ARTICLE IN PRESS

Available online at www.sciencedirect.com

ScienceDirect

ELSEVIER

Discrete Mathematics ▮ (▮▮▮▮) ▮▮▮–▮▮▮

DISCRETE MATHEMATICS

www.elsevier.com/locate/disc

On rectilinear duals for vertex-weighted plane graphs

Mark de Berg[1], Elena Mumford, Bettina Speckmann*

Department of Mathematics and Computer Science, TU Eindhoven, P.O. Box 513, 5600 MB Eindhoven, The Netherlands

Received 14 February 2006; accepted 13 December 2007

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. Furthermore, such a rectilinear cartogram can be constructed in $O(n \log n)$ time where $n = |V|$.

Keywords: Cartogram; Rectilinear layout

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 [6]. 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 [14], 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* [7]. 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 one-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

* Corresponding author.
E-mail addresses: mdberg@win.tue.nl (M. de Berg), e.mumford@tue.nl (E. Mumford), speckman@win.tue.nl (B. Speckmann).
[1] Supported by the Netherlands' Organisation for Scientific Research (NWO) under project No. 639.023.301.

0012-365X/$ - see front matter © 2007 Elsevier B.V. All rights reserved.
doi:10.1016/j.disc.2007.12.087

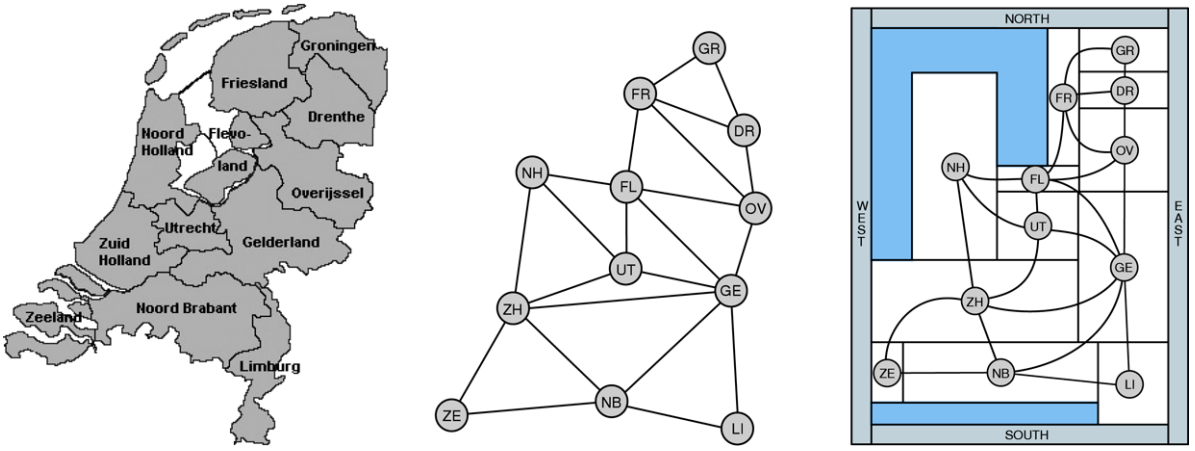

Fig. 1. The provinces of the Netherlands, their adjacency graph, a population cartogram—here additional "sea rectangles" were added to preserve the outer shape.

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.

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 that 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.[2]

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. The graph in Fig. 2 (left), for instance, does not have a rectangular dual. The graph in the middle of Fig. 2 does have a rectangular dual (Fig. 2 (right)) but if, for example, the weight of vertex 1 and 3 is 10 and the weight of vertex 2 and 4 is 100, then no rectangular cartogram with correct adjacencies and zero cartographic error exists.

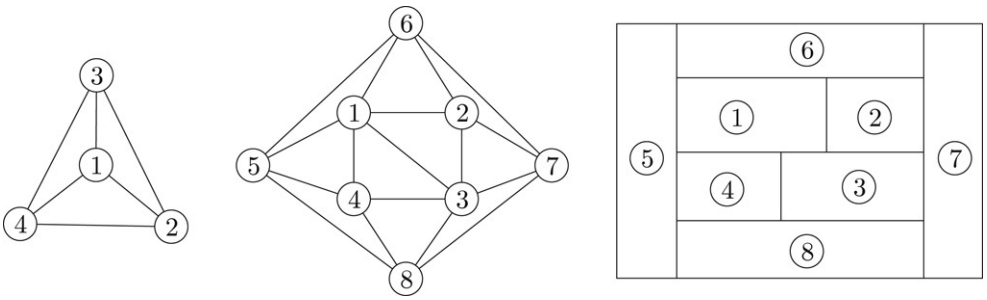

Fig. 2. No rectangular dual (left); the graph in the middle does have a rectangular dual (right) but for certain weights no rectangular cartogram can be constructed.

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 [11] 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. [8] gave an algorithm that always produces regions with the correct areas; unfortunately the adjacencies can be disturbed badly.

[2] Degree-2 nodes can easily be handled using suitable pre- and post-processing steps [11].

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

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 with four vertices on the outer face and without separating triangles – a separating triangle is a 3-cycle with vertices both inside and outside the cycle – then a rectangular dual always exists [1,10] and can be computed in linear time [9].

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 [5,6] for some examples from the cartography community. If we now ignore the area requirement then things become much better: Any plane triangulated graph admits a rectilinear dual. In fact, Liao et al. [12] 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 admits 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. [13] for a very special class of graphs, namely graphs that admit a sliceable dual – that is, a rectangular dual that can be obtained by recursively partitioning a rectangle by horizontal and vertical lines – with the additional property that "either the upper subrectangle or the lower one obtained by any horizontal slice will never be vertically sliced" [13]. They showed that by fixing the positions of some of the corners of the rectangles of such a layout one can give the regions correct areas by bending the edges of the rectangles. Each region in the resulting cartogram 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—with T-junctions as vertices. Thomassen showed [15] 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.)

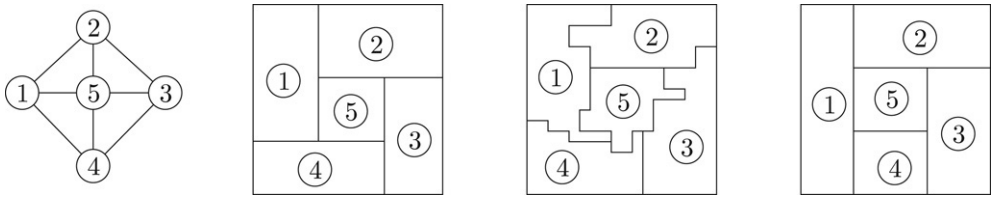

Fig. 3. A graph $\mathcal{G}$ with an rectangular, rectilinear, and sliceable dual.

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 $\mathcal{G}$. $\mathcal{G}(\mathcal{L})$ is unique for any layout $\mathcal{L}$. Though note that not every graph $\mathcal{G}$ has a dual layout. If it does, then the dual layout is not necessarily unique.

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).

A k-cycle in a plane graph that has vertices both inside and outside of the cycle is called *separating*. A separating 3-cycle is called a *separating triangle*.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

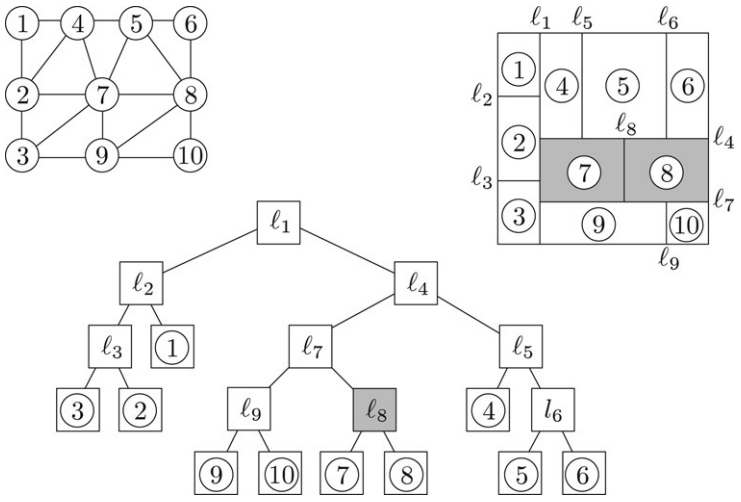

Fig. 4. A graph $\mathcal{G}$, the layout $\mathcal{L}_1$, and the BSP tree $\mathcal{T}$.

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 still construct a cartogram of complexity at most 40. In Section 4 we analyze the running time of our algorithm and in Section 5 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 [16] proved that every triangulated plane graph without separating triangles and without separating 4-cycles has a sliceable dual, which can be constructed in $O(n^2)$ time. 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—Fig. 3 depicts an example of a graph $\mathcal{G}$ and its sliceable dual $\mathcal{L}_1$. 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 $\mathcal{L}_2$ – see Fig. 5 – 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 $\mathcal{G}$—for instance, the regions 2 and 7 in Fig. 5 are not adjacent anymore. This is remedied in the second step, where we transform $\mathcal{L}_2$ into a layout $\mathcal{L}_3$ – see Fig. 6 for an example – 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, $\mathcal{L}_4$—see Fig. 7. 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 additionally 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, below and above $\ell(\nu)$). The rectangles $R(\nu)$ corresponding to the leaves are precisely the regions of the sliceable layout. See for example Fig. 4 —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.

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

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

aligned. Note that our initial layout has T-junctions only. However, X-junctions might appear later when the layout goes through further steps of our algorithm, including the step we are describing in this section.

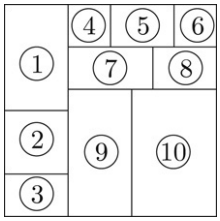

Fig. 5. Layout $\mathcal{L}_2$.

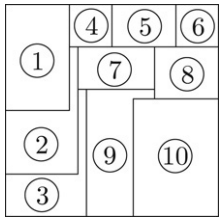

Fig. 6. Layout $\mathcal{L}_3$.

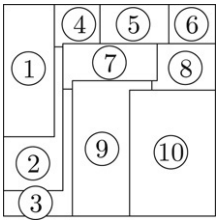

Fig. 7. Layout $\mathcal{L}_4$.

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 v in $\mathcal{T}$, the area of $R(v)$ is equal to the sum of the required areas of the regions represented by the leaves below v. Clearly this is true when we start the procedure at the root of $\mathcal{T}$. Now assume that we arrive at a node v which stores a slice line $\ell(v)$. We simply sum up all the required areas in the left subtree of v and adjust the position of the $\ell(v)$ in the unique way that assigns the correct areas to $R(\text{leftchild}(v))$ and $R(\text{rightchild}(v))$. When we reach a leaf there is nothing to do; the rectangle it represents now has the required area. See, for example, Fig. 5 that shows the layout $\mathcal{L}_2$ for the example in Fig. 4 and the weights $[w(1), \ldots, w(10)] = [0.15, 0.09, 0.06, 0.04, 0.06, 0.04, 0.08, 0.06, 0.18, 0.24]$.

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 $\ell(v)$. These strips are centered around $\ell(v)$ and are called the *tail strip* and the *shift strip* (see Fig. 8). The width of the tail strip is $2\varepsilon_v$ and the width of the shift strip is $2\delta_v$, where $\varepsilon_v < \delta_v$ and ε_v and δ_v are sufficiently small. The exact values of ε_v and δ_v 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.

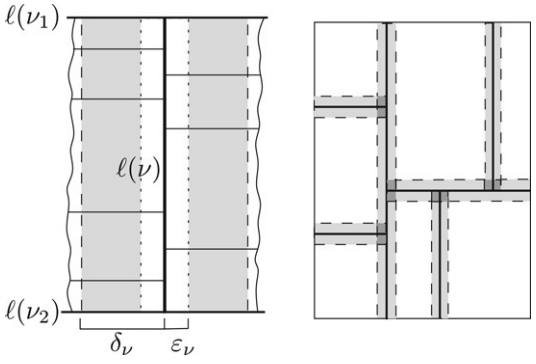

Fig. 8. The shift and tail strips for ℓ_v (left), ℓ_v has two external junctions where it meets ℓ_{v_1} and ℓ_{v_2}, all other junctions are internal; the intersection pattern of shift-strips (right).

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_v's all the junction points within the shift strip will lie on the slice line $\ell(v)$.

To restore the correct adjacencies, we traverse the BSP tree bottom-up. We maintain the invariant that after handling a node v, all adjacencies between regions inside $R(v)$ have been restored. Now suppose that we reach a node v. The invariant tells us that all adjacencies inside $R(\text{leftchild}(v))$ and $R(\text{rightchild}(v))$ have been restored. It remains to restore the correct adjacencies between regions on different sides of the slice line $\ell(v)$. We will describe how to restore the adjacencies for the case where $\ell(v)$ is vertical; horizontal slice lines are handled in a similar fashion, with the roles of the x- and y-coordinates exchanged.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

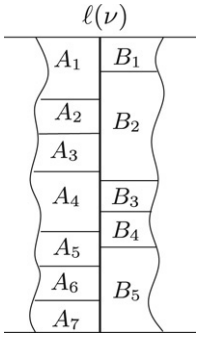

Fig. 9. Left and right neighbors.

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 Fig. 9. 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 $\ell(\nu)$.

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 $\ell(\nu)$.

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 first treat (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 in $\mathcal{L}_2$ (that is, the layout after Step 1 before Step 2), 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 $\ell(\nu)$ 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—see the second picture in Fig. 10.

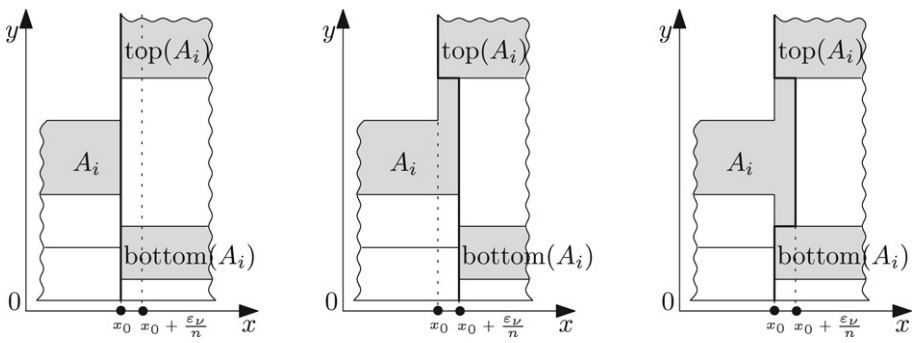

Fig. 10. Both case (a) and case (b) apply.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

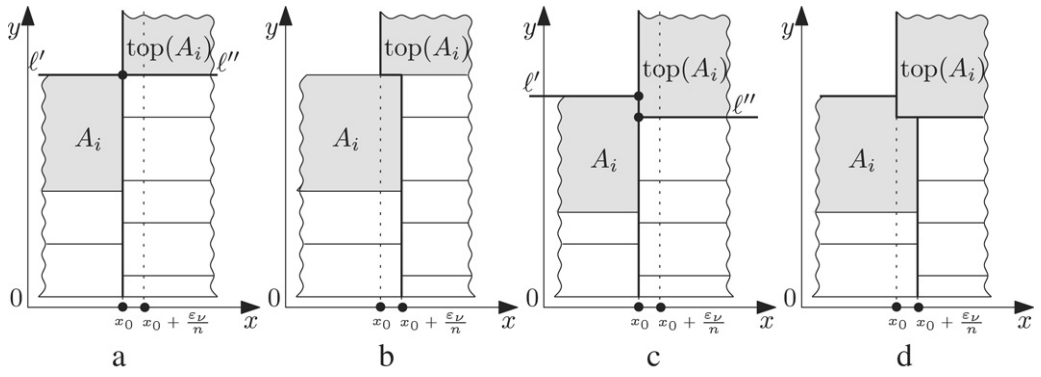

Fig. 11. (a), (b) Zero-tail up; (c), (d) negative tail up.

Note, that the tail can be of positive (Fig. 10), zero (Fig. 11(a)–(b)) or negative (Fig. 11(c)–(d)) length. The zero-tail occurs when the line along A_i's north border and the line along top(A_i)'s south border form an X-junction at the moment of handling—see Fig. 11(a). The negative tail occurs when the line along A_i's north border and the line along top(A_i)'s south border formed an X-junction in $\mathcal{L}_2$, but the end of the line along A_i's north border moved up when we handed that line earlier in Step 2—see Fig. 11(c).

(b) If A_i is not adjacent to bottom(A_i) and bottom(A_i) is lower than A_i in $\mathcal{L}_2$, 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)—see the third picture in Fig. 10.

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

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

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 every time a region grows a tail along it. Hence the slice line is shifted at most the number of tails that is grown along it. Next we analyze the maximum number of tails along a slice line.

Let k_1 be the number of regions growing one tail, and k_2 be the number of regions growing two tails. Let n_t be the number of tails. First of all, in order for the regions to have the need to grow any tails at all, there should be at least 2 regions on the opposite side of the line. Beside those two regions, for every region that grows 2 tails there is at least one region on the opposite side of the line, that is located between the regions the tails are reaching out for. Thus we have at least $k_2 + 2$ regions on the "non-growing-tails" side of the line. That means that the total number of regions on both sides of the line is at least $k_1 + 2k_2 + 2$, and at the same time it is at most n. The

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

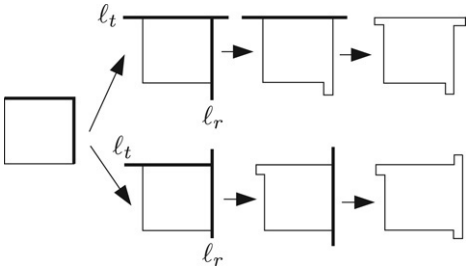

Fig. 12. Tailing a region.

number of tails is $k_1 + 2k_2$. We have $k_1 + 2k_2 \leqslant n - 2$. Hence a line is shifted at most $n - 2$ times. 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 or 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 (see Fig. 12). □

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.

The result of applying Step 2 to the layout of Fig. 5 is shown in Fig. 6.

Step 3: *Repairing the areas*

When we repaired the adjacencies in Step 2, we re-introduced some 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 since 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 ℓ; 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 sides of $C(\ell)$ are contained in the boundary of the tail strip of ℓ. The position of the right side of $C(\ell)$ 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, and let 2ε be the width of the tail strip of ℓ_2.

(i) *A_i did not get a downward tail and A_{i+1} did not get an upward tail* (see Fig. 13(a)).

We set the right side of the container $C(\ell)$ to be collinear with the part of ℓ_2 lying within ℓ's shift strip (see Fig. 13(a), (b)).

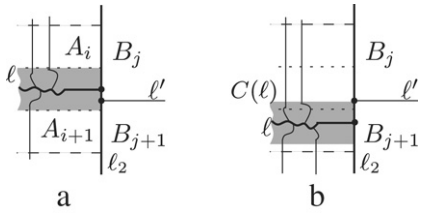

Fig. 13. (a) ℓ and ℓ_2 form a T-junction and there is another junction on ℓ_2 within $C(\ell)$; (b) $C(\ell)$ is moved down and makes A_i adjacent to B_{j+1}.

Note that there could be an extra junction on ℓ_2 within $C(\ell)$, formed by a line ℓ' that meets ℓ_2 from the other side—see Fig. 13. Then moving $C(\ell)$ in the direction of that junction could move ℓ past the junction, thus creating

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

a new adjacency and destroying an existing adjacency. In Fig. 13, for instance, the adjacency between A_{i+1} and B_j is destroyed and the adjacency between A_i and B_{j+1} is created. We claim that this can only happen if ℓ_2, ℓ, and ℓ' formed an X-junction before Step 1. Indeed, suppose they did not form an X-junction. If they still do not form an X-junction after Step 1, then by definition of the tail-strip width, the junction of ℓ_2 and ℓ' is outside the tail strip of ℓ. If, on the other hand, they do form an X-junction after Step 1, then A_{i+1} would have received a zero-tail in Step 2. Hence, ℓ_2, ℓ, and ℓ' formed an X-junction before Step 1, as claimed. So if the input graph is triangulated, this situation in fact does not arise. (If the input graph was not triangulated, then moving ℓ past the junction does not destroy any required adjacency, it just replaces one diagonal in a 4-cycle by the other.)

(ii) *A_i got a downward tail or A_{i+1} got an upward tail.*
In this case the right side of $C(\ell)$ will go through the leftmost edge of the tail of $A_i(A_{i+1})$—see Fig. 14.

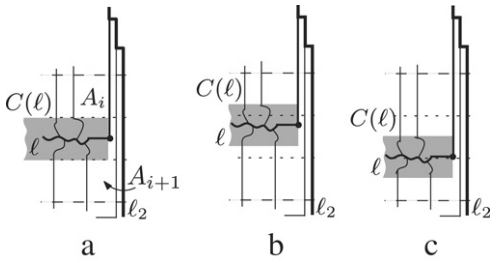

Fig. 14. (a) A_{i+1} has an upward tail; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down.

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.

Figs. 15–17 illustrate the case when ℓ and ℓ_2 were involved in an X-junction in $\mathcal{L}_1$—hence A_{i+1} could have a ("normal", zero- or negative) tail within ℓ's tail strip.

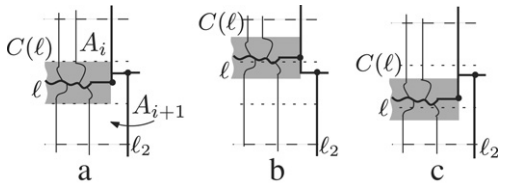

Fig. 15. (a) A_{i+1} got an upward tail with its end inside ℓ's tail strip; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down.

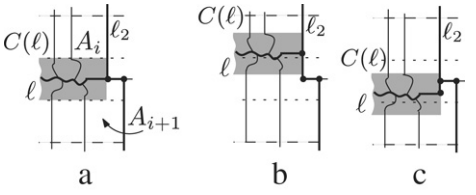

Fig. 16. (a) A_{i+1} got an upward zero-tail; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down.

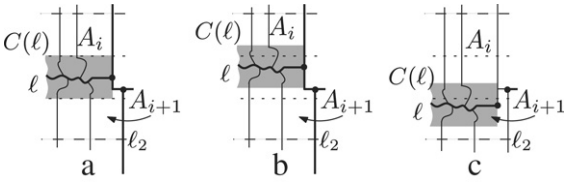

Fig. 17. (a) A_{j+1} has a negative tail upward; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down.

The position of the left side of $C(\ell)$ is determined in a similar fashion (with 2ε being the width of the tail strip of ℓ_1), as follows. Let B_j and B_{j+1} be the regions above and below ℓ and bordering ℓ_1.

(i) *B_j did not become a destination of an upward tail and B_{j+1} did not become a destination of a downward tail.*
This situation is symmetric to case (ii) of the right side of $C(\ell)$—see Fig. 18.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

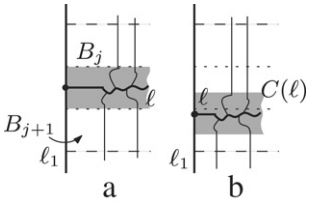

Fig. 18. (a) ℓ and ℓ_1 form a T-junction; (b) $C(\ell)$ is moved down.

(ii) *B_j became the destination of a downward tail or B_{j+1} became the destination of an upward tail.* The left side of $C(\ell)$ will go through the rightmost part of the slice line ℓ_1—see Fig. 19.

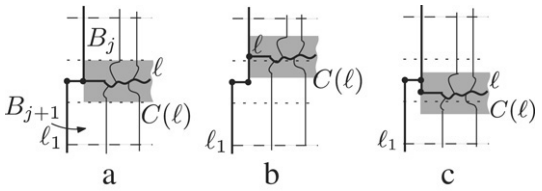

Fig. 19. (a) B_{j+1} is a destination of a downward tail; (b) moving $C(\ell)$ up; (c) moving $C(\ell)$ down.

Note that in all cases the required adjacencies – that is, the adjacencies in $\mathcal{L}_1$ – 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 to the left of 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. (Note that since the total area of the regions corresponding to the subtree rooted at the node ℓ is already correct, correcting the total area of the regions in the left subtree will automatically correct the total area of the regions corresponding to the right subtree.) 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 has 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}).

The shape of a region after Step 3 depends on the configuration and behavior of the four slice lines bounding it. All possible configurations are depicted in Fig. 20. Here we present the complexity analysis of a region A where the slice lines bounding A form the configuration depicted in Fig. 20(i).

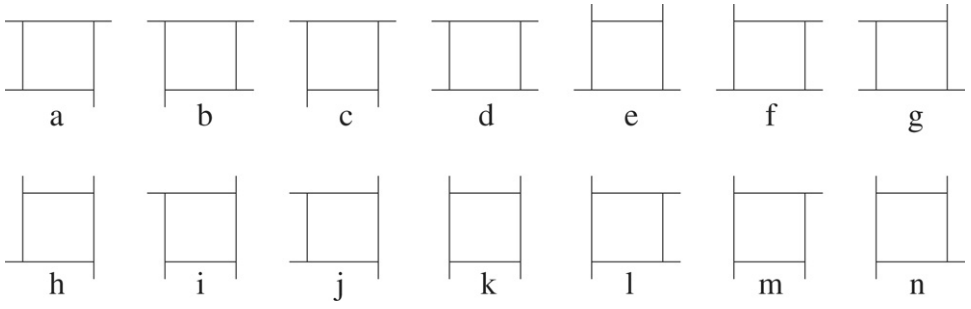

Fig. 20. All possible configurations of slice lines surrounding a region.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

Step 3 only has an effect on A if the slice lines around A are moved. We denote these lines by ℓ_1, ℓ_2, ℓ_3, and ℓ_4. Further, we denote by C_{ij} the corner of A where the lines ℓ_i and ℓ_j meet—see Fig. 21.

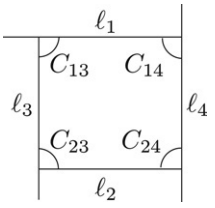

Fig. 21. Lines and corners around a region.

Step 2 only adds tails to regions that lie to the left or below a slice line and then only to those corners that are immediately adjacent to the slice line in question. Hence A might have tails at C_{14}, C_{24}, and C_{13} after Step 2. Every tail that A gained in Step 2 can only grow longer or shorter when the lines ℓ_1, ℓ_2, or ℓ_3 are moved in Step 3 (see Figs. 14–17). Thus, each of C_{14}, C_{24}, and C_{13} which had a non-zero-tail after Step 2 still contributes at most one concave vertex to the complexity of A after Step 3 (see Figs. 14, 15 and 17). If any of these three corners had a zero-tail after Step 2 (and hence did not contribute a convex vertex after Step 2) then it can acquire a tail in Step 3 (see Fig. 16). Thus in total corners C_{14}, C_{24}, and C_{13} contribute at most 3 concave vertices after Step 3. The corner C_{23} cannot have had a tail after Step 2, but it might have become the destination of a tail. In this case it might gain a concave vertex in Step 3 (see Fig. 19). Thus after Step 3 the region A has at most 4 concave vertices, and hence its complexity is at most 12.

Fig. 22 depicts all possible shapes that a region can have after Step 3.

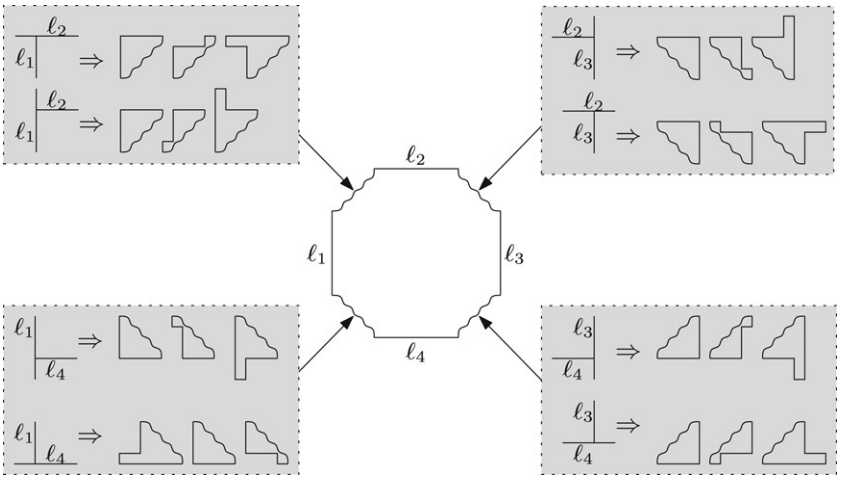

Fig. 22. All possible shapes of a region after Step 3.

It is easy to verify – see Figs. 13–19 – that a similar kind of reasoning can be applied to prove that for all other configurations of slice lines around a region the total number of concave vertices after Step 3 is bounded by four—at most one for each corner of the region in $\mathcal{L}_1$. □

It remains to prove that we can chose 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 form a junction 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.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

As for the second requirement, we provide very rough upper bounds on the values for the width of the shift and tail 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 in Step 3. (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 $\delta_k := \Delta/4 * ((\Delta(1-\Delta))/10)^{n-k-1}$ and the width of the tail strip is set to $\varepsilon_k := \delta_k * \Delta/2$, for $1 \leqslant k \leqslant n-1$, then Requirements 1 and 2 are fulfilled.*

Proof. We have to prove that if the width of the shift strip of slice line ℓ_k is set to $\delta_k = \Delta/4 * ((\Delta(1-\Delta))/10)^{n-k-1}$ and the width of the tail strip is set to $\varepsilon_k = \delta_k * \Delta/2$, for $1 \leqslant k \leqslant n-1$, then – when we move the container $C(\ell_k)$ of a slice line ℓ_k – there is enough area within its shift strip to compensate for the error introduced between its children. The proof is by induction on the slice line index.

Induction basis: ℓ_1. The error is introduced only when tailing during Step 2, because ℓ_1 is the first slice line handled in Step 3. Since the length of the slice line (here and later in the proof by the length of a slice line we mean its original length in $\mathcal{L}_1$) is 1, the error is less than half of the tail strip area, which is ε_1, and the available "maneuvering" area is $\delta_1 - \varepsilon_1 = (1 - \Delta/2)\delta_1$ which is clearly greater than ε_1.

Induction hypothesis: When the slice lines $\ell_1, ..., \ell_{k-1}$ were handled in Step 3, the container of each slice line was only moved within its shift strip.

Induction step: Consider a line ℓ_k at some node v in the BSP. Let it be vertical. Denote by R_{left} and R_{right} the union of the regions in the left and right subtrees of v. Consider the error for R_{right} (the error for R_{left} is the same with the sign reversed). Let ℓ_i, ℓ_j and ℓ_m be the slice lines around R_{right}. They are higher up in the hierarchy. Hence $i, j, m < k$ and the lines have already been processed. This implies that the error induced by each of these lines is not more than the area of the shift strips they are in.

The absolute value of error(R_{right}) is the sum of errors introduced by changes along ℓ_i, ℓ_j and ℓ_m (tailing in Step 2 and shifting in Step 3) and by tailing along ℓ_k itself. By the induction hypothesis $|\text{error}(R_{\text{right}})| < S_k$, where S_k is the sum of areas of shift strips of ℓ_i, ℓ_j and ℓ_m around R_{right} and the area of the part of ℓ_k's tails strip within R_{right}—see Fig. 23. The area for "maneuvering" inside ℓ_k's shift strip is bounded from below by the area s_k of the part of the ℓ_k's shift strip on the left side of ℓ_k outside the tail strip between the shift strips of ℓ_j and ℓ_m—see Fig. 23.

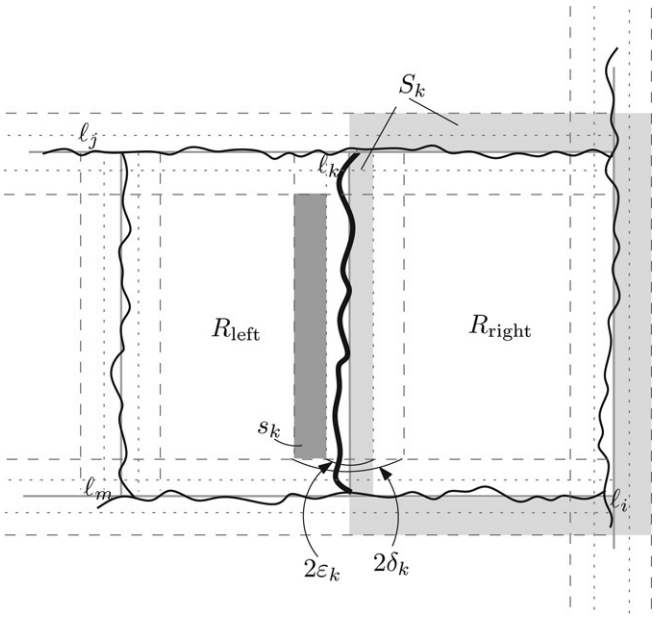

Fig. 23. The common area of R_{left} and R_{right} is correct. s_k is the minimal area we have available to correct the error between R_{left} and R_{right}. The error itself is bounded from above by S_k.

We need to show that $s_k - S_k > 0$. Since lengths of the slice lines ℓ_i, ℓ_j, ℓ_m and ℓ_k are less than 1, we have $S_k < \widetilde{S}_k$, where

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

$$\widetilde{S}_k = \delta_i + \delta_j + \delta_m + \varepsilon_k$$

and $s_k > \widetilde{s}_k$, where

$$\widetilde{s}_k = (\Delta - \delta_j - \delta_m) \cdot (\delta_k - \varepsilon_k).$$

Hence

$$s_k - S_k > \widetilde{s}_k - \widetilde{S}_k = (\Delta - \delta_j - \delta_m) \cdot (\delta_k - \varepsilon_k) - (\delta_i + \delta_j + \delta_m + \varepsilon_k).$$

Assume without loss of generality that $m > i$ and $m > j$, then $\delta_m = \max(\delta_i, \delta_j, \delta_m)$ and

$$\widetilde{s}_k - \widetilde{S}_k > (\Delta - 2\delta_m) \cdot (\delta_k - \varepsilon_k) - (3\delta_m + \varepsilon_k).$$

Substituting $\varepsilon_k = \Delta\delta_k/2$ on the right side we get

$$\begin{aligned}\widetilde{s}_k - \widetilde{S}_k &> (\Delta - 2\delta_m)(1 - \Delta/2)\delta_k - 3\delta_m - \Delta\delta_k/2 \\ &= \delta_k \cdot ((\Delta - 2\delta_m)(1 - \Delta/2) - 3\delta_m/\delta_k - \Delta/2) \\ &> \delta_k(\Delta(1 - \Delta)/2 - 5\delta_m/\delta_k).\end{aligned}$$

Since $k > m$ and therefore $\delta_m/\delta_k = (\Delta(1 - \Delta)/10)^{k-m} \leqslant \Delta(1 - \Delta)/10$ we have

$$\begin{aligned}\widetilde{s}_k - \widetilde{S}_k &\geqslant \delta_k(\Delta(1 - \Delta/2) - 5(\Delta(1 - \Delta)/10)) \\ &= 0.\end{aligned}$$

Hence $s_k - S_k > 0$ as claimed. □

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 [9]. Now let $\mathcal{G}$ 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 rectangles [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.

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

Proof. This follows directly from the correctness of the algorithm of the previous section, except for one subtlety: In the previous section the layout $\mathcal{L}_1$ only contained T-junctions – this is true since the input graph was triangulated –

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

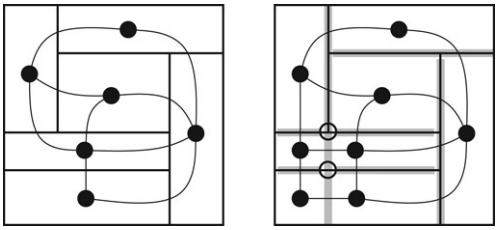

Fig. 24. Cutting $\mathcal{L}_0$ (left) with a BSP produces a layout $\mathcal{L}_1$ (right) with two X-junctions (circled).

but this is no longer the case. $\mathcal{L}_1$ is now obtained by cutting the layout $\mathcal{L}_0$ with a BSP, which means that it can have X-junctions. Hence, the faces of the graph $\mathcal{G}(\mathcal{L}_1)$ dual to $\mathcal{L}_1$ are not only triangles, but also 4-cycles—see Fig. 24.

Since the original layout $\mathcal{L}_0$ does not have X-junctions, all X-junctions in $\mathcal{L}_1$ are caused by edges of $\mathcal{L}_0$ being cut by a splitting line of the BSP.[3] This means that of the four subrectangles incident to an X-junction, two neighboring ones must belong to the same rectangle R_1 in $\mathcal{L}_0$; the other two subrectangles belong to other rectangles R_2 and R_3, possibly with $R_2 = R_3$.

Lemma 1(i) states that after Step 2, all adjacencies present in $\mathcal{L}_1$ are also present in $\mathcal{L}_3$. In the proof we did not use that $\mathcal{G}(\mathcal{L}_1)$ only has triangular faces, so the statement is still true. However, if $\mathcal{G}(\mathcal{L}_1)$ also has faces that are 4-cycles, then Step 2 might introduce new adjacencies between opposite vertices of such a 4-cycle. Because, by construction, every 4-cycle has two neighboring vertices that correspond to the same rectangle in $\mathcal{L}_0$, these added adjacencies do not pose a problem. They simply represent an existing adjacency between two rectangles in $\mathcal{L}_0$ for the second time. Recall that in the cases depicted in Figs. 13 and 18 the adjacencies may have been changed in Step 3 if there was an X-junction. But by the same argument as above, this does not pose a problem. Finally, note that the adjacencies between the subrectangles that belong to the same rectangle in $\mathcal{L}_0$ ensure that the regions of $\mathcal{L}_5$ are connected. We conclude that the algorithm does indeed produce a valid layout with the correct adjacencies. It follows immediately from the construction that it also gives each region the correct area. □

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.

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

Proof. A region is cut into at most four subregions A, B, C, and D when a rectilinear BSP is constructed on the rectangular dual $\mathcal{L}_0$. (When a region is cut into less than 4 subregions its complexity after Steps 1–3 can only be smaller than the worst-case complexity of a region cut into four.) W.l.o.g. we can assume that the first cut (by the line ℓ_1) is vertical—see Fig. 25.

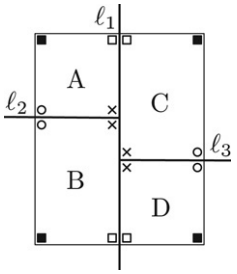

Fig. 25. Corners.

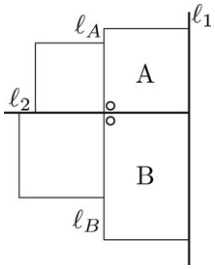

Fig. 26. Horizontal side corners.

[3] Here we assume for simplicity and without loss of generality that no two vertical (resp. horizontal) splitting lines used by the BSP have the same x-coordinate (resp. y-coordinate).

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

The four regions jointly have 16 corners, which can be classified as follows. There are four *original corners* which correspond to the corners of P (marked with ■). The remaining corners are *induced corners* which can be further subdivided into four *internal corners* (marked with $\times$), four *horizontal side corners* (marked with ◯), and four *vertical side corners* (marked with □).

We denote by P the region formed by the union of regions A–D after Step 3. The internal corners and any vertices they might have gained in Steps 1–3 lie inside P or on its edges and do not contribute to its complexity. In other words, they disappear when the subregions are merged. Hence, we only have to worry about what happens at the original and side corners.

According to the proof of Lemma 2 each original corner can gain one concave vertex in Steps 1–3 which implies that P has complexity $4 + 8 = 12$.

Now consider the horizontal side corners of A and B (see Fig. 26). By construction of the BSP, the slice line ℓ_A along A's west border and the slice line ℓ_B along B's west border both end on ℓ_2. In other words, ℓ_2 extends further to the west than the horizontal side corners. (Indeed, a slice line that just connects two sides of the same rectangle is useless and will not be used in the BSP.) Hence, in Step 2 the horizontal side corners of A and B are tailed, if at all, when ℓ_2 is handled. Next we argue that in fact there will be no tails to or from A and B at these side corners, and moreover they do not gain a concave vertex in Step 3. Before Step 1 the only north neighbor of B is A, and A and B are still adjacent after Step 1 because they both end on ℓ_1. This implies that B will not grow a tail in Step 2. Similarly, B is the only south neighbor of A, so no other region will grow a tail along ℓ_2 to reach A. This means that ℓ_2 must be straight at the horizontal side corners of A and B. It follows – see Figs. 13 and 18 – that no concave vertices are introduced in Step 3. Thus no extra vertices are introduced at the horizontal side corners of A and B. A similar argument applies to the horizontal side corners of C and D. Thus, the horizontal side corners contribute at most themselves to the list of vertices of P, hence at most 4 in total to P's complexity, which is now $4 + 8 + 4 = 16$.

It remains to analyze what happens at the vertical side corners. If A and C have no neighbors along ℓ_1 but B and D, then only one of the vertical side corners of A and C can get a tail during Step 2—see Fig. 27(a)–(b) for an example. By construction of the BSP, B and D must have additional neighbors along ℓ_1. It follows from the analysis below that in this case the vertical side corners contribute a total of 4 vertices to P's complexity which then becomes $4 + 8 + 4 + 4 = 20$. Similar reasoning applies to the case when B and D have no neighbors along ℓ_1 but A and C. In this case, at most one of the vertical side corners can be reached by a tail along ℓ_0—see Fig. 27(c)–(d) for an example.

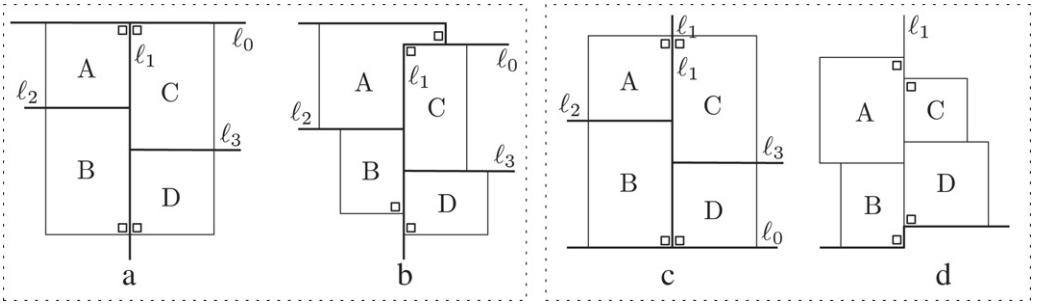

Fig. 27. (a) Both A and B have only one neighbor along ℓ_1, (b) after Step 2; (c) Both C and D have only one neighbor along ℓ_1, (d) after Step 2.

Now assume that all of A–D have neighbors along ℓ_1 that are not part of P. For what happens in Step 2, we resort to a complete case analysis.

We distinguish cases according to the location of the three corners a, b, and c of C and D with respect to the four intervals 1-4 on ℓ_1 which are defined by the corresponding corners of A and B—see Fig. 28. We denote each case with a triple (a, b, c), where, for example, $a = 1$ denotes a case where the corner a is contained in interval 1.

Only the cases when corners do not coincide with the endpoints of the intervals are depicted in Fig. 29, and in every case the vertical side corners contribute 4 vertices to P's complexity which becomes $4 + 8 + 4 + 4 = 20$, as claimed. The cases when one or more of the corners do coincide with the interval endpoints produce either the same or smaller complexity outer shapes.

Note that only the location of corners a, b, and c after Step 1 is relevant for P's outer shape, the relation between ℓ_2 and ℓ_3 before Step 1 influences only the interior of P which has no impact on P's complexity. All cases in Fig. 29

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

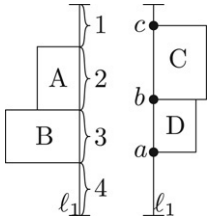

Fig. 28. Case distinction.

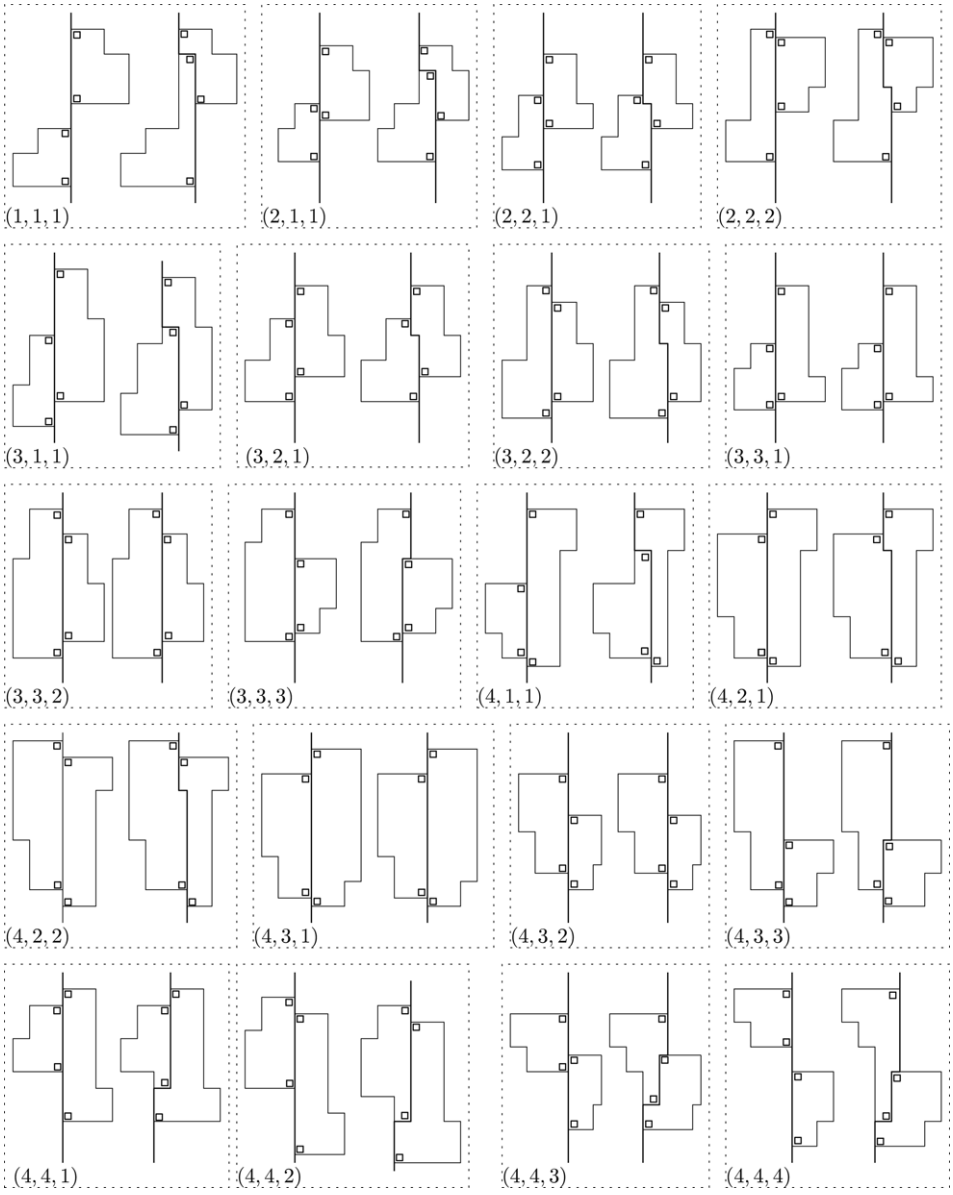

Fig. 29. Corners charged to the vertical side vertices after Step 2.

are drawn for the situation where ℓ_2 is above ℓ_3. As an example, in Fig. 30 we illustrate the interior of P for both relative positions of ℓ_2 and ℓ_3.

It remains to argue that Step 3 does not introduce any extra vertices at the vertical side corners. This can be done by a careful inspection of each of the cases in Fig. 29. For instance, consider case (1, 1, 1). Since only A and possibly

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

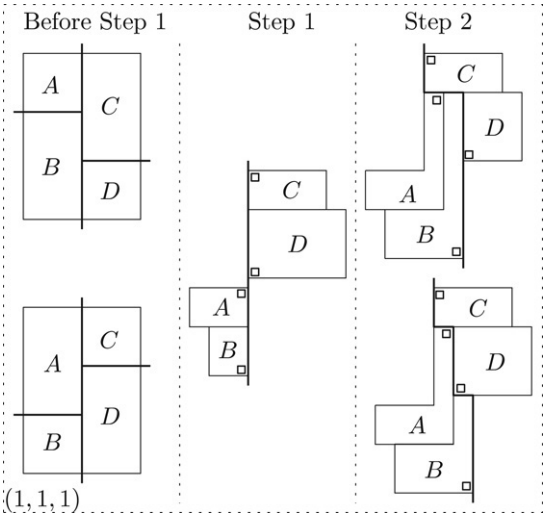

Fig. 30. The interior of P depending on the relative positions of ℓ_2 and ℓ_3 in case (1, 1, 1).

B were adjacent to C, no other tails end on C. Hence, ℓ_1 is straight near C's vertical side corner, and the container of the slice line through C's top edge extends all the way to ℓ_1. Hence, no extra vertices are introduced when that container is moved. Similarly, the container of the slice line through the bottom edge of B extends all the way to ℓ_1, so no extra vertices are introduced there either. Finally, it is easy to see that no extra vertices are introduced in Step 3 at the vertical side corners of A and D. Similar reasonings can be applied to all other cases in Fig. 29; we omit easy but tedious details. Hence, the total complexity remains 20, as claimed. □

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. [12] 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 two rectangles and then proceed as in the previous case: We cut the collection of rectangles with a BSP to obtain a sliceable layout $\mathcal{L}_1$, we assign weights to the rectangles in $\mathcal{L}_1$, run Steps 1–3, and merge regions belonging to the same vertex in $\mathcal{G}$. This leads to the following result.

Theorem 3. *Any vertex-weighted plane triangulated graph $\mathcal{G}$ admits a cartogram of complexity at most* 40.

Proof. We preprocess the rectilinear layout created by the algorithm of Liao et al. [12] by cutting each T-shaped region A (L-shapes can be considered to be degenerated T-shapes) into two rectangles A_1 and A_2 as shown in Fig. 31. This is possible because the algorithm presented in [12] always produces layouts where each T-shaped region has the following properties

(i) the region is oriented as a letter "T" written upside down
(ii) the heights of its 2 horizontal branches are identical.

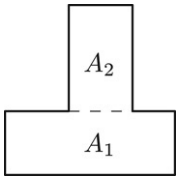

Fig. 31. Cutting the T-shape.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

Each of the regions A_1 and A_2 after Step 3 has at most 20 vertices. Hence the complexity of their union is at most 40. □

4. Runtime analysis

Our algorithm uses three data structures. The first one stores the input graph $\mathcal{G}$. The second one stores the adjacency graph $\mathcal{G}(\mathcal{L}_1)$ of $\mathcal{L}_1$, that is, the adjacency graph of the sliceable subdivision which we obtain after constructing a BSP on the rectangular dual $\mathcal{L}_0$ of $\mathcal{G}$. The third structure represents the layout that is being transformed into the cartogram. $\mathcal{G}$ and $\mathcal{G}(\mathcal{L}_1)$ are stored as a set of vertices, where with each vertex we store four lists of pointers that refer to its north, east, south and west neighbors. Furthermore, each vertex v of $\mathcal{G}$ stores pointers to the at most 12 nodes in $\mathcal{G}(\mathcal{L}_1)$ that correspond to v. Also, each vertex y of $\mathcal{G}(\mathcal{L}_1)$ has a pointer to the corresponding region in the layout. The layout itself is stored in a BSP tree $\mathcal{T}$, as described in Section 2. Recall that each internal node of $\mathcal{T}$ stores a slice line. With each slice line ℓ we also store two sorted lists of regions that are bordering ℓ from the left and the right (or the bottom and the top), respectively. Each leaf of $\mathcal{T}$ contains a pointer to the corresponding node in $\mathcal{G}(\mathcal{L}_1)$.

Computing a rectangular dual $\mathcal{L}_0$ of $\mathcal{G}$ takes linear time [9]. If $\mathcal{G}$ does not allow a rectangular dual, then we compute a rectilinear dual using L- and T-shapes in addition to rectangles, also in linear time [12]. We cut each region into at most three rectangles to construct a rectangular subdivision $\mathcal{L}_0$. Constructing a BSP on $\mathcal{L}_0$ takes $O(n \log n)$ time, using the algorithm by d'Amore and Franciosa [3]. At the same time we can also construct $\mathcal{G}(\mathcal{L}_1)$ at no additional cost since each region is split at most three times. The lists of adjacent regions for each slice line ℓ in $\mathcal{T}$ can be created by traversing $\mathcal{T}$ bottom-up in linear time.

In Step 1 we first calculate the weight for each internal node, traversing $\mathcal{T}$ bottom-up, and then move each slice line to meet the weight requirements, traversing $\mathcal{T}$ top-down. Both weight calculation and moving the lines takes linear time in total. Step 2 takes $O(k)$ time per slice line ℓ, where k is the number of ℓ's neighbors. Since each region is a neighbor of at most 4 lines, updating all lines takes linear time in total. Moving the slice lines in Step 3 also takes linear time. Finally, combining the at most 12 regions in the cartogram that correspond to a vertex v of $\mathcal{G}$ can be done in linear time as well.

Theorem 4. *Any vertex-weighted plane triangulated graph $\mathcal{G}$ with n vertices admits a cartogram of complexity at most* 40*, which can be constructed in $O(n \log n)$ time.*

5. Conclusions

We proved that every plane triangulated vertex-weighted graph admits a rectilinear cartogram of constant complexity. For a graph with n vertices such a cartogram can be constructed in $O(n \log n)$ time. We implemented our algorithm (adding some heuristics to improve its performance) and presented the results in [4]. Our experimental results show that in practice, our algorithm always constructs a cartogram with complexity at most 10. Nevertheless, it is an interesting open problem to give tight upper and lower bounds on the 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 complexity bound which we obtain for such graphs is much better. Finally, the tails which our algorithm adds to get the correct adjacencies can be quite thin—even too thin, from a practical point of view. But again, our experiments [4] show that by using some simple heuristics one can ensure that the tails are wide enough to enable the reader to clearly identify the regions' adjacencies.

References

[1] J. Bhasker, S. Sahni, A linear algorithm to check for the existence of a rectangular dual of a planar triangulated graph, Networks 7 (1987) 307–317.
[2] T. Biedl, B. Genc, Complexity of octagonal and rectangular cartograms, Technical Report, University of Waterloo, 2005.
[3] F. d'Amore, P. Franciosa, On the optimal binary plane partition for sets of isothetic rectangles, Information Processing Letters 44 (5) (1992) 255–259.
[4] M. de Berg, E. Mumford, B. Speckmann, Optimal BSPs and rectilinear cartograms, in: Proc. 14th International Symposium on Advances in Geographic Information Systems, 2006, pp. 19–26.
[5] Cartogram Central. http://www.ncgia.ucsb.edu/projects/Cartogram_Central/index.html.
[6] B. Dent, Cartography — Thematic Map Design, 5th edn, McGraw-Hill, 1999.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087

[7] N. Dougenik, D. Niemeyer, An algorithm to construct continuous area cartograms, Professional Geographer 3 (1985) 75–81.
[8] R. Heilmann, D. Keim, C. Panse, M. Sips, Recmap: Rectangular map approximations. in: Proc. IEEE Symposium on Information Visualization, 2004, pp. 33–40.
[9] G. Kant, X. He, Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems, Theoretical Computer Science 172 (1997) 175–193.
[10] K. Koźmiński, E. Kinnen, Rectangular dual of planar graphs, Networks 5 (1985) 145–157.
[11] M.v. Kreveld, B. Speckmann, On rectangular cartograms, Computational Geometry: Theory and Applications 37 (3) (2007) 175–187.
[12] C.-C. Liao, H.-I. Lu, H.-C. Yen, Compact floor-planning via orderly spanning trees, Journal of Algorithms 48 (2003) 441–451.
[13] M. Rahman, K. Miura, T. Nishizeki, Octagonal drawings of plane graphs with prescribed face areas, in: Proce. 30th International Workshop on Graph-Theoretic Concepts in Computer Science, in: LNCS, vol. 3353, Springer, 2004, pp. 320–331.
[14] E. Raisz, The rectangular statistical cartogram, Geographical Review 24 (1934) 292–296.
[15] C. Thomassen, Plane cubic graphs with prescribed face areas, Combinatorics, Probability and Computing 1 (1992) 371–381.
[16] G. Yeap, M. Sarrafzadeh, Sliceable floorplanning by graph dualization, SIAM Journal of Discrete Mathematics 8 (2) (1995) 258–280.

Please cite this article in press as: M. de Berg, et al., On rectilinear duals for vertex-weighted plane graphs, Discrete Mathematics (2008), doi:10.1016/j.disc.2007.12.087