On Rectilinear Duals for Vertex-Weighted Plane Graphs

Mark de Berg*, Elena Mumford, and Bettina Speckmann

Department of Mathematics & Computer Science, TU Eindhoven, The Netherlands
{mdberg, speckman}@win.tue.nl, e.mumford@tue.nl

Abstract. Let $\mathcal{G} = (V, E)$ be a plane triangulated graph where each vertex is assigned a positive weight. A rectilinear dual of $\mathcal{G}$ is a partition of a rectangle into $|V|$ simple rectilinear regions, one for each vertex, such that two regions are adjacent if and only if the corresponding vertices are connected by an edge in E. A rectilinear dual is called a cartogram if the area of each region is equal to the weight of the corresponding vertex. We show that every vertex-weighted plane triangulated graph $\mathcal{G}$ admits a cartogram of constant complexity, that is, a cartogram where the number of vertices of each region is constant.

1 Introduction

Motivation. Cartographers have developed many different techniques to visualize statistical data about a set of regions like countries, states or counties. *Cartograms* are among the most well known and widely used of these techniques. The regions of a cartogram are deformed such that the area of a region corresponds to a particular geographic variable [4]. The most common variable is population: In a population cartogram, the areas of the regions are proportional to their population. There are several types of cartograms. Of particular relevance for this paper are the *rectangular cartograms* introduced by Raisz in 1934 [12], where each region is represented by a rectangle. This has the advantage that the areas (and thereby the associated values) of the regions can be easily estimated by visual inspection.

Whether a cartogram is good is determined by several factors. In this paper we focus on two important criteria, namely the correct adjacencies of the regions of the cartogram and the *cartographic error* [5]. The first criterion requires that the dual graph of the cartogram is the same as the dual graph of the original map. Here the *dual graph* of a map—also referred to as *adjacency graph*—is the graph that has one node per region and connects two regions if they are adjacent, where two regions are considered to be adjacent if they share a 1-dimensional part of their boundaries (see Fig. 1). The second criterion, the cartographic error, is defined for each region as $|A_c - A_s| / A_s$, where A_c is the area of the region in the cartogram and A_s is the specified area of that region, given by the geographic variable to be shown.

* Supported by the Netherlands' Organisation for Scientific Research (NWO) under project no. 639.023.301.

P. Healy and N.S. Nikolov (Eds.): GD 2005, LNCS 3843, pp. 61–72, 2005.

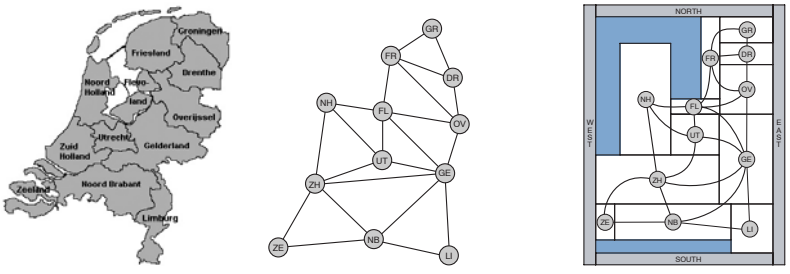

Fig. 1. The provinces of the Netherlands, their adjacency graph, a population cartogram—here additional "sea rectangles" were added to preserve the outer shape

From a graph-theoretic point of view constructing rectangular cartograms with correct adjacencies and zero cartographic error translates to the following problem. We are given a plane graph $\mathcal{G} = (V, E)$ (the dual graph of the original map) and a positive weight for each vertex (the required area of the region for that vertex). Then we want to construct a partition of a rectangle into rectangular regions whose dual graph is $\mathcal{G}$—such a partition is called a *rectangular dual* of $\mathcal{G}$—and where the area of each region is the weight of the corresponding vertex. As usual, we assume the input graph $\mathcal{G}$ is plane and triangulated, except possibly the outer face; this means that the original map did not have four or more countries whose boundaries share a common point and that $\mathcal{G}$ does not have degree-2 nodes.[1]

Unfortunately not every vertex-weighted plane triangulated graph admits a rectangular cartogram, even if we ignore the vertex weights and concentrate only on the correct adjacencies. There are several possibilities to address this problem. One is to relax the strict requirements on the adjacencies and areas. For example, Van Kreveld and Speckmann [14] gave an algorithm that constructs rectangular cartograms that in practice have only a small cartographic error and mild disturbances of the adjacencies. Heilmann et al. [6] gave an algorithm that always produces regions with the correct areas; unfortunately the adjacencies can be disturbed badly. The other extreme is to ignore the area constraints and focus only on getting the correct adjacencies—that is, to focus on rectangular duals rather than cartograms. This setting is relevant for computing floor plans in VLSI design. As mentioned above, ignoring the area constraints still does not guarantee that a solution exists. But, if the input graph is a triangulated plane graph without separating triangles—a separating triangle is a 3-cycle with vertices both inside and outside the cycle—then a rectangular dual always exists [1, 8] and can be computed in linear time [7].

Another option is to use different shapes for the regions. We restrict our attention to so-called *rectilinear cartograms*, which use rectilinear polygons as regions—see [10, 4] for some examples from the cartography community. If we

[1] Degree-2 nodes can easily be handled using suitable pre- and postprocessing steps [14].

now ignore the area requirement then things become much better: Any plane triangulated graph admits a rectilinear dual. In fact, Liao et al. [9] recently showed that any plane triangulated graph admits a rectilinear dual with regions of small complexity, namely rectangles, L-shapes, and T-shapes. The main questions now are: Does any plane triangulated vertex-weighted graph admit a rectilinear cartogram with zero cartographic error and correct adjacencies? And if so, can it always be done with a constant number of vertices per region?

This problem was studied by Rahman et al. [11] for a very special class of graphs, namely a certain subclass of graphs that admit a sliceable dual—see below. They showed that such graphs admit a rectilinear cartogram where every region has at most 8 vertices. Biedl and Genc [2] showed that it is NP-hard to decide if a rectilinear cartogram that uses regions with at most 8 vertices exists for a given graph. Furthermore, a rectangular layout can be interpreted as a plane, cubic graph. Thomassen showed [13] that any such graph can be drawn with straight (but not necessarily horizontal or vertical) edges such that every bounded face has any prescribed area. These results leave the two questions stated above still unanswered. Our paper answers them: We prove that any plane triangulated vertex-weighted graph admits a rectilinear cartogram all of whose regions have constant complexity. Before we describe our results in more detail we first define the terminology we use more precisely.

Terminology. A *layout* $\mathcal{L}$ is a partition of a rectangle R into a finite set of interior-disjoint regions. We consider only *rectilinear layouts*, where every region is a simple rectilinear polygon whose sides are parallel to the edges of R. We define the complexity of a rectilinear polygon as the total number of its vertices and the complexity of a rectilinear layout as the maximum complexity of any of its regions. A rectilinear layout is called *rectangular* if all its regions are rectangles. Thus, a rectangular layout is a rectilinear layout of complexity 4. Finally, a rectangular layout is called *sliceable* if it can be obtained by recursively slicing a rectangle by horizontal and vertical lines, which we call *slice lines*. (In computational geometry, such a recursive subdivision is called a (rectilinear) *binary space partition*, or *BSP* for short.)

We denote the dual graph (also called connectivity graph) of a layout $\mathcal{L}$ by $\mathcal{G}(\mathcal{L})$. Given a graph $\mathcal{G}$, a layout $\mathcal{L}$ such that $\mathcal{G} = \mathcal{G}(\mathcal{L})$ is called a *dual layout* (or simply a *dual*) for $\mathcal{G}$. The dual $\mathcal{G}(\mathcal{L})$ is unique for any layout $\mathcal{L}$. Note that not every graph $\mathcal{G}$ has a dual layout. If it does, then the dual layout is not necessarily unique.

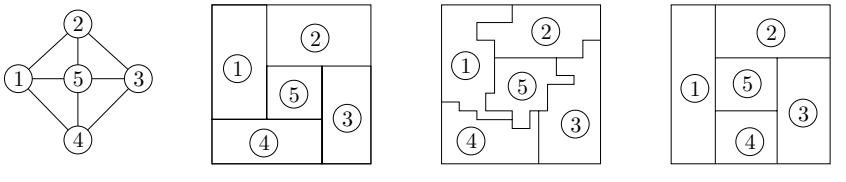

Fig. 2. A graph $\mathcal{G}$ with a rectangular, rectilinear, and sliceable dual

Every vertex v of a vertex-weighted graph $\mathcal{G}$ has a positive weight $w(v)$ associated with it. Given a vertex-weighted plane graph $\mathcal{G}$ that admits a dual $\mathcal{L}$, we say that $\mathcal{L}$ is a *cartogram* if the area of each region of $\mathcal{L}$ is equal to the weight of the corresponding vertex of $\mathcal{G}$. The cartogram is called *rectangular (rectilinear, sliceable)* if the corresponding layout is rectangular (rectilinear, sliceable).

Results. In Section 2 we show how to construct a cartogram of complexity 12 for any vertex-weighted plane triangulated graph that has a sliceable dual. We extend our results in Section 3 to general vertex-weighted plane triangulated graphs $\mathcal{G}$. Specifically, if $\mathcal{G}$ admits a rectangular dual then we can construct a cartogram of complexity at most 20, otherwise we can construct a cartogram of complexity at most 60. In Section 4 we conclude with several open problems.

2 Graphs That Admit a Sliceable Dual

Let $\mathcal{G} = (V, E)$ be a vertex-weighted plane triangulated graph with n vertices that admits a sliceable dual. The exact characterization of such graphs is still unknown, but Yeap and Sarrafzadeh [15] proved that every triangulated plane graph without separating triangles and without separating 4-cycles has a sliceable dual. W.l.o.g. we assume that the vertex weights of $\mathcal{G}$ sum to 1, and that the rectangle R that we want to partition is the unit square.

Let $\mathcal{L}_1$ be a sliceable dual for $\mathcal{G}$. We scale and stretch $\mathcal{L}_1$ such that it becomes a partition of the unit square R. We will transform $\mathcal{L}_1$ into a cartogram for $\mathcal{G}$ in three steps. In the first step we transform $\mathcal{L}_1$ into a layout $\mathcal{L}_2$ where every region has the correct area. In doing so, however, we may loose some of the adjacencies, that is, $\mathcal{L}_2$ may no longer be a dual layout for $\mathcal{G}$. This is remedied in the second step, where we transform $\mathcal{L}_2$ into a layout $\mathcal{L}_3$ whose dual is $\mathcal{G}$. In this step we re-introduce some errors in the areas. But these errors are small, and we can remove them in the third step, which produces the final cartogram, $\mathcal{L}_4$. Below we describe each of these steps in more detail.

Step 1: Setting the Areas Right
The first step is relatively easy. Recall that a sliceable layout is a recursive partition of R into rectangles by vertical and horizontal slice lines. This recursive partition can be modelled as a BSP tree $\mathcal{T}$. Each node ν of $\mathcal{T}$ corresponds to a rectangle $R(\nu) \subseteq R$ and the interior nodes store a slice line $\ell(\nu)$. The rectangles $R(\nu)$ are defined recursively, as follows. We have $R(\text{root}(\mathcal{T})) = R$. Furthermore, $R(\text{leftchild}(\nu)) = R(\nu) \cap \ell^-(\nu)$ and $R(\text{rightchild}(\nu)) = R(\nu) \cap \ell^+(\nu)$, where $\ell^-(\nu)$ and $\ell^+(\nu)$ denote the half-space to the left and right of $\ell(\nu)$ (or, if $\ell(\nu)$ is horizontal, above and below $\ell(\nu)$). The rectangles $R(\nu)$ corresponding to the leaves are precisely the regions of the sliceable layout. See for example Figure 3—the shaded rectangle corresponds to the shaded node. The BSP tree for a sliceable layout is not necessarily unique, because different recursive partition processes may lead to the same layout.

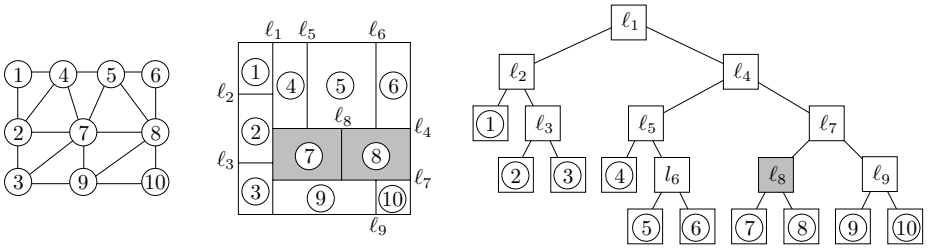

Fig. 3. A graph $\mathcal{G}$, the layout $\mathcal{L}_1$, and the BSP tree $\mathcal{T}$

The point where two or maximally three slice lines meet is called a *junction (point)*. We distinguish between T- and X-junctions. A T-junction involves two slice lines while an X-junction involves three slice lines, two of which are aligned.

Now, let $\mathcal{T}$ be a BSP tree that models the sliceable layout $\mathcal{L}_1$. We will transform $\mathcal{L}_1$ into $\mathcal{L}_2$ by changing the coordinates of the slice lines used by $\mathcal{T}$ in a top-down manner. We maintain the following invariant: When we arrive at a node ν in $\mathcal{T}$, the area of $R(\nu)$ is equal to the sum of the required areas of the regions represented by the leaves below ν. Clearly this is true when we start the procedure at the root of $\mathcal{T}$. Now assume that we arrive at a node ν which stores a slice line $\ell(\nu)$. We simply sum up all the required areas in the left subtree of ν and adjust the position of the $\ell(\nu)$ in the unique way that assigns the correct areas to $R(\text{leftchild}(\nu))$ and $R(\text{rightchild}(\nu))$. When we reach a leaf there is nothing to do; the rectangle it represents now has the required area.

Step 2: Setting the Adjacencies Right

The movement of the slice lines in Step 1 may have changed the adjacencies between the regions. To remedy this, we will use the BSP tree $\mathcal{T}$ again.

Before we start, we define two strips for each slice line $\ell(\nu)$. These strips are centered around $\ell(\nu)$ and are called the *tail strip* and the *shift strip*. The width of the tail strip is $2\varepsilon_\nu$ and the width of the shift strip is $2\delta_\nu$, where $\varepsilon_\nu < \delta_\nu$ and ε_ν and δ_ν are sufficiently small. The exact values of ε_ν and δ_ν will be specified in Step 3. At this point it is relevant only that we can choose them in such a way that the shift strips of two slice lines are disjoint except when two slice lines meet.

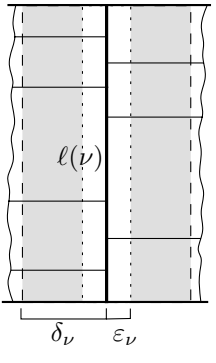

Fig. 4. The shift and tail strips for ℓ_i

We will make sure that the changes to the layout in Step 2 all occur within the tail strips and that the changes in Step 3 all occur within the shift strips. Due to the choice of the δ_ν's all the junction points within the shift strip will lie on the slice line $\ell(\nu)$.

To restore the correct adjacencies, we traverse the BSP tree bottom-up. We maintain the invariant that after handling a node ν, all adjacencies between regions inside $R(\nu)$ have been restored. Now suppose that we reach a node ν. The invariant tells us that all

adjacencies inside $R(\text{leftchild}(\nu))$ and $R(\text{rightchild}(\nu))$ have been restored. It remains to restore the correct adjacencies between regions on different sides of the slice line $\ell(\nu)$. We will describe how to restore the adjacencies for the case where $\ell(\nu)$ is vertical; horizontal slice lines are handled in a similar fashion, with the roles of the x- and y-coordinates exchanged.

Let $A_1, A_2, \ldots, A_k$ be the set of regions inside $R(\nu)$ bordering $\ell(\nu)$ from the left, and let $B_1, B_2, \ldots, B_m$ be the set of regions inside $R(\nu)$ bordering $\ell(\nu)$ from the right. Both the A_i's and the B_j's are numbered from top to bottom—see Figure 5. We write $A_i \prec A_j$ to indicate that A_i is above A_j; thus $A_i \prec A_j$ if and only if $i < j$. The same notation is used for the B_j's. Now consider the tail strip centered around $\ell(\nu)$. All slice lines ending on $\ell(\nu)$ are straight lines within the tail strip (and, in fact, even within the shift strip). This is true before Step 2, but as we argue later, it is still true when we start to process $\ell(\nu)$.

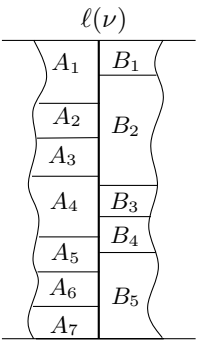

Fig. 5. Left and right neighbors

In Step 1 (and when Step 2 was applied to $R(\text{leftchild}(\nu))$ and $R(\text{rightchild}(\nu))$), the slice lines separating the A_i's from each other and the slice lines separating the B_j's from each other may have shifted, thus disturbing the adjacencies between the A_i's and B_j's. For each A_i, we define $\text{top}(A_i) := B_k$ if B_k is the highest region (among the B_j's) adjacent to A_i in the original layout $\mathcal{L}_1$. Similarly, $\text{bottom}(A_i)$ is the lowest such region. This means that in $\mathcal{L}_1$, the region A_i was adjacent to all B_j with $\text{top}(A_i) \preceq B_j \preceq \text{bottom}(A_i)$. We restore these adjacencies for A_i by adding at most two so-called *tails* to A_i, as described below. This is done from top to bottom: We first handle A_1, then A_2, and so on. During this process the slice line $\ell(\nu)$ will be deformed—it will no longer be a straight line, but it will become a rectilinear poly-line. However, the part of $\ell(\nu)$ bordering regions we still have to handle will be straight. More precisely, we maintain the following invariant: When we start to handle a region A_i, the part of $\ell(\nu)$ that lies below the bottom edge of $\text{top}(A_i)$ is straight and the right borders of all $A_j \succeq A_i$ are collinear with that part of ℓ.

Next we describe how A_i is handled. There are two cases, which are not mutually exclusive: Zero, one, or both of them may apply. When both cases apply, we treat first (a) and then (b).

(a) If A_i is not adjacent to $\text{top}(A_i)$ and $\text{top}(A_i)$ is higher than A_i, then we add a tail from A_i to $\text{top}(A_i)$. (If A_i is not adjacent to $\text{top}(A_i)$ and $\text{top}(A_i)$ is lower than A_i, then case (b) will automatically connect A_i to $\text{top}(A_i)$.) More precisely, we add a rectangle to the right of A_i whose bottom edge is collinear with the bottom edge of A_i and whose top edge is contained in the bottom edge of $\text{top}(A_i)$. The width of this rectangle is $\frac{\varepsilon_\nu}{n}$. Moreover, we shift the part of the slice line below $\text{top}(A_i)$ by $\frac{\varepsilon_\nu}{n}$ to the right. Observe that this will make all the B_j below $\text{top}(A_i)$ smaller and all A_j below A_i larger.

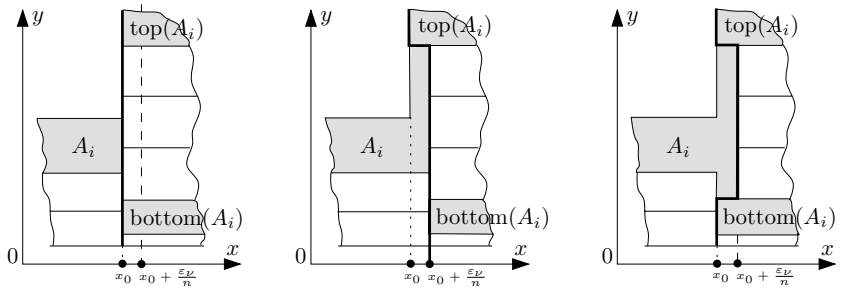

Fig. 6. Both case (a) and case (b) apply

(b) If A_i is not adjacent to bottom(A_i) and bottom(A_i) is lower than A_i, then we also add a tail, as follows. (If A_i was not adjacent to bottom(A_i) and bottom(A_i) was higher than A_i, then necessarily case (a) has already been treated and in fact A_i is now adjacent to bottom(A_i).) First, we shift the part of the slice line below the top edge of bottom(A_i) by $\frac{\varepsilon_\nu}{n}$ to the left. Observe that this will enlarge bottom(A_i) and all the B_j below it, and make all $A_j \succ A_i$ smaller. Next, we add a rectangle of width $\frac{\varepsilon_\nu}{n}$ to A_i, which connects A_i to bottom(A_i). Its top edge is contained in the bottom edge of A_i, its right edge is collinear to A_i's right edge, and its bottom edge is contained in the top edge of bottom(A_i).

Note that every tail "ends" on some B_j, that is, no tail extends all the way to the slice lines on which $\ell(\nu)$ ends. This implies that

- no bends are introduced inside the shift strips of the two slice lines on which $\ell(\nu)$ ends (as we already claimed earlier).
- the bordering sequence (the sets of countries along each side of a slice line and their order) of any other slice line remains unchanged.
- the bottom end of $\ell(\nu)$ shifts only within the tail strip of $\ell(\nu)$.

Lemma 1. *The layout $\mathcal{L}_3$ obtained after Step 2 has the following properties:*

(i) If two regions are adjacent in $\mathcal{L}_1$, then they are also adjacent in $\mathcal{L}_3$.
(ii) The tails that are added when handling a slice line ℓ all lie within the tail strip of ℓ.
(iii) Each region gets at most three tails.

Proof.

(i) It follows from the construction that each region A_i along a slice line $\ell(\nu)$ has the required adjacencies after $\ell(\nu)$ has been handled. Hence, the construction maintains the invariant that all adjacencies within $R(\nu)$ are restored after $\ell(\nu)$ has been handled. Therefore, after the slice line that is stored at the root of $\mathcal{T}$ is handled, all adjacencies have been restored.

(ii) A tail inside a tail strip of width $2\varepsilon_\nu$ has width $\frac{\varepsilon_\nu}{n}$ and is always adjacent to the current slice line. A slice line is shifted at most $n-2$ times by $\frac{\varepsilon_\nu}{n}$. Hence, the tails lie within the tail strip, as claimed.

(iii) A region can get tails only when the slice line ℓ_r on its right and the slice line ℓ_t along its top are handled. Since a region must be either the topmost region along ℓ_r or the rightmost region along ℓ_t it can only get a double tail along one of these slice lines. Thus each region receives at most 3 tails. Note that since the tails along the same slice line are aligned, a region does not get more than three concave vertices. □

Note that if $\mathcal{G}$ is triangulated then Lemma 1 (i) implies that two regions in $\mathcal{L}_3$ are adjacent if and only if they are adjacent in $\mathcal{L}_1$: All required adjacencies are present and in a plane triangulated graph there is no room for additional adjacencies.

Step 3: Repairing the Areas

When we repaired the adjacencies in Step 2, we re-introduced some small errors in the areas of the regions. We now set out to remedy this. In Step 2, the slice lines actually became rectilinear poly-lines. These poly-lines, which we will keep on calling slice lines for convenience, are monotone: A horizontal (resp. vertical) line intersects any vertical (resp. horizontal) slice line in a single point, a segment, or not at all. We will repair the areas by moving the slice lines in a top-down manner, similar to Step 1. But because we do not want to loose any adjacencies again, we have to be more careful in how we exactly move a slice line. This is described next.

Assume that we wish to move a horizontal slice line $\ell = \ell_\nu$; vertical slice lines are treated in a similar manner. Let ℓ_1 and ℓ_2 be the slice lines to the left and to the right of ℓ, that is, the slice lines on which ℓ ends. We define a so-called *container* for ℓ, denoted by $C(\ell)$. The container $C(\ell)$ is a rectangle containing most of ℓ, as well as parts of the other slice lines ending on ℓ. Instead of moving the slice line ℓ we will move the container $C(\ell)$ and its complete contents.

We first define the container $C(\ell)$ more precisely. The top and bottom edge of $C(\ell)$ are contained in the boundary of the tail strip of ℓ. The position of the right edge of ℓ is determined by what happened at the junction between ℓ and ℓ_2 when ℓ_2 was processed during Step 2. Let A_i and A_{i+1} be the regions above and below ℓ and bordering ℓ_2.

(i) *A_i did not get a downward tail and A_{i+1} did not get an upward tail.*
In this case either there is no other junction on ℓ_2 within ℓ's shift strip, or there is exactly one and it lies within ℓ's tail strip (see Fig. 7(a)). If there is a junction on ℓ_2 within ℓ's tail strip in the direction in which $C(\ell)$ should be moved, then we set the right edge of the container $C(\ell)$ at distance ε_ν/n from ℓ_2 (see Fig. 7(b)). Otherwise, the right edge of the container is collinear with the part of ℓ_2 lying within ℓ's shift strip (see Fig. 7(c)).

(ii) *A_i got a downward tail or A_{i+1} got an upward tail.*
Note that in this case more tails may have entered the tail strip of ℓ. For example, if A_{i+1} got an upward tail then some other regions below A_{i+1} possibly got an upward tail as well. In this case the right edge of $C(\ell)$ will go through the leftmost such tail edge—see Fig. 8. Figures 9 and 10 illustrate

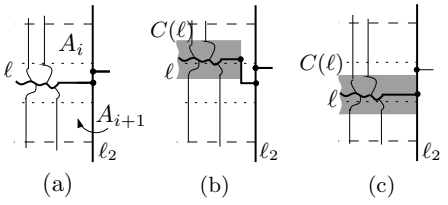

Fig. 7. (a) ℓ and ℓ_2 form a T-junction; (b) $C(\ell)$ is moved up and there is a junction in its way; (c) $C(\ell)$ is moved down and there is no junction in its way

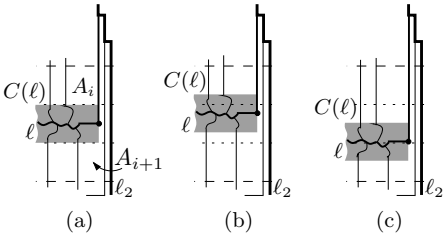

Fig. 8. (a) A_{i+1} has an upward tail; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down

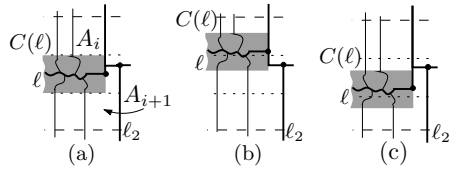

Fig. 9. (a) A_{i+1} got an upward tail with its end inside ℓ's tail strip; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down

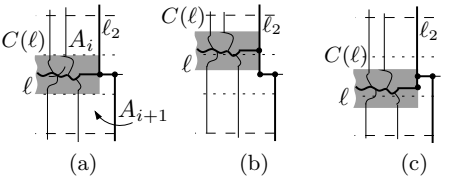

Fig. 10. (a) A_{i+1} got an upward tail of length 0; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down

the case, when ℓ and ℓ_2 were involved in an X-junction in $\mathcal{L}_1$—hence A_{i+1} could have a tail within ℓ's tail strip.

The position of the left edge of $C(\ell)$ is determined in a similar fashion, the details can be found in the full paper. Note that no matter what was going on on the other sides of ℓ_1 and ℓ_2, the adjacencies are preserved when $C(\ell)$ is moved.

Recall that we are repairing the areas in a top-down manner. When we get to slice line ℓ, we need to make sure that the total area above ℓ—or rather the total area of the regions corresponding to the left subtree of the node corresponding to ℓ in the BSP tree—is correct. We do this by moving the container $C(\ell)$. We will show below that the error we have to repair is so small that it can be repaired by moving $C(\ell)$ within the shift strip of ℓ. The parts of the slice lines ending on ℓ that are inside the shift strip and outside the tail strip are all straight segments; this follows from Lemma 1 (ii). Hence, when we move $C(\ell)$ we can simply shrink or stretch these segments, and the topology does not change. We first analyze what happens to the complexity of the regions when we move the containers.

Lemma 2. *After Step 3 a region gets at most 4 concave vertices in total.*

Proof. We might only "bend" a slice line ℓ, ending on slice lines ℓ_1 and ℓ_2, when moving its container $C(\ell)$. Thus we can introduce concave vertices to two regions adjacent to ℓ and ℓ_1 (ℓ_2), denoted above as B_j and B_{j+1} (A_i and A_{i+1}). It is easy to verify—see Figures 7–10—that a region can only get an extra concave

vertex at the junction of ℓ and ℓ_2 when the corner of the region did not yet get a tail in Step 2. The same is true for the junction of ℓ and ℓ_1. Hence the total number of concave vertices is bounded by four—at most one for each corner of the region in $\mathcal{L}_1$. □

It remains to prove that we can choose the widths of the tail strip and shift strip appropriately. The two properties that we require are as follows.

Requirement 1. *The shift strips of slice lines do not intersect if the slice lines do not intersect after Step 1.*

Requirement 2. *The shift strip of each slice line ℓ is wide enough so that, when handling ℓ in Step 3, moving the container $C(\ell)$ can repair the areas while staying within the shift strip.*

For the first requirement it is sufficient to take the width of the shift strip to be smaller than $\Delta/2$, where $\Delta := \min(\Delta_x, \Delta_y)$ and Δ_x (Δ_y) is the minimum difference between any two distinct x-coordinates (y-coordinates) of the vertical (horizontal) slice lines after Step 1.

As for the second requirement, we provide a very rough estimate of the values for the width of the shift and tails strips, just to show that suitable values exist. Number the slice lines $\ell_1, \ldots \ell_{n-1}$ in the same order in which we handle them. (For example, the slice line at the root of the BSP tree will be ℓ_1.)

Lemma 3. *If the width of the shift strip of slice line ℓ_k is set to $\delta_k := \Delta/4 \cdot ((\Delta(1-\Delta))/10)^{n-k-1}$ and the width of the tail strip is set to $\varepsilon_k := \delta_k \cdot \Delta/2$, for $1 \leqslant k \leqslant n-1$, then Requirements 1 and 2 are fulfilled.*

The proof of Lemma 3 can be found in the full version of the paper. We conclude this section with the following theorem:

Theorem 1. *Let $\mathcal{G}$ be a vertex-weighted plane triangulated graph that admits a sliceable dual. Then $\mathcal{G}$ admits a cartogram of complexity at most 12.*

3 General Graphs

In the previous section we described an algorithm to construct cartograms for graphs that admit a sliceable dual. Next we consider more general graphs, namely graphs that admit a rectangular dual and arbitrary triangulated plane graphs. These more general classes of graphs are handled by adding an extra step before the three steps described in the previous section.

We begin with graphs that admit a rectangular dual, that is, plane triangulated graphs without separating triangles. Such a rectangular dual can be constructed, for example, by the algorithm of Kant and He [7]. Let now $\mathcal{G}$ be a plane triangulated graph without separating triangles and $\mathcal{L}_0$ a rectangular dual of $\mathcal{G}$. We construct a rectilinear BSP on $\mathcal{L}_0$, that is, we recursively partition $\mathcal{L}_0$ using horizontal or vertical splitting lines until each cell in the partitioning intersects a single rectangle from $\mathcal{L}_0$. This can be done in such a way that each

rectangle in $\mathcal{L}_0$ is cut into at most four subrectangles [3]. The resulting layout of these subrectangles, $\mathcal{L}_1$, is sliceable by construction.

We then assign weights to the subrectangles. If a rectangle in $\mathcal{L}_0$ representing a vertex v of $\mathcal{G}$ was cut into k subrectangles in $\mathcal{L}_1$ then each subrectangle is assigned weight $w(v)/k$. (In practice it may be better to make the weight of each subrectangle proportional to its area.) Next, we perform Step 1–3 of the previous section on the layout $\mathcal{L}_1$ with these weights. Each rectilinear region in the layout $\mathcal{L}_4$ obtained after Step 3 corresponds to a subrectangle in $\mathcal{L}_1$. Finally, we merge the regions corresponding to subrectangles coming from the same rectangle in $\mathcal{L}_0$—and, hence, from the same vertex of $\mathcal{G}$—thus obtaining a layout $\mathcal{L}_5$ with one region per vertex of $\mathcal{G}$. The next lemma guarantees the correctness of our approach, its proof can be found in the full paper.

Lemma 4. *The algorithm described above produces a layout where each region has the correct area and adjacencies.*

It remains to analyze the complexity of the regions in the final layout. Of course we can just multiply the bound from the previous section by four, since each vertex in $\mathcal{G}$ is represented by four rectangles in $\mathcal{L}_1$. This results in a bound of 48. The next lemma shows that things are not quite that bad, its proof can be found in the full paper.

Lemma 5. *The algorithm described above produces regions of complexity at most 20.*

The next theorem summarizes our result for graphs that admit a rectangular dual.

Theorem 2. *Let $\mathcal{G}$ be a vertex-weighted plane triangulated graph that admits a rectangular dual, i.e., $\mathcal{G}$ has no separating triangles. Then $\mathcal{G}$ admits a cartogram of complexity at most 20.*

We now turn our attention to general plane triangulated graphs. As mentioned earlier, Liao et al. [9] showed that any plane triangulated graph has a rectilinear dual that uses L- and T-shapes—that is, regions of maximal complexity 8—in addition to rectangles. We cut each region into at most three subrectangles and then proceed as in the previous case: We cut the collection of subrectangle with a BSP to obtain a sliceable layout $\mathcal{L}_1$, we assign weights to the rectangles in $\mathcal{L}_1$, run Step 1–3, and merge regions belonging to the same vertex in $\mathcal{G}$. This immediately gives the following corollary.

Corollary 1. *Any vertex-weighted plane triangulated graph $\mathcal{G}$ admits a cartogram of complexity at most 60.*

4 Conclusions

We proved that every plane triangulated vertex-weighted graph admits a rectilinear cartogram of constant complexity. Currently, however, our method is not

practical. First of all, although the complexity of the cartogram is bounded by a constant, it is rather high. So interesting open problems are to give an algorithm that produces cartograms of smaller complexity and to give lower bounds on the minimum complexity required to guarantee the existence of a cartogram. It would also be useful to give an exact characterization of the graphs that admit a sliceable dual, since the bound we obtain for such graphs is much better. A second problem with our algorithm from a practical point of view is that the tails we add to get the correct adjacencies can be quite thin. It would be nice to see if it is possible to do with wider tails.

References

1. J. Bhasker and S. Sahni. A linear algorithm to check for the existence of a rectangular dual of a planar triangulated graph. *Networks*, 7:307–317, 1987.
2. T. Biedl and B. Genc. Complexity of octagonal and rectangular cartograms. In *Proceedings of the 17th Canadian Conference on Computational Geometry*, pages 117–120, 2005.
3. F. d'Amore and P. G. Franciosa. On the optimal binary plane partition for sets of isothetic rectangles. *Information Processing Letters*, 44(5):255–259, 1992.
4. B. Dent. *Cartography - thematic map design*. McGraw-Hill, 5th edition, 1999.
5. J. A. Dougenik, N. R. Chrisman, and D. R. Niemeyer. An algorithm to construct continous area cartograms. *Professional Geographer*, 37:75–81, 1985.
6. R. Heilmann, D. A. Keim, C. Panse, and M. Sips. Recmap: Rectangular map approximations. In *Proceedings of the IEEE Symposium on Information Visualization (INFOVIS)*, pages 33–40, 2004.
7. G. Kant and X. He. Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems. *Theoretical Computer Science*, 172:175–193, 1997.
8. K. Koźmiński and E. Kinnen. Rectangular dual of planar graphs. *Networks*, 5:145–157, 1985.
9. C.-C. Liao, H.-I. Lu, and H.-C. Yen. Floor-planning using orderly spanning trees. *Journal of Algorithms*, 48:441–451, 2003.
10. NCGIA / USGS. Cartogram Central, 2002. `http://www.ncgia.ucsb.edu/projects/Cartogram_Central/index.html`.
11. M. S. Rahman, K. Miura, and T. Nishizeki. Octagonal drawings of plane graphs with prescribed face areas. In *Proceedings of the 30th International Workshop on Graph-Theoretic Concepts in Computer Science (WG)*, number 3353 in LNCS, pages 320–331. Springer, 2004.
12. E. Raisz. The rectangular statistical cartogram. *Geographical Review*, 24:292–296, 1934.
13. C. Thomassen. Plane cubic graphs with prescribed face areas. *Combinatorics, Probability and Computing*, 1:371–381, 1992.
14. M. van Kreveld and B. Speckmann. On rectangular cartograms. *Computational Geometry: Theory and Applications*, 2005. To appear.
15. G. K. Yeap and M. Sarrafzadeh. Sliceable floorplanning by graph dualization. *SIAM Journal of Discrete Mathematics*, 8(2):258–280, 1995.