d-side. Then we can apply Lemma 1 and find a rectangular extension of $\mathcal{R}$ where R is extreme on the d-side. □

Unfortunately, though, we cannot extend all extensible rectangles at the same time. However, we show that we can actually extend all those rectangles that we want to extend for an optimal solution.

We call a rectangle of a certain complex belonging to a top-level region *engaged* if it

wants to be adjacent to a rectangle of another top-level region, and the direction of their desired adjacency is the same as the direction of the adjacency between these two regions in $\mathcal{T}$. We say it is d-engaged if this direction is $d \in \mathbb{D}$.

Therefore, the rectangles that we want to extend are exactly those that are d-extensible and d-engaged, since they are the only ones that help preserve bottom-level exterior adjacencies. It turns out that extending all these rectangles is possible, because the engaged rectangles of $\mathcal{R}$ have a special property:

Lemma 4. *If we walk around the boundary of a simple rectangle complex $\mathcal{R}$, we encounter all d-engaged rectangles consecutively.*

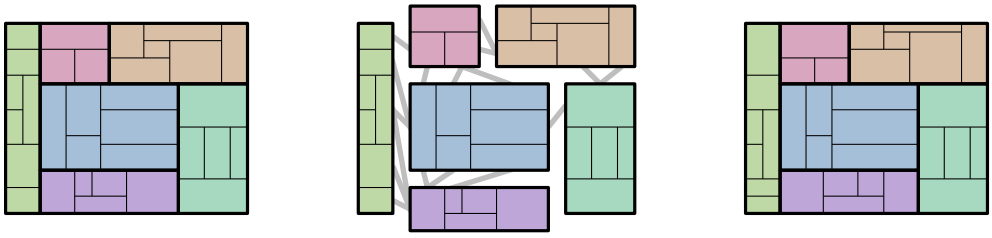
Proof. Suppose that when walking clockwise along the boundary of $\mathcal{R}$ we encounter rectangles R_1, R_2, R_3 that are d-,d'-, and d-engaged, respectively. Since R_1 and R_3 are both d-engaged, they have the same direction of external adjacency in $\mathcal{L}$. However, if $d' \neq d$, then R_2 has a different direction, implying that in $\mathcal{L}$ this is also the same way. This contradicts the fact that in the target top-level subdivision $\mathcal{T}$ the complex $\mathcal{R}$ is a rectangle, so the rectangles engaged in the four different directions appear contiguously. □

This property of d-engaged rectangles is useful due to the following fact.

Lemma 5. *Let $\mathcal{R}$ be a convex rectangle complex composed of r rectangles, and let S be a subset of the extensible and engaged rectangles in $\mathcal{R}$ with the property that if we order them according to a clockwise walk along the boundary of $\mathcal{R}$, all d-extensible rectangles in S are encountered consecutively for each $d \in \mathbb{D}$ and in the correct clockwise order. We can compute, in $O(r)$ time, a rectangular extension $\mathcal{R}'$ of $\mathcal{R}$ in which all d-extensible rectangles in S are extreme in direction d, for all $d \in \mathbb{D}$.*

Proof. We use the same idea as in the proof of Lemma 3, but now we extend all rectangles in S at the same time. Since by Lemma 4 all d-engaged rectangles appear consecutively around the boundary of $\mathcal{R}$, there cannot be any conflicts preventing the extension of d-engaged rectangles: rectangles with the same direction extend all toward the same side, thus they can all be made extreme. On the other hand, two rectangles extended toward different directions cannot influence each other because that would imply that the directions do not appear contiguously or in the wrong order.

It remains to apply Lemma 1 and show that the rectangular extension can be found in linear time. Since $\mathcal{R}$ with the rectangles in S extended is still a simple polygon, all holes of the complex after augmenting it by the four external rectangles are adjacent to the external rectangles. Hence there are no windmill holes. We can then start walking clockwise along the boundary of $\mathcal{R}$ at the first d-extended rectangle and extend all d-extensible rectangles until we reach the first $d+1$-extended rectangle. None of these rectangles is blocked in direction d. We close the corner between the d-side and the $d+1$-side by extending either the last d-extended rectangle in direction $d+1$ or vice versa, which is always possible since they cannot both block each other. We continue this process along all four sides of $\mathcal{R}$. Let H be a remaining hole on the d-side. It has the property that none of its adjacent rectangles is d-extensible and that it is bounded by two staircases. We can then close the hole in linear



(a) (b) (c)

Figure 7: (a) After we solved all the different top-level regions separately, we do not necessarily have the right adjacencies yet. (b) We can indicate the desired adjacencies that are still possible (i.e., the adjacencies between two edges of rectangles that actually ended up on the outside) as a graph. Note that this graph is planar. (c) We can modify the insides of the rectangles to realize all desired adjacencies.

time by simultaneously walking along the two staircases and maximally extending rectangles orthogonally to direction d. □

By Lemma 4 the engaged and extensible rectangles form a subset of rectangles for which Lemma 5 holds, thus by using the lemma we can find a rectangular extension where all extensible and engaged rectangles are extreme in the appropriate direction.

Then we can apply this idea to each region. Now we still have to match up the adjacencies in an optimal way, that is, preserving as many adjacencies from the input as possible. This can be done by matching horizontal and vertical adjacencies independently. It is always possible to get all the external bottom-level adjacencies that need to be preserved. This can be seen as follows (see also Figure 7). We process first all horizontal adjacencies. Consider a maximal horizontal segment in the target top-level subdivision. Then the position and length of the boundary of each region adjacent to that segment are fixed, from the target top-level subdivision. The only freedom left is in the x-coordinates of the vertical edges of the bottom-level rectangles that form part of that boundary (except for the leftmost and rightmost borders of each region, which are also fixed). The ordered sequences of the rectangles on either side of the horizontal segment are compatible with the order of the adjacencies to be preserved across the segment. Therefore, even if the x-coordinates on one side of the segment are already given, we can assign the x-coordinates on the other side such that all adjacencies are satisfied. Hence we can process all maximal horizontal segments in a monotone bottom-up fashion and realize all desired vertical adjacencies. The vertical boundaries are independent, thus can be processed in exactly the same way to realize all desired horizontal adjacencies. This yields the main theorem in this subsection.

Theorem 1. *Let $\mathcal{L}$ be a two-level rectilinear layout, where n is the number of cells in the bottom level, and where all top-level regions are orthogonally convex. For a given target top-level subdivision $\mathcal{T}$, we can find, in $O(n)$ time, a rectangular layout of $\mathcal{L}$ that respects $\mathcal{T}$, preserves all oriented internal bottom-level adjacencies, and preserves as many oriented external bottom-level adjacencies as possible.*

3.2 Unconstrained top-level subdivision

In this section the target top-level subdivision of the rectangle complexes is not given, i.e., we are given a rectilinear input layout and need to find a rectangular output layout preserving all adjacencies of the rectangle complexes and preserving a maximum number of adjacencies of the cells of different complexes.

We can represent a particular rectangular top-level subdivision $\mathcal{L}$ as a *regular edge labeling* [14] of the dual graph of the top-level subdivision. Let $G(\mathcal{L})$ be the extended dual graph of $\mathcal{L}$. Then $\mathcal{L}$ induces an edge labeling as follows: an edge corresponding to a joint vertical (horizontal) boundary of two rectangular complexes is colored blue (red). Furthermore, blue edges are directed from left to right and red edges from bottom to top. Clearly, the edge labeling obtained from $\mathcal{L}$ in this way satisfies that around each inner vertex v of $G(\mathcal{L})$ the incident edges with the same color and the same direction form contiguous blocks around v. The edges incident to each of the external vertices $\{l, t, r, b\}$ all have the same label. Such an edge labeling is called *regular* [14]. Each regular edge labeling of the extended dual graph $G(\mathcal{L})$ defines an equivalence class of top-level subdivisions and our goal is to optimize over those.

The family of all possible output top-level rectangular subdivisions shares the same dual graph, i.e., the dual graph of the target top-level subdivision. In order to represent this family we apply a technique described by Eppstein et al. [8]. Let $\mathcal{L}$ be the input rectilinear top-level subdivision, and let $G(\mathcal{L})$ be its extended dual graph. The first step is to decompose $G(\mathcal{L})$ by its separating 4-cycles into minors called *separation components* with the property that they do not have non-trivial separating 4-cycles any more, i.e., 4-cycles with more than a single vertex in the inner part of the cycle. If C is a separating 4-cycle, then the interior separation component consists of C and the subgraph induced by the vertices interior to C. The outer separation component is obtained by replacing all vertices in the interior of C by a single vertex connected to each vertex of C. This decomposition can be obtained in linear time [8]. We can then treat each component in the decomposition independently and finally construct an optimal rectangular top-level subdivision from the optimal solutions of its descendants in the decomposition tree. Hence we consider a single component of the decomposition, which by construction has no non-trivial separating 4-cycles.

3.2.1 Preprocessing of the bottom level

We start with a preprocessing step to compute the number of realizable external bottom-level adjacencies for pairs of adjacent top-level regions. This allows us to ignore the bottom-level cells in later steps and to focus on the top-level subdivision and orientations of top-level adjacencies.

Let $\mathcal{L}$ be an output rectangular top-level subdivision, let $\mathcal{R}$ and $\mathcal{S}$ be two adjacent rectangle complexes in $\mathcal{L}$, and let $d \in \mathbb{D}$ be an orientation. Then we define $\omega(\mathcal{R}, \mathcal{S}, d)$ to be the total number of adjacencies between d-engaged and d-extensible rectangles in $\mathcal{R}$ and $-d$-engaged and $-d$-extensible rectangles in $\mathcal{S}$. By Lemma 5 there is a rectangular extension of $\mathcal{R}$ and $\mathcal{S}$ with exactly $\omega(\mathcal{R}, \mathcal{S}, d)$ external bottom-level adjacencies between $\mathcal{R}$ and $\mathcal{S}$.

We show the following (perhaps surprising) lemma:

Lemma 6. *For any pair $\mathcal{L}$ and $\mathcal{L}'$ of rectangular top-level subdivisions and any pair $\mathcal{R}$ and $\mathcal{S}$ of rectangular rectangle complexes, whose adjacency direction with respect to $\mathcal{R}$ is d in $\mathcal{L}$ and d' in $\mathcal{L}'$, the number of external bottom level adjacencies between $\mathcal{R}$ and $\mathcal{S}$ in any optimal solution for $\mathcal{L}'$ differs by $\omega(\mathcal{R},\mathcal{S},d') - \omega(\mathcal{R},\mathcal{S},d)$ from $\mathcal{L}$. For adjacent rectangle complexes whose adjacency direction is the same in both top-level subdivisions the number of adjacencies in any optimal solution remains the same.*

Proof. The value $\omega(\mathcal{R},\mathcal{S},d)$ is the maximum number of external bottom-level adjacencies between $\mathcal{R}$ and $\mathcal{S}$ that can be realized if $\mathcal{S}$ is adjacent to $\mathcal{R}$ in direction d. By Theorem 1 there is a rectangular layout respecting the top-level subdivision $\mathcal{L}$ in which this number of adjacencies between $\mathcal{R}$ and $\mathcal{S}$ is realized. So clearly the difference in $\mathcal{R}$-$\mathcal{S}$ adjacencies is $\omega(\mathcal{R},\mathcal{S},d') - \omega(\mathcal{R},\mathcal{S},d)$. Adjacent pairs of rectangle complexes whose adjacency direction remains the same are not affected by changes of adjacency directions elsewhere in the top-level subdivision. □

This basically means we can consider changes of adjacency directions locally and independent from the rest of the subdivision. Furthermore, since the values $\omega(\mathcal{R},\mathcal{S},d)$ are directly obtained from counting the numbers of d-extensible and d-engaged rectangles in $\mathcal{R}$ (or $-d$-extensible and $-d$-engaged rectangles in $\mathcal{S}$) we get the next lemma.

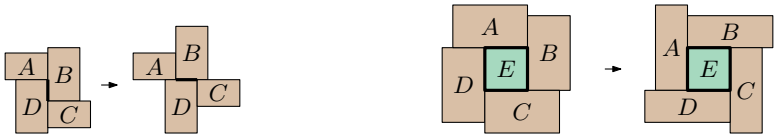
Lemma 7. *We can compute all values $\omega(\mathcal{R},\mathcal{S},d)$ in $O(n)$ total time.*

3.2.2 Optimizing in a graph without separating 4-cycles

Here we will prove the following:

Theorem 2. *Let G be an embedded triangulated planar graph with k' vertices without separating 3-cycles and without non-trivial separating 4-cycles, except for the outer face which consists of exactly four vertices. Furthermore, let a weight $\omega(e,d)$ be assigned to every edge e in G and every orientation d in $\mathbb{D}$. Then we can find a rectangular subdivision of which G is the extended dual that maximizes the total weight of the directed adjacencies in $O(k'^4 \log k')$ time.*

In order to optimize over all rectangular subdivisions with the same extended dual graph we make use of the representation of these subdivisions as elements in a distributive lattice or, equivalently, as closures in a partial order induced by this lattice [8]. There are two *moves* or *flips* by which we can transform one rectangular subdivision (or its regular edge labeling) into another one, *edge flips* and *vertex flips* (Figure 8). They form a graph where each equivalence class of rectangular subdivisions is a vertex and two vertices are connected by an edge if they are transformable into each other by a single move, with the edge directed toward the more counterclockwise subdivision with respect to this move. This graph is acyclic and its reachability ordering is a distributive lattice [10]. It has a minimal (maximal) element that is obtained by repeatedly performing clockwise (counterclockwise) moves.



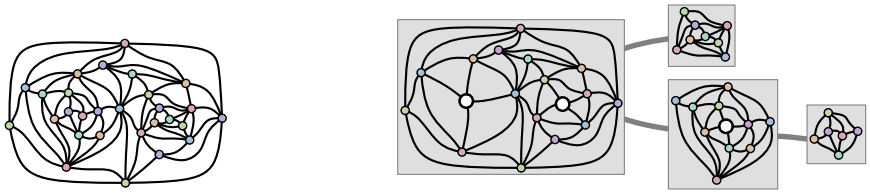
(a) edge flip BD (b) vertex flip E

Figure 8: Flip operations

By Birkhoff's representation theorem [3] each element in this lattice is in one-to-one correspondence to a partition of a partial order $\mathcal{P}$ into an upward-closed set U and a downward-closed set L. The elements in $\mathcal{P}$ are pairs (x, i), where x is a flippable item, i.e., either the edge of an edge flip or the vertex of a vertex flip [8]. The integer i is the so-called flipping number $f_x(\mathcal{L})$ of x in a particular subdivision $\mathcal{L}$, i.e., the well-defined number of times flip x is performed counterclockwise on any path from the minimal element $\mathcal{L}_{\min}$ to $\mathcal{L}$ in the distributive lattice. An element (x, i) is smaller than another element (y, j) in this order if y cannot be flipped for the j-th time before x is flipped for the i-th time. For each upward- and downward-closed partition U and L, the corresponding subdivision can be reconstructed by performing all flips in the lower set L. $\mathcal{P}$ has $O(k'^2)$ vertices and edges and can be constructed in $O(k'^2)$ time [8]. The construction starts with an arbitrary subdivision, performs a sequence of clockwise moves until we reach $\mathcal{L}_{\min}$, and from there performs a sequence of counterclockwise moves until we reach the maximal element. During this last process we count how often each element is flipped, which determines all pairs (x, i) of $\mathcal{P}$. Since each flip (x, i) affects only those flippable items that belong to the same triangle as x, we can initialize a queue of possible flips, and iteratively extract the next flip and add the new flips to the queue in total time $O(k'^2)$. In order to create the edges in $\mathcal{P}$ we again use the fact that a flip (x, i) depends only on flips (x', i'), where x' belongs to the same triangle as x and i' differs by at most 1 from i. The actual dependencies can be obtained from their states in $\mathcal{L}_{\min}$.

Next, we assign weights to the nodes in $\mathcal{P}$. Let $\mathcal{L}_{\min}$ be the subdivision that is minimal in the distributive lattice, i.e., the subdivision where no more clockwise flips are possible. For an edge-flip node (e, i) let $\mathcal{R}$ and $\mathcal{S}$ be the two rectangle complexes adjacent across e. Then the weight $\omega(e, i)$ is obtained as follows. Starting with the adjacency direction between $\mathcal{R}$ and $\mathcal{S}$ in $\mathcal{L}_{\min}$ we cycle i times through the set $\mathbb{D}$ in counterclockwise fashion. Let d be the i-th direction and d' the $(i+1)$-th direction. Then $\omega(e, i) = \omega(e, d') = \omega(\mathcal{R}, \mathcal{S}, d') - \omega(\mathcal{R}, \mathcal{S}, d)$. For a vertex-flip node (v, i) let $\mathcal{R}$ be the degree-4 rectangle complex surrounded by the four complexes $\mathcal{S}_1, \ldots, \mathcal{S}_4$. We again determine the adjacency directions between $\mathcal{R}$ and $\mathcal{S}_1, \ldots, \mathcal{S}_4$ in $\mathcal{L}_{\min}$ and cycle i times through $\mathbb{D}$ to obtain the i-th directions $d_1, \ldots, d_4$ as well as the $(i+1)$-th directions $d'_1, \ldots, d'_4$. Then $\omega(v, i) = \sum_{j=1}^{4} \omega(\mathcal{R}, \mathcal{S}_j, d'_j) - \omega(\mathcal{R}, \mathcal{S}_j, d_j)$. Equivalently, if the four edges incident to v are $e_1, \ldots, e_4$, we have $\omega(v, i) = \sum_{j=1}^{4} \omega(e_j, d'_j)$.

Finally, we compute a maximum-weight closure of $\mathcal{P}$ using a max-flow algorithm [1, Chapter 19.2], which will take $O(k'^4 \log k')$ time for a graph with $O(k'^2)$ nodes. Recall that by Birkhoff's representation theorem [3] each subdivision in the distributive lattice corresponds to a partition of $\mathcal{P}$ into an upward-closed set U and a downward-closed set L as induced by a closure of $\mathcal{P}$. Hence a maximum-weight closure of $\mathcal{P}$ corresponds to an output top-level



(a) (b)

Figure 9: (a) A graph with non-trivial separating 4-cycles. Note that some 4-cycles intersect each other. (b) A possible decomposition tree of 4-cycle-free graphs (root on the left).

subdivision with the maximum number of preserved adjacencies.

3.2.3 Optimizing in General Graphs

In this section, we show how to remove the restriction that the graph should have no separating 4-cycles. We do this by decomposing the graph G by its separating 4-cycles and solving the subproblems in a bottom-up fashion.

Lemma 8 (Eppstein et al. [8])**.** *Given a plane graph G with k vertices, there exists a collection $\mathcal{C}$ of separating 4-cycles in G that decomposes G into separation components that do not contain separating 4-cycles any more. Such a collection $\mathcal{C}$ and the decomposition can be computed in $O(k)$ time.*

These cycles naturally subdivide G into a tree of subgraphs, which we will denote as T_G. Still following [8], we add an extra artificial vertex inside each 4-cycle, which corresponds to filling the void in the subdivision after removing all rectangles inside by a single rectangle. Figure 9 shows an example of a graph G and a corresponding tree T_G.

Now, all nodes of T_G have an associated graph without separating 4-cycles on which we can apply Theorem 2. The only thing left to do is assign the correct weights to the edges of these graphs. For a given node ν of T_G, let G_ν be the subgraph of G associated to ν (with potentially extra vertices inside its 4-cycles).

For every leaf ν of T_G, we assign weights to the internal edges of G_ν by setting $\omega(e, d) = \omega(\mathcal{R}, \mathcal{S}, d)$ if e separates $\mathcal{R}$ and $\mathcal{S}$ in the top-level subdivision $\mathcal{L}$. For the external edges of G_ν (the edges that are incident to one of the "corner" vertices of the outer face), we fix the orientations in the four possible ways, leading to four different problems. We apply Theorem 2 four times, once for each orientation. We store the resulting solution values as well as the corresponding optimal subdivisions at ν in T_G.

Now, in bottom-up order, for each internal node ν in T_G, we proceed in a similar way with one important change: for each child μ of ν, we first look up the four optimal subdivisions of μ and incorporate them in the weights of the four edges incident to the single extra vertex that replaced G_μ in G_ν. Since these four edges must necessarily have four different orientations, their states are linked, and it does not matter how we distribute the weight over them; we can simply set the weight of three of these edges to 0 and the remaining

one to the solution of the appropriately oriented subproblem. The weights of the remaining edges are derived from $\mathcal{L}$ as before, and again we fix the orientations of the external edges of G_ν in four different ways and apply Theorem 2 to each of them. We again store the resulting four optimal values and the corresponding subdivisions at ν, in which we insert the correctly oriented subsolutions for all children μ of ν.

This whole process takes $O(k^4 \log k)$ time in the worst case. Finally, since weights are expressed as differences with respect to the minimal subdivision $\mathcal{L}_{\min}$ we compute the value of $\mathcal{L}_{\min}$ and add the offset computed as the optimal solution to get the actual value of the globally optimal solution. This takes $O(n)$ time.

Theorem 3. *Let $\mathcal{L}$ be a two-level rectilinear layout, such that the extended dual graph G of the top-level subdivision has no separating 3-cycles. Let n be the number of cells in the bottom level and k the number of regions in the top level. Then we can find a spatial treemap that preserves all oriented internal bottom-level adjacencies, and preserves as many oriented external bottom-level adjacencies as possible in $O(k^4 \log k + n)$ time.*

4 Without preserving orientations

In this section we study the variant of the problem where we do not need to preserve the orientations of the adjacencies that are maintained in the output treemap. We still assume that the required top-level subdivision of the output treemap is given in advance.

We first define the *boundary adjacency graph* G on the boundary cells of all rectangle complexes. There is a vertex in G for each cell that belongs to the boundary of a rectangle complex. Since the top-level subdivision is given, we know where the four corners separating the boundary sides are. There is an *internal adjacency* edge in G between any two vertices of the same rectangle complex whose cells are adjacent in the complex. There are *external adjacency* edges in G between vertices of cells of different rectangle complexes if the cells are adjacent in the input.

Next we note that the boundary adjacency graph G as a subgraph of the dual graph for the input layout is a planar graph. Our goal is to select subsets of the vertices to be on the boundary of each rectangle complex whose induced subgraph has as many external adjacency edges as possible but also would not create any separating triangles if we imagine connecting all the external adjacencies corresponding to one boundary to one vertex. Recall from Lemma 2 that a corresponding rectangular rectangle complex exists if and only if the dual graph has no separating triangles.

For the remainder we restrict ourselves to the case of two top-level regions, and only at the very end extend the arguments to more regions. From our boundary adjacency graph we remove those internal adjacency edges that correspond to two directly neighboring cells when traversing the boundary of the corresponding rectangle complex. In the remaining graph, we cannot maintain the external adjacencies of any two cells that remain connected by an internal adjacency edge since these two cells, if aligned at the same external side of the top-level rectangle, would form a separating triangle together with the outside of their top-level rectangle, e.g., consider a neighboring pair of a red and a green cell in Fig. 11. Hence, we need to find an independent set of vertices in this remaining adjacency graph. We

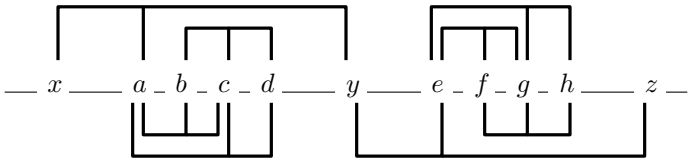


Figure 10: Equality gadget $x \Leftrightarrow z$

first prove that preserving as many bottom-level external adjacencies as possible is NP-hard already for the case of two top-level regions.

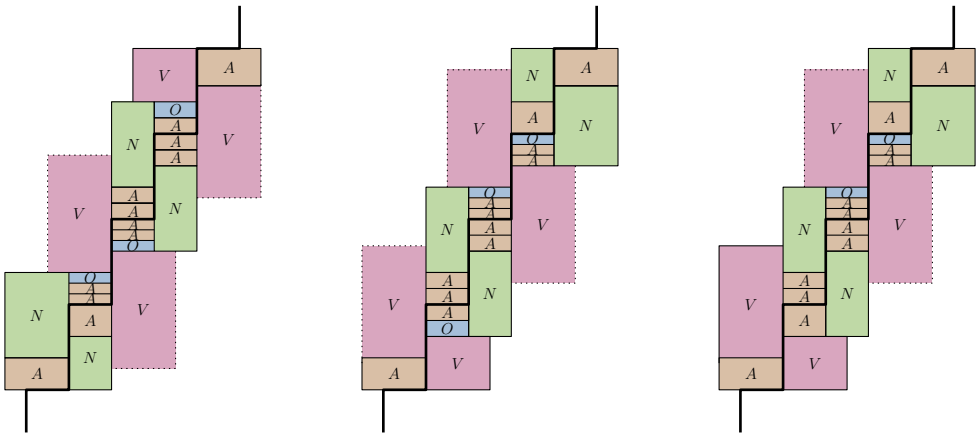
4.1 NP-hardness

In positive 1-in-3-SAT each clause contains exactly three (non-negated) variables, and we need to decide whether there is a truth assignment such that exactly one variable per clause is true. As input we are given the collection of clauses together with a planar embedding of the associated graph such that all variables are on a straight line and no edge crosses the straight line. This problem was shown to be NP-complete by Mulzer and Rote [18]. We reduce from the following NP-hard variant of positive 1-in-3-SAT.

Lemma 9. *Planar positive 1-in-3-SAT with variables on a line and with every variable occurring in at least one clause on each side of the line and in at most three clauses is NP-hard.*

Proof. Consider the *equality gadget* in Figure 10. It enforces that x and z are equivalent. The equality gadget consists of the same clauses as the equality gadget in [18], but it arranges them such that any variable other than x and z occurs in a clause on each side of the line. Concatenating equality gadgets gives us a sequence of equivalent variables $x_0, \ldots, x_{k+1}$, where x_0 and x_{k+1} occur in exactly one clause and the other variables occur in exactly two clauses, one clause on each side of the line. Given an instance of planar positive 1-in-3-SAT, we replace any variable that occurs in $k > 1$ clauses by such a sequence, and use $x_1, \ldots, x_k$ to connect to one additional clause each. We connect any remaining variable x that only occurs in one clause (like x_0 and x_{k+1}) to two new identical clauses $(x \vee a \vee b)$ and $(x \vee a \vee b)$ with additional variables a and b, one on each side of the line. □

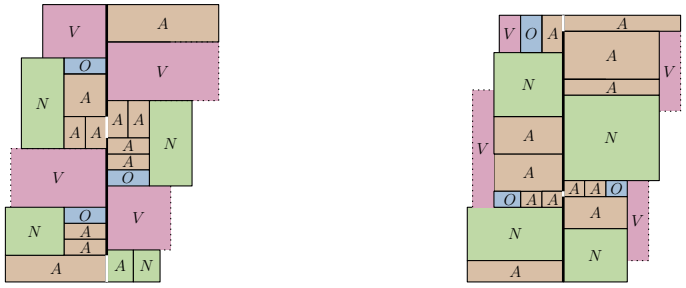
In the reduction we have two top-level regions with a staircase boundary between them. For each variable we have a variable gadget consisting of a set of rectangles on both sides of the boundary. Figure 11 shows the three variable gadgets used. Which gadget is used, depends on the number of occurrences in clauses. From the V- and N-rectangles we can on each side of the boundary only keep an independent set (in terms of the adjacency graph) since any adjacent pair would induce a separating triangle due to the A-rectangles. A V-rectangle extends beyond its vertical and/or its horizontal dotted line; we call such a V-rectangle *connecting*. O-rectangles are only placed opposite to those V-rectangles that are extended in this way. The complete boundary is filled with vertex gadgets.



(a) Type (a) gadget (b) Type (b) gadget (c) Type (c) gadget

Figure 11: A set of rectangles representing a variable. (a) one occurrence above the line, two below; (b) two above, one below; (c) one above and one below.

A clause is represented by three adjacencies: for a clause on the left of the boundary, from the type-(a) variable gadget a V-rectangle extends upwards and from the type-(c) variable gadget a V-rectangle extends to the left, so that the two rectangles touch. From the type-(b) vertex gadget a V-rectangle extends to the left and upwards, so that it touches the two other rectangles. The case to the right of the boundary is analogous. We do this for every clause. Finally any empty spaces are filled with rectangles greedily.



(a) true (b) false

Figure 12: Subdivisions corresponding to a variable assignment for Type (a) gadget

For the moment assume that for a variable either all V-rectangles (and no N-rectangles) are kept on the boundary (this corresponds to setting a variable to true) or all N-rectangles (and no V-rectangles) are kept (this corresponds to setting a variable to false). Additionally we keep as many A- and O-rectangles as possible. These cases are illustrated for the Type (a) gadget in Figure 12. In the figure the boundary between the top-level regions is drawn in white for adjacencies that have been newly added and therefore do not count below. In the case that we keep all V-rectangles (Figure 12(a)) we can achieve thirteen or fourteen adjacencies depending on whether the variable occurs two or three times. Thus, we achieve eleven adjacencies plus the number of occurrences of the variable in clauses. In the case that

we keep all N-rectangles we can achieve eleven adjacencies (Figure 12(b)). Additionally, we get one adjacency for each pair of neighboring vertex gadgets. For the three variable gadgets involved in the same clause gadget, we can for at most one keep all V-variables since the clause gadget would otherwise yield a separating triangle. Thus if we have m variables and n clauses, the total number of adjacencies is $11m + m - 1$ plus the number of occurrences of variables that have been set to true. Now if the original formula has a satisfying assignment with exactly one variable true per clause then the number of adjacencies we can achieve in this way is $12m - 1 + n$, and if there is no such assignment the number of adjacencies is smaller.

It remains to show that we can indeed assume that for a vertex gadget we have only V-rectangles or only N-rectangles on the boundary. First observe that as long as we keep all connecting V-rectangles on the boundary, there is no advantage in dropping any of the remaining ones. It remains to prove that we do not get more than eight adjacencies if we do not choose to keep all connecting V-rectangles on the boundary; this implies that such a configuration has no advantage over the one with all N-rectangles, thus we can replace it by the latter.

We refer to external adjacencies by the symbols of the rectangles, e.g., we call an adjacency between a V- and an A-rectangle a V-A adjacency. We now go through all cases. If we keep the N-rectangles on one side and the V-rectangles on the other side then we get no V-V or N-N adjacencies, three V-A, three N-A, at most two V-O adjacencies and three A-A adjacencies, thus at most eleven. There is one configuration in which we keep a pair of externally-adjacent V-rectangles and a pair of externally-adjacent N-rectangles. In this case we have one V-V, one N-N, two V-A, two N-A, at most two V-O and three A-A adjacencies, thus at most ten. If we keep three V-rectangles, but not all connecting ones, then we loose at least three adjacencies, so we keep at most eleven. All other cases give subsets of the cases above, and leave us with even fewer (at most ten) adjacencies. Thus, the assumption was valid. From this the following theorem follows.

Theorem 4. *Given a rectilinear input layout $\mathcal{L}$, finding a rectangular layout that respects the top-level subdivision of $\mathcal{L}$, preserves all internal adjacencies, and preserves as many bottom-level external adjacencies as possible is NP-hard.*

4.2 Upper bound

We now show that it is sometimes not possible to preserve more than a factor 1/4 of the external adjacencies.

We will construct an example with $4h + 7$ boundary rectangles (and therefore $4h + 6$) external adjacencies between two regions. To be more precise, we will construct a graph which we show is the dual graph of a pair of rectangle complexes. Figure 13 illustrates the construction. The graph consists of two pieces (one for each rectangle complex). The top piece has 7 vertices, 3 of which are isolated and 4 of which are connected by a maximal outerplanar graph. The bottom piece consists of $4h$ vertices, in four groups of h. The first and third groups of vertices are all isolated, while the vertices of the second and fourth groups are pairwise connected. Finally, we add a complete planar bipartite graph between

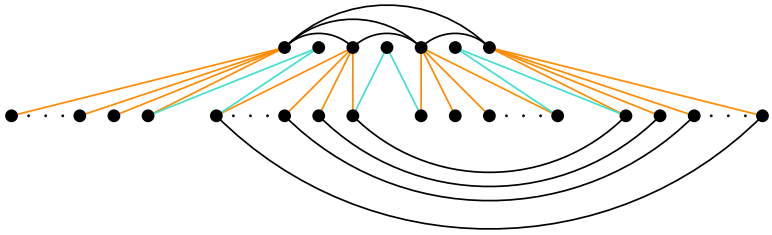


Figure 13: Illustration of the construction showing that we cannot keep more than a quarter of the external adjacencies. Black edges indicate internal adjacencies; orange and blue edges are external adjacencies. Blue edges are incident to an isolated vertex in the top graph.

the two groups of vertices by connecting each group of h vertices to one of the 4 connected vertices in the top graph, and filling the gaps with 6 additional edges.

Claim 1. We cannot preserve more than $h + 6$ edges in this construction.

Proof. Recall that we need to take an independent set in both the top and bottom outerplanar graphs. This implies that in the top graph, if we keep the first or fifth vertex (from left to right), we cannot keep any of the other three. Therefore, we can either keep the complete first or third group of h edges, or a combination of edges in the second and fourth group together. However, each edge in the second group is connected to an edge in the fourth group via an internal adjacency in the bottom graph, so we can keep at most half of the edges in these groups together: at most h. Since there are only 6 edges not in a group, we can keep at most $h + 6$ edges. □

If desired, we can create another copy of the construction and place it upside down to have a non-constant number of vertices at the top as well as at the bottom.

Theorem 5. *There exist rectilinear input layouts for which no spatial treemap exists that respects the top-level adjacencies, preserves all internal adjacencies, and preserves more than* $1/4$ *of the bottom-level external adjacencies.*

4.3 Algorithm

We first describe an algorithm for two top-level regions. The connected components of internal adjacency edges form outerplanar graphs (which have treewidth at most 2) on which we can solve the maximum weight independent set problem in linear time exactly [2] (where the weight of a vertex is its degree in terms of external adjacency edges). We first solve the maximum weight independent set problem for one of the sides of the boundary exactly and then solve it for the other side using only the adjacencies with rectangles from the maximum weight independent set.

By three-coloring the outerplanar graph, we see that the weight of the independent set is at least a third of the total weight since the maximum weight of a color class is a lower bound for the weight of the independent set. Applying the same argument to the other outerplanar graph, shows that we preserve at least a third of the remaining adjacencies.

Thus overall we keep at least 1/9 of the external adjacencies. Furthermore, the weight of the independent set for the first side is an upper bound on the number of external adjacencies that we can preserve. Since we preserve a third of this weight, our algorithm is a 1/3-approximation.

For more than two top-level regions the choices of which rectangles to place on a boundary are not independent. Instead of solving the problem for the boundaries between two regions we solve it for the line segments of the top-level subdivision (with possibly more than one region on each of the sides of the line segment). The choices of which rectangles to keep adjacent to line segments are also not independent, but if we only consider horizontal or only vertical line segments they are. By optimizing only for horizontal or only for vertical line segments, we again loose at most a factor of 1/2 in terms of the number of adjacencies preserved. Thus, overall we can preserve at least 1/18 of the external adjacencies and obtain a 1/6-approximation. We can compute a corresponding rectangular subdivision in linear time [13].

Theorem 6. *Given a rectilinear input layout, we can find a spatial treemap that respects the top-level adjacencies, preserves all internal bottom-level adjacencies, and preserves at least* 1/18 *of the external bottom-level adjacencies in* $O(n)$ *time.*

Corollary 1. *Given a rectilinear input layout, let* s *be the maximal number of external bottom-level adjacencies preserved in any spatial treemap that respects the top-level adjacencies and preserves all internal bottom-level adjacencies. We can find a spatial treemap that respects the top-level adjacencies, preserves all internal bottom-level adjacencies, and preserves at least* $s/6$ *external bottom-level adjacencies in* $O(n)$ *time.*

5 Discussion and Future Work

We provided the first systematic study of maintaining topological structure in multilevel treemaps. We focus on preserving adjacencies, and studied two variants, based on whether or not the orientations of these adjacencies should be preserved as well. Our main conclusion is that while in general, in contrast to rectangular cartograms (1-level treemaps), not all adjacencies can be preserved, it is possible to provide efficient algorithms that preserve the maximum number (in case of preserving orientations) or a constant factor of the maximum number (in case of not preserving orientations) of adjacencies.

We foresee several directions in which this research can be extended in the future. First of all, it may be possible to improve the running times and/or constants of our algorithms. In particular, in Section 3, it would be very interesting if the dependency on k can be improved: although we expect k to be much smaller than n in practice, the quartic dependency may still cause the first term to dominate the linear term. Also, while we show in Section 4 that constant-factor approximation algorithms exist, we do not expect the constants to be tight; new ideas seem to be necessary to obtain better bounds.

As a second direction of future research, it would be interesting to test our algorithms on real-world input data. The main objective of such a test is two-fold: an implementation would verify the practicality of our algorithms, but more importantly, it would allow us to

compare the resulting treemaps to treemaps produced by different methods with emphasis on other aspects than topology. A comparative study of how well adjacencies, relative positions, and/or distances are preserved by various methods would be very interesting.

Finally, we mention taking a broader perspective on maintaining topology in spatial treemaps as a direction of future work. Our current work is the first study of maintaining topology, but interprets the concept very strictly: two regions are either adjacent or not. It may be more useful to parameterize the concept, for instance by pairwise distance in the dual graph. Also, generalizing our ideas to treemaps with more than two levels would be interesting.

Acknowledgements

This research was initiated at the MARC 2009 workshop in Vught, the Netherlands. We would like to thank Jo Wood for proposing this problem, and all participants for sharing their thoughts on this subject.

D. E. is partially supported by the National Science Foundation (NSF) under grants 0830403 and CCF-1228639. D. E. and M. L. are partially supported by the U.S. Office of Naval Research under grant N00014-08-1-1015. M. L. is partially supported by the Netherlands Organisation for Scientific Research (NWO) under grant 639.021.123. K. B. is partially supported by the Netherlands Organisation for Scientific Research (NWO) under grant 612.001.207. M. N. is partially supported by the German Research Foundation (DFG) under grant NO 899/1-1. R. I. S. is partially supported by projects MINECO MTM2015-63791-R and Gen. Cat. DGR2014SGR46, and by MINECO through the Ramón y Cajal program.

References

[1] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

[2] S. Arnborg and A. Proskurowski. Linear time algorithms for NP-hard problems restricted to partial k-trees. *Discrete Appl. Math.*, 23(1):11–24, 1989.

[3] G. Birkhoff. Rings of sets. *Duke Math. J.*, 3(3):443–454, 1937.

[4] K. Buchin, D. Eppstein, M. Löffler, M. Nöllenburg, and R. I. Silveira. Adjacency-preserving spatial treemaps. In *Algorithms and Data Structures (WADS'11)*, volume 6844 of *LNCS*, pages 159–170. Springer Berlin Heidelberg, 2011.

[5] K. Buchin, B. Speckmann, and S. Verdonschot. Optimizing regular edge labelings. In *Graph Drawing (GD'10)*, volume 6502 of *LNCS*, pages 117–128. Springer Berlin Heidelberg, 2011.

[6] B. de Bono, P. Grenon, M. Helvensteijn, J. Kok, and N. Kokash. ApiNATOMY: Towards multiscale views of human anatomy. In *Advances in Intelligent Data Analysis XIII*, volume 8819 of *LNCS*, pages 72–83. Springer, 2014.

[7] F. S. L. G. Duarte, F. Sikansi, F. M. Fatore, S. G. Fadel, and F. V. Paulovich. Nmap: A novel neighborhood preservation space-filling algorithm. *IEEE Trans. Vis. Comput. Graph.*, 20(12):2063–2071, 2014.

[8] D. Eppstein, E. Mumford, B. Speckmann, and K. Verbeek. Area-universal and constrained rectangular layouts. *SIAM J. Computing*, 41(3):537–564, 2012.

[9] D. Eppstein, M. van Kreveld, B. Speckmann, and F. Staals. Improved grid map layout by point set matching. In *Pacific Visualization Symposium (PacificVis'13)*, pages 25–32. IEEE, 2013.

[10] É. Fusy. Transversal structures on triangulations: A combinatorial study and straight-line drawings. *Discrete Math.*, 309(7):1870–1894, 2009.

[11] M. Ghoniem, M. Cornil, B. Broeksema, M. Stefas, and B. Otjacques. Weighted maps: treemap visualization of geolocated quantitative data. In *IS&T/SPIE Electronic Imaging*, volume 9397, pages 93970G:1–15. International Society for Optics and Photonics, 2015.

[12] R. Heilmann, D. A. Keim, C. Panse, and M. Sips. Recmap: Rectangular map approximations. In *Information Visualization (INFOVIS'04)*, pages 33–40. IEEE, 2004.

[13] G. Kant and X. He. Two algorithms for finding rectangular duals of planar graphs. In *Graph-Theoretic Concepts in Computer Science (WG'93)*, volume 790 of *LNCS*, pages 396–410. Springer Berlin Heidelberg, 1993.

[14] G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theor. Comput. Sci.*, 172(1–2):175–193, 1997.

[15] N. Kokash, B. de Bono, and J. Kok. Template-based treemaps to preserve spatial constraints. In *Information Visualization Theory and Applications, (IVAPP'14)*, pages 39–49. SciTePress, 2014.

[16] K. Koźmiński and E. Kinnen. Rectangular duals of planar graphs. *Networks*, 15(2):145–157, 1985.

[17] F. Mansmann, D. A. Keim, S. C. North, B. Rexroad, and D. Sheleheda. Visual analysis of network traffic for resource planning, interactive monitoring, and interpretation of security threats. *IEEE Trans. Vis. Comput. Graph.*, 13(6):1105–1112, 2007.

[18] W. Mulzer and G. Rote. Minimum-weight triangulation is NP-hard. *J. ACM*, 55(2):11:1–11:29, May 2008.

[19] E. Raisz. The rectangular statistical cartogram. *Geogr. Rev.*, 24(2):292–296, 1934.

[20] B. Shneiderman. Tree visualization with tree-maps: 2-d space-filling approach. *ACM Trans. Graph.*, 11(1):92–99, 1992.

[21] B. Shneiderman, M. Lima, and J. Talasek. Every algorithm has art in it: Treemap art project, 2014. Exhibit Catalog, National Academy of Sciences, Washington, DC.

[22] A. Slingsby, J. Dykes, and J. Wood. Configuring hierarchical layouts to address research questions. *IEEE Trans. Vis. Comput. Graph.*, 15(6):977–984, 2009.

[23] A. Slingsby, J. Dykes, and J. Wood. Rectangular hierarchical cartograms for socio-economic data. *J. Maps*, 6(1):330–345, 2010.

[24] S. Tak and A. Cockburn. Enhanced spatial stability with hilbert and moore treemaps. *IEEE Trans. Vis. Comput. Graph.*, 19(1):141–148, 2013.

[25] M. van Kreveld and B. Speckmann. On rectangular cartograms. *Comput. Geom. Theory Appl.*, 37(3):175–187, 2007.

[26] J. Wood and J. Dykes. Spatially ordered treemaps. *IEEE Trans. Vis. Comput. Graph.*, 14(6):1348–1355, 2008.