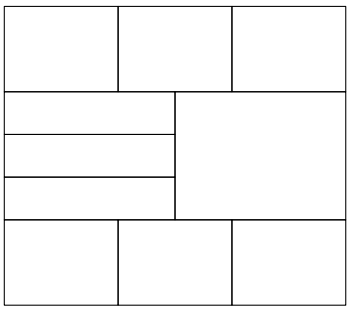
Technische Universiteit
Eindhoven
University of Technology

Department of Mathematics and Computer Science
Applied Geometric Algorithms Research Group

On k-Sided Rectangular Duals

S. J. Beekhuis



Supervisors:
prof. dr. B. Speckmann
dr. K. A. B. Verbeek
dr. J. Nederlof
dr. R. A. Pendavingh

Eindhoven, March 9, 2017
sander@sanderbeekhuis.nl

Abstract

A *rectangular layout* (or simply *layout*) $\mathcal{L}$ is a partition of a rectangle into a finite set of interior-disjoint rectangles. Rectangular layouts have many applications, for example as rectangular cartograms in cartography, chip designs in VLSI and floorplans in architecture. A rectangular cartogram is a map in which the regions are replaced by scaled rectangles representing some quantity.

In the rectangular cartogram application it is desirable to find a layout that always keeps the same adjacencies between the interior rectangles when their sizes change. For example, when displaying a cartogram of a single quantity at different points in time, we prefer to do this using a set of layouts with the same adjacencies. Such layouts are *area-universal*.

The interior of a rectangular layout contains vertical and horizontal line segments. Any line segment that can not extend farther on either side is a *maximal segment*. A rectangular layout is *k-sided* if every maximal segment has at most k rectangles on one of its sides. A *rectangular dual* of a graph G is a rectangular layout whose adjacencies are the same as those of G. A necessary and sufficient condition for a layout to be area-universal is that it is *one-sided*, that is k-sided with $k = 1$. Unfortunately, not all graphs with a rectangular dual have a one-sided dual.

For graphs without a one-sided dual, a k-sided dual with k as small as possible is often a dual that has the smallest risk of adjacencies changing when changing the size of the interior rectangles.

In this thesis we prove that we can not find a k-sided rectangular dual for all graphs admitting a rectangular dual, no matter the constant k. However, for certain graphs, namely those without separating 4-cycles, this might be possible. We do not quite obtain this result. Instead, we present an algorithm giving a $d - 1$-sided dual, with d the maximal degree of the vertices of G in $\bar{G}$, a so-called corner assignment.

Contents

Introduction 1

Motivation. In for example atlases, *rectangular cartograms* are used to display information, such as population or economic strength, in a spatial manner. In a rectangular cartogram the geographic regions of an ordinary map are replaced by rectangles; we let these rectangles maintain adjacencies with each other to suggest geographic location and scale them proportionally to the quantities they represent. Raisz [7] introduced these cartograms and provided cartograms of, for instance, land area, population (see Figure 1) and wealth of the United States of America. In a rectangular cartogram it is preferable to maintain the adjacencies of the regions that are replaced by rectangles, this in order to keep the representation recognizable. However, this is only possible under certain conditions that will be given later in this chapter. The cartograms made by Raisz do not keep all adjacencies. For example, Florida and Alabama are not adjacent in Figure 1 while they are in reality.

The value of the displayed quantities, like population or wealth, often changes over time. When we draw a set of cartograms displaying a single quantity at different moments in time, it is desirable that the adjacencies between the rectangles in these cartograms remain the same, no matter the moment in time. Moreover, it would be even better if the nature of these adjacencies, that is whether the rectangles border in a vertical or horizontal manner, does not change. This raises the question: When is this possible?

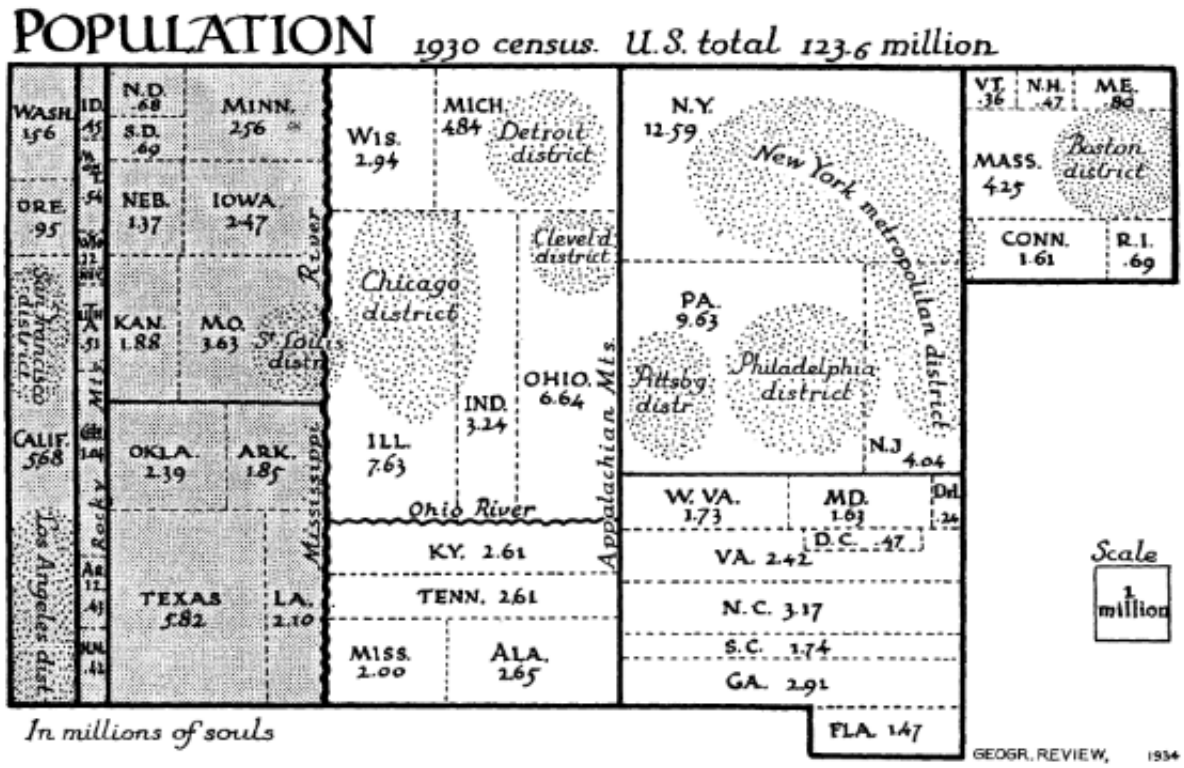


Figure 1: A cartogram by Raisz [7] made in 1934.

Rectangular layout. Mathematically, a rectangular cartogram is a *rectangular layout* (or simply *layout*). A layout $\mathcal{L}$ is a partition of an axis-parallel rectangle into a finite set of interior-disjoint axis-parallel rectangles. Roughly speaking, we say that two rectangular layouts are *combinatorially equivalent* (or simply *equivalent*) when they have the same adjacencies and these adjacencies are in the same manner, that is horizontal or vertical. We will later, in Section 3, define this more thoroughly. A rectangular layout is *area-universal* when it has a combinatorially equivalent layout, regardless the area sizes we assign to each rectangle. Three area-universal equivalent layouts are given in Figure 2.

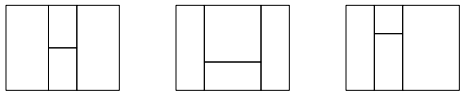


Figure 2: Three area-universal layouts.

The question above then becomes: For which maps can we create area-universal layouts? Clearly not those maps that do not have any corresponding layouts with the same adjacencies. However, it will turn out there are more maps without a corresponding area-universal layout.

Adjacency graphs. We can represent the adjacencies of map regions by an *adjacency graph* G where each region is represented by a vertex and two vertices are connected by an edge exactly when their regions are adjacent. Similarly, in the *adjacency graph* $\mathcal{G}(\mathcal{L})$ of a rectangular layout $\mathcal{L}$ each rectangle is represented by a vertex and two vertices are connected by an edge exactly when their rectangles are adjacent. A layout $\mathcal{L}$ is a *rectangular dual* of a graph G if we have that $G = \mathcal{G}(\mathcal{L})$. In general, a single graph can have multiple non-equivalent rectangular duals. An example of this is the graph in Figure 3.

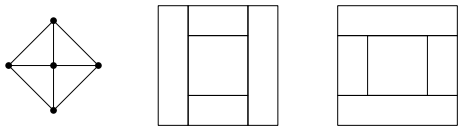


Figure 3: A graph with two non-equivalent duals.

Rinsma found a graph in [8], displayed in Figure 4, that has no area-universal rectangular duals. That is, all rectangular layouts with this graph as adjacency graph are not area-universal.

One-sided layouts. So, unfortunately not all graphs admit area-universal duals. We would like to know which graphs do. Before we can state this, we need to define one-sided layouts. Note that the interior of a rectangular layout contains vertical and horizontal line segments. Any line segment that can not extend any farther on either side is a *maximal segment*. A rectangular layout is *one-sided* if every maximal segment has only one rectangle on one of its sides. The layouts in Figure 2 and 3 are all one-sided.

In [1] Eppstein et al. show that rectangular layouts are area-universal exactly when they are one-sided. So, Rinsma's result also implies that not every adjacency graph of a map can be represented by a one-sided layout.

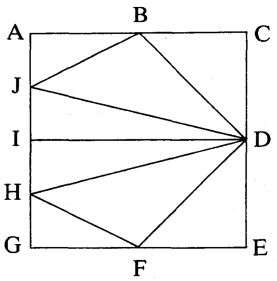
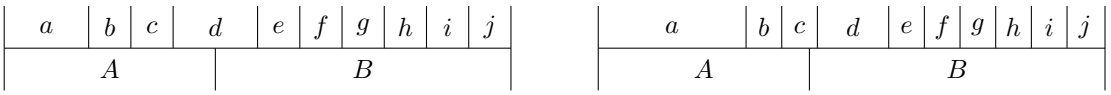


Figure 4: The graph by Rinsma [8] that is not one-sided.

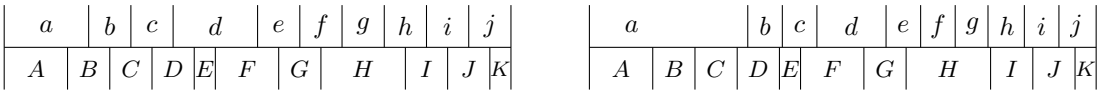
k-sided layouts. Let us consider those graphs that do not admit any one-sided, and thus area-universal, dual as rectangular dual. Since any dual for such a graph is not area-universal, it is inevitable that adjacencies between rectangles in this dual change when we resize them. For these graphs we want to find layouts that have the least number of adjacency changes when area sizes change. This is beneficial for, for example, applications displaying cartograms on a continuous timescale. Since every time the adjacencies of the layout change, we have to compute a different rectangular layout with the right adjacencies, providing the user a rougher viewing experience.

We call a layout *k-sided* if k is the smallest integer such that every maximal segment has at most k rectangles on one of its sides. This is a direct generalization of one-sidedness. This generalization is useful because, when changing the areas of rectangles in a k-sided layout fewer adjacencies change, in general, if k is small then if k is large.



(a) Before resizing a (b) After resizing a

Figure 5: A 2-sided segment



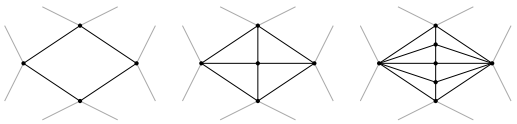
(a) Before resizing a (b) After resizing a

Figure 6: A 10-sided segment

We illustrate this by comparing a typical 2-sided and a typical 10-sided segment. Let us first consider the 2-sided segment in Figure 5a, if the size of a doubles only two adjacencies change, namely dA disappears and cB appears, as can be seen in Figure 5b. While for a typical 10-sided segment doubling the size of a leads to 15 changed adjacencies, namely aC aD bB bC bD bE cC cD cE cF dE dG eF eH fG, as can be seen in Figure 6. Hence we would like to find k-sided layouts for all graphs with k as small as possible.

Counterexample. In this thesis we provide two results on k-sidedness. The first result is the existence of a family of graphs G_k that, for any constant $k \in \mathbb{N}$, has members that are not k-sided (Theorem 4). The family of graphs in this result is characterized by the occurrence of nested separating 4-cycles. A 4-*cycle* is a cycle of length 4. Such a cycle is separating if there are vertices in both its interior and exterior. Separating 4-cycles are *nested* if one is contained in the other, but some of their vertices overlap. These different types of 4-cycles are demonstrated in Figure 7.

Separating 4-cycles in general are difficult to treat when trying to create a k-sided layout. But examples found during our research seem to indicate that, in particular, nested 4-cycles are the most difficult to treat. Finding k-sided rectangular duals is not the only problem that has difficulty with separating 4-cycles.

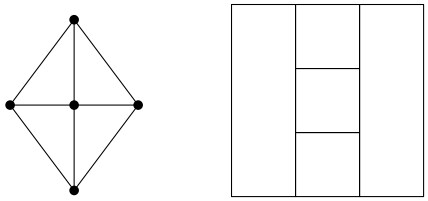


(a) A 4-cycle. (b) A separating 4-cycle. (c) A nested separating 4-cycle.

Figure 7

Yeap and Sarrafzadeh [10] investigated the problem of finding a *sliceable* dual for a graph. A rectangular layout is sliceable when it is either a single rectangle or when it has a single maximal segment which splits the layout into two sliceable layouts. Yeap and Sarrafzadeh were able to show that all graphs without a separating 4-cycle have a sliceable rectangular dual. They were unable to gain traction on graphs with a separating 4-cycle (in their words this is a complex cycle of length 4).

Nevertheless, not all problems are intractable on graphs with a separating 4-cycle. The problem of finding an area-universal rectangular dual of a graph, already mentioned above, was investigated by Eppstein et al. in [1]. They managed to find such a rectangular dual, when it exists, for any graph G. This result does affect graphs containing separating 4-cycles since there are such graphs that admit area-universal duals, as can be seen in Figure 8.



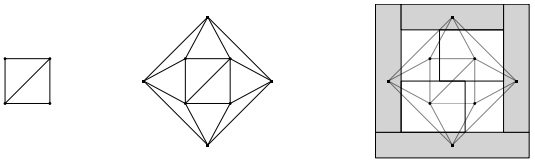
(a) Graph with separating 4-cycle. (b) Corresponding area-universal dual.

Figure 8

Corner assignments. Before we can state the second result of this thesis, we first introduce the

concept of corner assignments. A corner assignment $\bar{G}$ of a graph G is an augmentation of G with 4 external vertices, which we call its *poles*, with the following three properties (i) every interior face has degree 3, (ii) the exterior face has degree 4 and (iii) $\bar{G}$ has no separating triangles. A corner assignment fixes which rectangles are in the corners of the rectangular dual $\mathcal{L}$, namely those adjacent to two poles, which explains the terminology. In Figure 9 we see a graph, a possible corner assignment of this graph, and a possible rectangular dual (in this case, up to equivalence, the only possible dual) of this corner assignment.

Existence of rectangular duals. Now we have formulated corner assignments we can also state which graphs admit rectangular duals. A graph admits a rectangular dual if and only if it admits a corner assignment. This was shown independently by Kozminski and Kinnen [6] and Ungar [9].



(a) A graph G. (b) A corner assignment of G. (c) A rectangular dual corresponding to this corner assignment.

Figure 9

Algorithm. After finding the counterexample, we focused our efforts on obtaining an algorithm that would provide a k-sided layout for corner assignments without a separating 4-cycle, for some constant $k \in \mathbb{N}$. Unfortunately, we fell short of this goal and only found an algorithm that provides a $d-1$-sided layout, where d is the maximal degree of the vertices of G in the corner assignment $\bar{G}$ (Theorem 6).

That being said, during our research we did not find any graphs without separating 4-cycles that did not admit any 2-sided layouts. For example, the graph, by Rinsma, without a one-sided dual does have a 2-sided dual, as can be seen in Figure 10. We thus conjecture that these graphs admit a 2-sided layout and it is just the algorithm for finding them that eludes us.

Its worthy of note, that the bound obtained by the algorithm is not stronger than the bound provided by the counterexample. That is, there might be an algorithm providing $O(d)$-sided layouts for all graphs G admitting a rectangular dual.

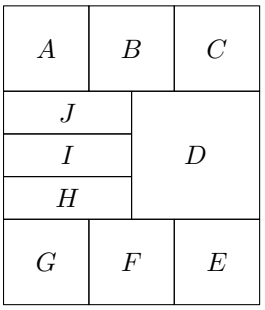


Figure 10: A 2-sided dual of the Rinsma graph in Figure 4.

Overview. The rest of this thesis is focused on obtaining these two results. In order to do this we give extensive definitions on graphs, paths and cycles in Section 2. In Section 3 we introduce the notion of regular edge labellings. A regular edge labeling is a way of coloring and orienting the edges of a graph that corresponds to a rectangular dual of that graph. We frequently use this notion in the rest of this thesis. Once we have these preliminaries out of the way, we prove Theorem 4 in Section 4 and Theorem 6 in Section 5. We prove Theorem 4 by counterexample and Theorem 6 by giving a constructive algorithm.

2 *Preliminaries*

Before we continue with the rest of thesis we will in this section first define some basic definitions for graphs, paths and cycles.

Graphs. A *graph* G is an abstraction of a network. The objects are represented by a set of *vertices*. Connections between objects are represented by a set of *edges*; each edge connects two vertices. Two distinct edges do not have the same vertices and no edge starts and ends at the same vertex. That is, all graphs in this thesis are *simple*. An edge is *incident* to a vertex v if that edge connects v to another vertex. The *degree* of a vertex is the number of edges incident to this vertex. All graphs in this thesis are *planar*. That is, they can be embedded in the plane without their edges crossing. A *face* is connected component of the maximal subset of the plane that is disjoint from the embedded graph. The *degree* of a face is the number of vertices on its boundary. A face of degree 3 is a *triangular* face. The *outer face* of a graph is the one and only unbounded face. A vertex is *incident* to a face when it lies on its boundary.

Vertices bordering the outer face are *outer vertices* while all other vertices are *interior vertices*.

Angular order. For a fixed embedding of G the *angular order* at a vertex v is the clockwise order of the edges incident to v. We identify these edges with their other endpoints. Two vertices x, y are said to be *consecutive* in the angular order at v when the edges vx and vy are consecutive in the angular order.

Paths. A path $\mathcal{P}$ is a sequence of vertices such that every two consecutive vertices are connected by an edge. The first and last vertex of the path are its *extreme* vertices while the rest are *internal* vertices of this path. The *length* of a path is the number of edges used to connect the vertices. That, is one less than the number of vertices. In this thesis all paths are *simple*, that is, no vertex occurs twice in the path except possibly the extreme vertices.

Cycles. A cycle is a path whose extreme vertices coincide. Because a cycle is a path the start and end vertex are the only vertices that occurs more than once. We call a cycle of length k a *k-cycle*. A *triangle* is cycle of length 3 (i.e. a 3-cycle). By Jordan's curve theorem a cycle splits the plane into two parts, one bounded and one unbounded. We call the bounded part the *interior* of this cycle and the unbounded part the *exterior* of this cycle. Furthermore, the cycle of all vertices bordering the outer face is the *outer cycle*. We call a cycle *separating* if there are vertices in both its interior and exterior. An *interior edge* of a cycle is then an edge contained in the interior of the cycle. An *interior path* is a path connecting two distinct vertices off the cycle and whose edges are interior edges.

3 *Regular edge labellings*

Adjacency graphs of layouts. In an *adjacency graph* $\mathcal{G}(\mathcal{L})$ of a layout $\mathcal{L}$ each rectangle is represented by a vertex. In an adjacency graph we connect two vertices by an edge exactly when their rectangles are adjacent. In the *extended adjacency graph* $\mathcal{G}_{\mathcal{E}}(\mathcal{L})$ we also add 4 vertices $\mathrm{N}, \mathrm{E}, \mathrm{S}, \mathrm{W}$ (so-called *poles*) in the outer face, one associated to the north, east, south, west boundary segment of the outer rectangle, respectively. In this graph two vertices are connected if their rectangles or boundary segments intersect. If we take the *extended adjacency graph* of a layout and remove the 4 vertices corresponding to the outer face, we end up with the regular *adjacency graph* of that layout. In this setting a layout $\mathcal{L}$ is a *rectangular dual* of a graph G if we have that $G = \mathcal{G}(\mathcal{L})$.

Regular edge labellings. Given a layout $\mathcal{L}$ we can easily find its adjacency graph and thus for which graph G it is a rectangular dual. However, finding a rectangular dual of a graph G is more involved. Due to the algorithm by He [4] it is sufficient to find a *regular edge labeling* on a corner assignment of G. In this section we introduce regular edge labellings.

Regular edge labellings were first introduced by Kant and He [5]. Fusy also studied these structures [2, 3] under the name of *transversal structures*. A *regular edge labeling* is a coloring and orientation of the edges of the extended adjacency graph $\mathcal{G}_{\mathcal{E}}(\mathcal{L})$. This coloring and orientation is given by the following procedure. For every edge vw in $\mathcal{G}_{\mathcal{E}}(\mathcal{L})$ we consider whether the shared boundary of the rectangles is vertical or horizontal and we color the corresponding edge blue or red, respectively. If we color the edge blue, we orient it from the leftmost rectangle to the rightmost rectangle and if we color it red, we orient from bottom to top. We neither color nor orient the edges between the poles.

From the nature of the adjacencies in a rectangular layout we can deduce the following two rules, both illustrated in Figure 11, for a regular edge labeling.

1. (Interior vertex) In the angular order of every interior vertex we have the following subsequent non-empty sets: Incoming red edges, incoming blue edges, outgoing red edges and outgoing blue edges and these are the only sets that exist.
2. (Pole) N has only incoming red edges, E has only incoming blue edges, S has only outgoing red edges and W has only outgoing blue edges, except for the uncolored edges between the poles.

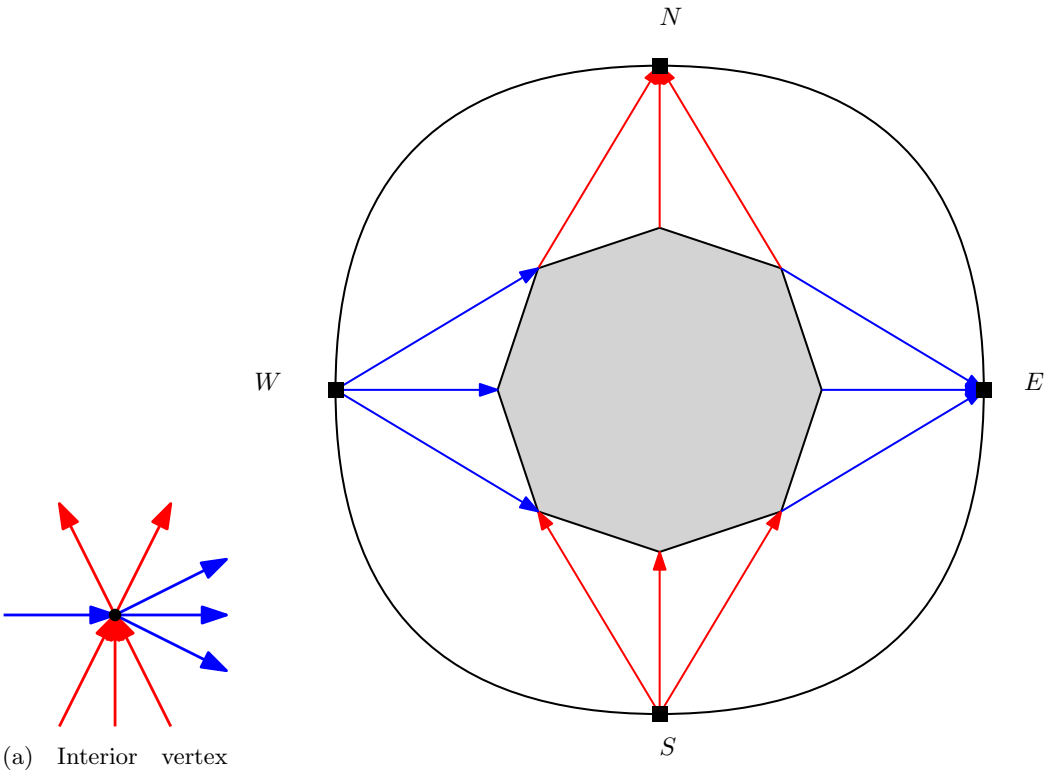
In [4] He showed that given a regular edge labeling of a corner assignment we can reconstruct a rectangular layout represented by this regular edge labeling.

Combinatorially equivalent. We say two duals of G are *combinatorially equivalent* or simply *equivalent* when they give raise to the same regular edge labeling of $\widetilde{G}$. That is, their rectangles have the same adjacencies with the same orientation (horizontal or vertical).

Properties. Since we often use regular edge labellings in this thesis we prove some properties for them. We show that a regular edge labeling has no mono-colored triangles and that the red and blue subgraph of a regular edge labeling both form a *st*-planar graph.

Observation 1 (Fusy, [2])**.** *A regular edge labeling has no monochromatic triangles even when ignoring orientation.*

Proof. Suppose we have a monochromatic triangle. Without loss of generality we suppose this triangle is blue. Then at least one of the vertices has an incoming blue edge followed directly by an outgoing blue edge or an outgoing blue edge followed directly by an incoming blue edge in its angular order. Thus, this vertex has either an empty set of outgoing or incoming red edges and hence violates the interior vertex condition of a regular edge labeling. □



(a) Interior vertex condition

(b) Pole condition

Figure 11: Regular edge labeling conditions

st-planar graphs. In this thesis we repeatedly use regular edge labellings, so it is a good idea to investigate their structure. Kant and He [5] and Fusy [2] both note that a regular edge labeling is closely linked to a pair of *st-planar graphs*. *st*-planar graphs are also known under the name of *bipolar orientations*. We state this observation in the same way it is stated by Fusy, which is slightly different from the way Kant and He state it. An *st*-planar graph is an oriented planar graph with one source (in-degree 0) s and one sink (out-degree 0) t. Both s and t lie on the outer face. Moreover, such an *st*-planar graph has no directed cycles.

Observation 2 (Fusy, [2])**.** *The blue edges of* $G \setminus \{\mathrm{N}, \mathrm{S}\}$ *form an st-planar graph with* $s = \mathrm{W}$ *and* $t = \mathrm{E}$*. Moreover, the red edges of* $G \setminus \{\mathrm{W}, \mathrm{E}\}$ *form an st-planar graph with* $s = \mathrm{S}$ *and* $t = \mathrm{N}$*.*

Proof. Note that we have no monochromatic directed cycles because such a cycle would for example correspond to a group of adjacent rectangles that have no leftmost or topmost one. By the interior vertex condition interior vertices can not be sources or sinks, this leaves the non-removed poles to be the sources and sinks, as required. □

We refer to these *st*-planar graphs as the *blue graph* and *red graph* of some regular edge labeling and we refer to their faces as *blue faces* and *red faces*. An example of such a colored extended adjacency graph with the blue and red graph can be found in Figure 12.

Every face F in an *st*-planar graph has the same structure. The boundary of F consists of two directed paths, so-called *boundary paths*, with common start vertex v and end vertex w. We say v is the *split* vertex of F and w is the *merge* vertex of F. These boundary paths are subsequent in the clockwise angular order at v. We say the first path in the angular order, starting at the beginning of the adjacent pair, is the *top boundary path* (blue graph) or *left boundary path* (red graph) of F and the second one is the *bottom boundary path* (blue graph) or *right boundary path* (red graph).

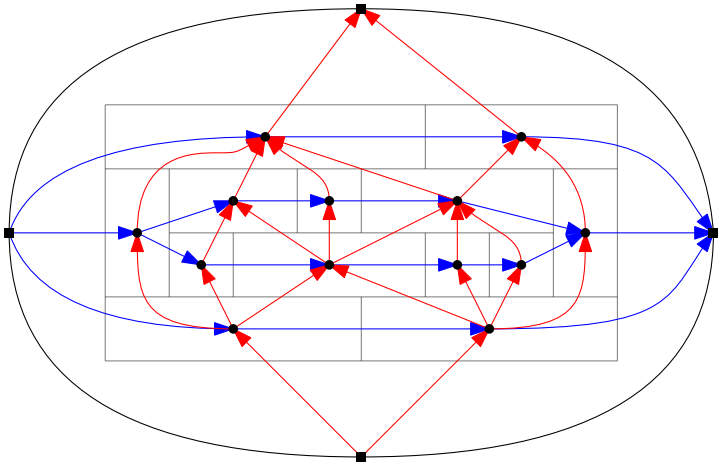


Figure 12: An example regular edge labeling with corresponding rectangular dual.

Maximal segments. Due to the way we color a regular edge labeling of $\mathcal{G}_{\mathcal{E}}(\mathcal{L})$ given a layout $\mathcal{L}$ a horizontal maximal segment corresponds to a blue face and a vertical maximal segment corresponds to a red face; as one can see in Figure 12.

Let us consider a face corresponding to a maximal segment. The number of internal vertices of both boundary paths without counting the split and merge vertices is the number of rectangles on the respective sides of the maximal segment. Hence, a one-sided maximal segment corresponds to a face with one boundary path of length 2 and a k-sided maximal segment to a face with the shortest boundary path of length at most $k + 1$. We can not have faces with a boundary path of length 1 since such a face can not enclose a valid segment.

Observation 3. *No face can have a boundary path of length* 1.

Proof. A boundary path can not have length 1 since by construction of a regular edge labeling a red or blue face encloses a maximal segment and thus has to go through at least one intermediate rectangle, and thus through at least one intermediate vertex. □

This observation generalizes the observation that we can not have monochromatic triangles.

4 *A family of graphs not* k*-sided for any* k

In this section we prove the following theorem.

Theorem 4. *There is a family of planar graphs G_k that, for any $k \in \mathbb{N}$, has members that are not k-sided.*

All members of this family G_k contain separating 4-cycles. The maximum vertex degree, d, of G_k is about $2k+O(1)$. This also means that in principle it might be possible to create an algorithm providing $O(d)$-sided layouts for any corner assignment. However, our algorithm in Section 5 only works for corner assignments without separating 4-cycles.

Multiple corner assignments. If there is no k-sided rectangular dual for a certain corner assignment $\bar{G}$ of G, there may still be another corner assignment of G that admits a k-sided dual. However, if we use $H_k = \bar{G}_k$ as an auxiliary graph we note that G_k is the interior of a separating 4-cycle of H_k. This auxiliary graph will be known as *scaffold* of G_k. We prove in Lemma 5 this implies that G_k, as induced subgraph, has to be colored in accordance with the corner assignment $\bar{G}_k$. We first prove Lemma 5 before we prove Theorem 4.

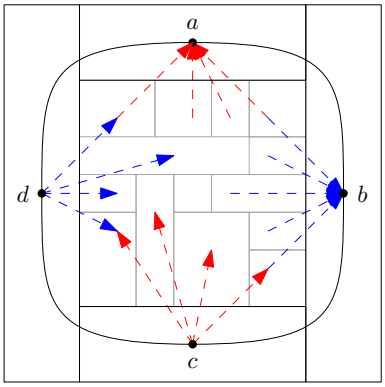


Figure 13: The structure inside a separating 4-cycle.

Lemma 5. *Let $\mathcal{C}$ be a separating 4-cycle of $\bar{G}$ with interior $\mathcal{I}$. We can label the vertices of $\mathcal{C}$ by a, b, c and d in clockwise order such that all interior edges incident to a, b, c and d are incoming red, incoming blue, outgoing red and outgoing blue, respectively.*

Proof. The interior I of $\mathcal{C}$ is represented by some rectilinear shape in every rectangular dual $\mathcal{L}$ of $\bar{G}$. Such a shape must have at least 4 clockwise right turns if we travel along its boundary in a clockwise direction otherwise the shape is not closed.

Yet, such a clockwise turn can not occur due to a single rectangle. Instead, such a turn can only occur when two rectangles are adjacent to each other. Because a 4-cycle represents only 4 pairs of adjacent rectangles, the representation of I in $\mathcal{L}$ can only have 4 clockwise turns. Hence, it must be a rectangle, the only rectilinear shape with just 4 clockwise turns.

The interior I of $\mathcal{C}$ is represented by a rectangle $\mathcal{I}$ in any rectangular dual. Since two disjoint rectangles can only be adjacent to each other at one side, $\mathcal{I}$ has four sides that need to be covered and $\mathcal{I}$ is adjacent to only four rectangles we know that every side of the rectangle $\mathcal{I}$ is adjacent to a single rectangle. We then denote by a the vertex corresponding to the rectangle above $\mathcal{I}$, b the rectangle right of $\mathcal{I}$, c the rectangle below $\mathcal{I}$ and d the rectangle left of $\mathcal{I}$. The construction at the end of the proof can also be seen in Figure 13.

Then the required coloring follows from how a regular edge labeling is obtained from a layout. □

Hence, if we know the color and orientation of one interior edge incident to a vertex of a separating 4-cycle $\mathcal{C}$ we know the color and orientation of all interior edges of $\mathcal{C}$ incident to $\mathcal{C}$.

Lemma 5 is useful because it allows us to consider a single corner assignment $\bar{G}$ of G by using it as scaffold. Suppose we want to investigate some specific corner assignment $\bar{G}$ of G with poles N, E, S and W then we consider the graph $\bar{G} = H$ as a graph in its own right, and inspect the coloring of this graph.

H admits a corner assignment $\bar{H}$ without separating triangles by connecting the new poles as indicated in Table 1. $\bar{H}$ is shown in Figure 14. G is displayed in thick lines and with filled vertices. An arbitrary corner assignment $\bar{G} = H$ is then drawn with thin lines and hollow vertices. A corner assignment of H is then drawn with dashed edges and hollow vertices.

The graph H can have more than one corner assignment but they all contain the separating 4-cycle $\mathcal{C} = NESW$. Thus, by Lemma 5 we see that, without loss of generality, the interior edges of $\mathcal{C}$ incident to N are colored incoming red, those incident to E are colored incoming blue, those incident to S are colored outgoing red and those incident to W are colored outgoing blue.

Proof of Theorem 4. Now we can consider the family of graphs G_k with the corner assignment $\bar{G}_k$ given in Figure 15. We know we only have to consider this corner assignment since we can force it using a scaffold and Lemma 5. In G_1 the dots are replaced by a single vertex, in G_2 the dots are replaced by two vertices and so on. Each member has 2 maximal separating 4-cycles. These are both marked by thick lines in Figure 15. Many of the edges in this graph have only one possible color and orientation violating the constraints of a regular edge labeling. Firstly, we color the edges incident with the poles in accordance with

NE	N	E	SE	NW
SE	S	E	NE	SW
SW	S	W	SE	NW
NW	N	W	NE	SW

Table 1: The neighbors of the new poles.

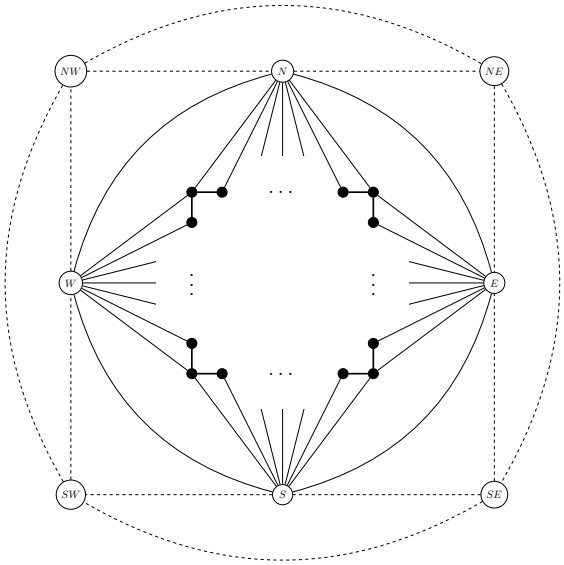


Figure 14: The construction of a scaffold.

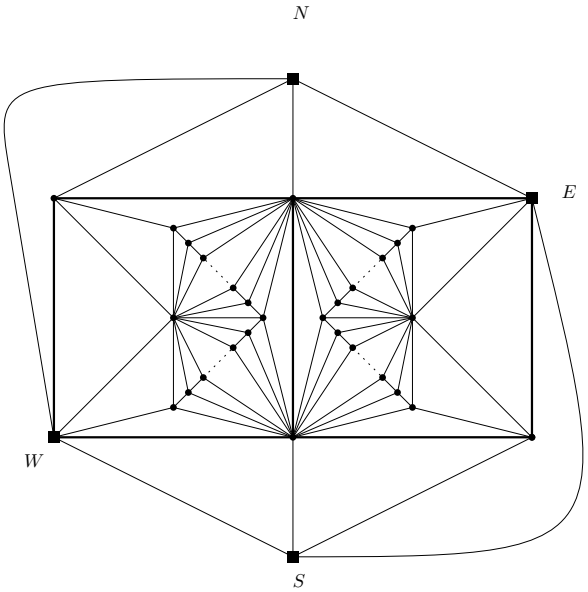
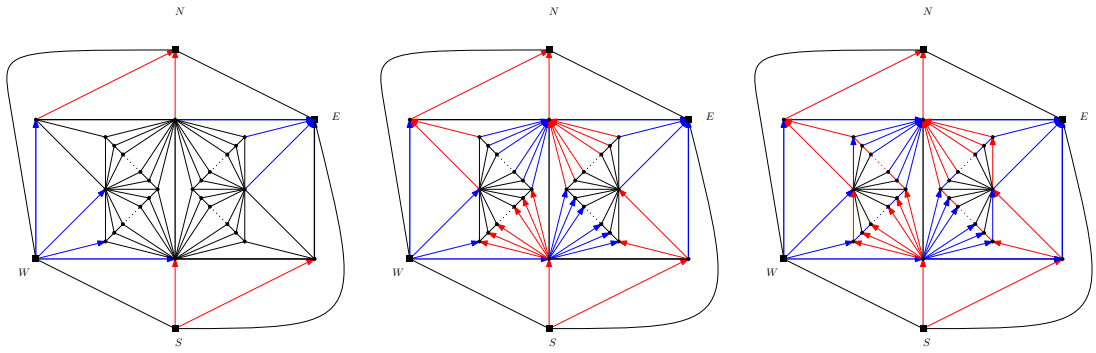


Figure 15: A family of graphs not k-sided for any k.

the exterior vertex condition. Subsequently, we can use Lemma 5 on both maximal separating 4-cycles to color even more edges and finally we can color the edges in triangles of which the other two edges have the same color using Observation 1. All these forced colorings are performed in Figure 16.

The result is then the graph displayed in Figure 17. The black edges in this figure are edges that do not have a forced coloring by the above argument (Although most of them can be forced by Lemma 5). We focus on the centered black edge e, e is an interior edge of both the red and blue faces drawn with dashed edges in Figure 17. Both boundary paths of these faces are of length larger than k. Hence, e has to be colored both red and blue to prevent that face corresponding to a k-sided segment occurs in the regular edge labeling. Since an edge can not be colored red and blue at the same time G_k is not k-sided.

Since this proof does not depend on the value of k, the family G_k has graphs that are not k-sided for any k. □



(a) Coloring the edges adjacent to the poles.

(b) Propagate through the separating 4-cycle.

(c) Colors chosen such that there are no monochromatic triangles.

Figure 16: Coloring the graph.

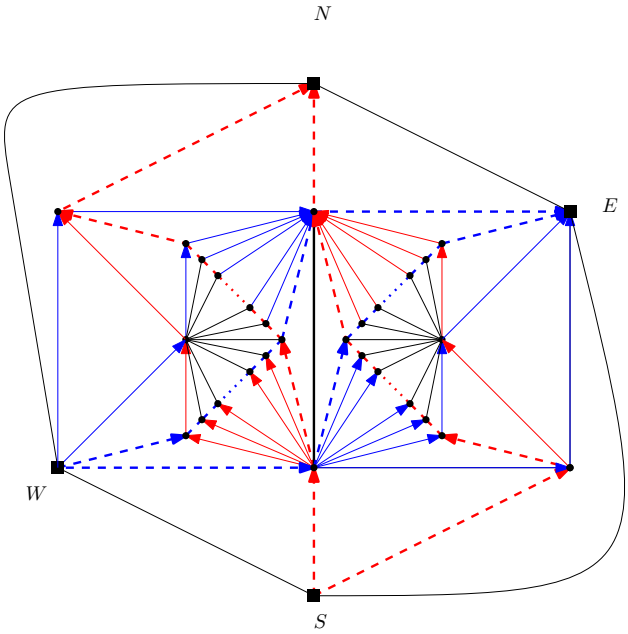


Figure 17: The graph after all coloring steps.

5 *The algorithm*

Algorithms for regular edge labellings. Kant and He [5] were the first to design algorithms that determine a regular edge labeling given a graph G. Fusy [3] developed a different algorithm computing a specific regular edge labeling using a method which shrinks a cycle while coloring the exterior of this cycle in accordance with a regular edge labeling. He refers to such a cycle as a *sweepcycle*. Unfortunately his proof of this algorithm is rather concise, leaving rather much room for the reader to fill in the details.

In this algorithm Fusy starts by denoting the outer cycle of $\bar{G} \setminus \{\mathrm{N}\}$ as the sweepcycle $\mathcal{C}$. He then shrinks this sweepcycle by updating it with interior paths of $\mathcal{C}$. During this update he maintains invariants on the structure of both the cycle and its exterior. After doing a finite amount of updates, the sweepcycle has no more interior vertices. At this point Fusy completes the algorithm and obtains a regular edge labeling.

Our algorithm. In this section we will present an algorithm providing a d-sided rectangular dual for any corner assignment $\bar{G}$ without separating 4-cycles, where d is the highest degree among vertices of G in $\bar{G}$. Hence, we will prove the following theorem.

Theorem 6. *Graphs G that have a corner assignment $\bar{G}$ without separating 4-cycles are $d-1$-sided, where d is the maximal degree of the vertices of G in $\bar{G}$.*

Our algorithm will need a number of steps to get to the stated result. In Figure 18 we can see the four main step we need for finding a $d-1$ sided layout. In every step we ensure new restrictions while maintaining the old ones. In the last step, "Blue Face Subdivision", we can then use these accumulated restrictions to build a $d-1$ sided layout.

The first step is finding a regular edge labeling with a certain restriction on *splitvertices*, vertices having more than one blue outgoing edge. We show that these splitvertices can not be next to something we call the handle of a large topfan, that is, a vertex that has more than two incoming red edges.

In the second step we will make sure that our regular edge labeling L is *vertically one-sided*. That is, all vertical segments in the corresponding rectangular layout are one-sided segments. We can

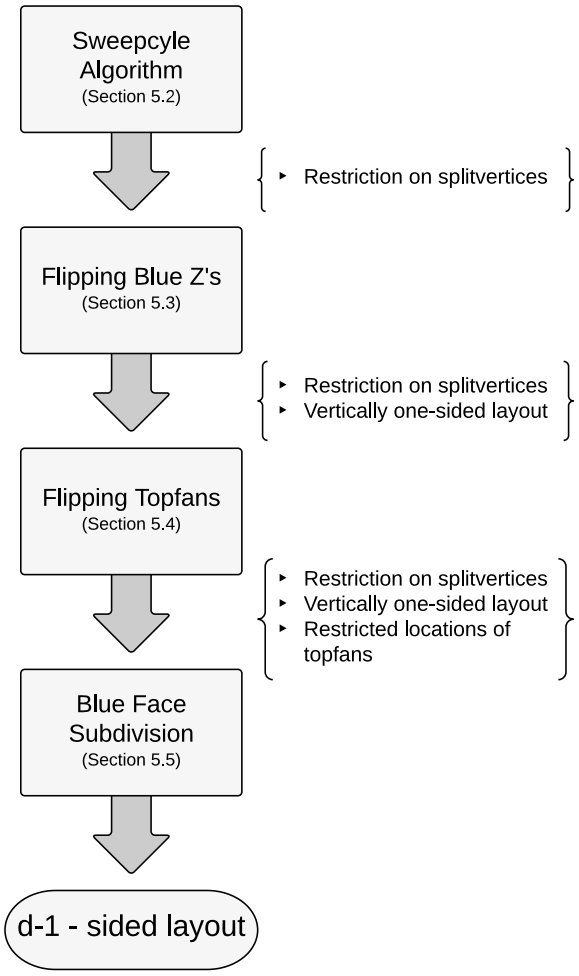


Figure 18: The flow of the algorithm.

translate this to a condition on the red faces of L, and this condition can then be translated to the existence of a structure we call a *blue* Z. In this step we resolve all occurrences of such blue Z's. We suspect that this step is not necessary because the regular edge labeling provided by the sweepcycle algorithm is already vertically one-sided. However, we were unable to prove this.

The third step then consists of handling most topfans in the regular edge labeling by "Flipping" them. A *topfan* is a several red edges going to the same vertex in a triangle strip that is formed by a blue face. Such a topfan is *large* when this has at least three incoming red edges. In Figure 19 an example of a large topfan is given by the thick edges. This is the most difficult for those topfans that are adjacent to a splitvertex and hence we have to ignore some of these. If we had been able to handle all topfans, finding a k-sided layout would have been easier.

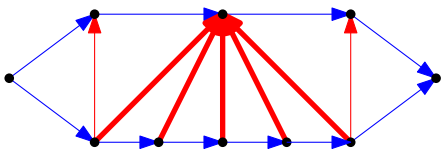


Figure 19: The thick edges are a topfan.

However, due to the restriction on split vertices we have maintained during the algorithm and the fact that all remaining topfans are adjacent to such splitvertices we are still able to produce a $d-1$ sided layout in the last step.

Overview. To describe our algorithm we introduce the notion of right neighbor paths in Section 5.1. This notion will be heavily used in the sweepcycle algorithm in Section 5.2. In the rest of the section we describe the rest of the algorithm, please refer to Figure 18 to see which steps are described in what section. In Appendix A we give an example execution of our algorithm. It can be usefull to consult this execution while reading this section.

5.1 THE RIGHT NEIGHBOR PATH OF A PATH

In the sweepcycle step of the algorithm (Section 5.2) we use the *right neighbor path* of a path. In this section we show that for any path $P = p_1 \dots p_k$ with no internal vertices incident to the outer face and without violating chords, which we will introduce shortly, on the right of the path, the right side neighbors of P are again a path Q (Lemma 9). We even show some additional properties hold for Q (Lemma 10 and 12). Similar things also hold for the left neighbors of P but we do not need this for our algorithm.

The right side of a path is not yet defined. To do so we start this section by expanding on the notion of angular orders. During the proofs in this section we also need various types of chords so we subsequently introduce these. Afterwards, we can finally discuss right neighbor paths.

Angular orders. We assume a fixed embedding for G. Recall that the order at a vertex v is the clockwise order of the edges incident to v and that two vertices x, y are said to be *consecutive* in the angular order at v when the edges vx and vy are consecutive in the angular order. Sometimes, we want to denote number of subsequent vertices, which we call an *interval*, in the angular order. We let $[x, y]$ denote all the vertices in the angular order of v from x to y and we let the *exclusive interval* (x, y) denote the same vertices without x and y. In Figure 20, for example, the interval $[v, p_{i+1}]$ consists of the vertices v, w, p_{i+1} and (w, u) consists of the vertices p_{i+1}, x.

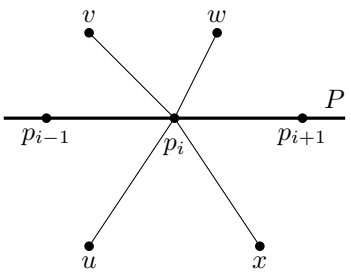


Figure 20: Angular order of p_i.

Given a path P and an internal vertex $p_i \in P$. A neighbor $v \notin \mathcal{P}$ of p_i lies on the *left* of P if it lies in the interval (p_{i-1}, p_{i+1}) in the angular order of p_i. Otherwise, v lies in the interval (p_{i+1}, p_{i-1}) in the angular order of p_i. In this case v lies on the *right* of P. We use the same notion of left and right for edges. That is, an edge $e \notin P$ adjacent to p_i lies to left or right if its other end point lies to the left or right, respectively. In Figure 20 v and $p_i v$ lie on the left of P and u and $p_i u$ lie on the right of P.

Path manipulations. With $\overleftarrow{P}$ we denote the *reversed path* $p_k \dots p_1$. We use $\oplus$ to denote the *concatenation* of paths. That is, given a second path Q with vertices $q_1 \dots q_\ell$ and $p_k = q_1$ the path $P \oplus Q$ consists of $p_1 \dots p_{k-1} q_1 q_2 \dots q_\ell$. Recall that a cycle is simply a path starting and ending at the same vertex. Hence, if we have two internally disjoint paths P, Q from s to t then $P \oplus \overleftarrow{Q}$ is a cycle. Furthermore, we use a vertical bar to denote the *restriction* of a path to a certain set of vertices. So $P|_{p_i, p_j}$ with $i < j$ is the subpath of P with vertices $p_i \dots p_j$.

Violating chords. A *chord* of a path is an edge that connects two non-subsequent vertices. A path without chords is *chordfree*. The path P in Figure 21 has the chord $p_1 p_3$. A *k-chord* is a path C of length k that connects two non-subsequent vertices p_i, p_j of P such that $P \cap C = \{p_i, p_j\}$. Note that $P|_{p_i, p_j} \oplus \overleftarrow{C}$ is a cycle. A (k-)chord C is *separating* if this cycle is separating. In Figure 21 there are two 2-chords, $p_3 u p_5$ and $p_3 v p_5$, but only one of them is separating, namely $p_3 v p_5$. The *violating chords* are given by all chords and separating 2-chords together. We use this name since we can not find a right neighbor path when we have such violating chords. In Figure 21 $p_1 p_3$ and

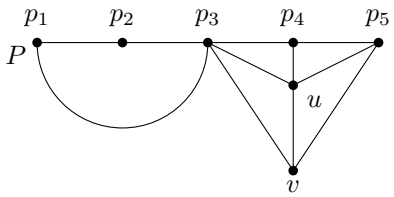


Figure 21: A path with a chord and a separating 2-chord.

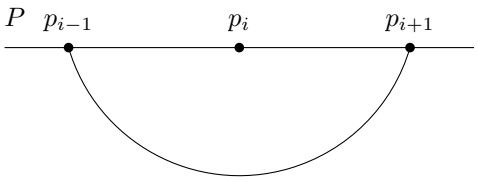


Figure 22: The hypothetical situation that p_i has no right neighbor.

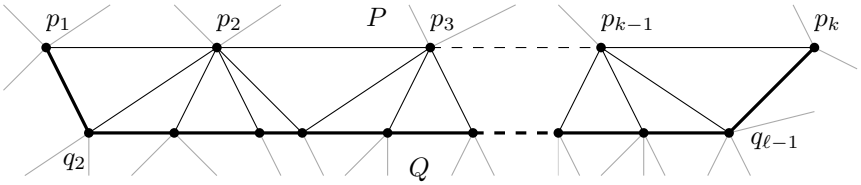


Figure 23: A right neighbor path.

p_3vp_5 are violating chords while p_3up_5 is not.

Right neighbor paths. We already mentioned that in the sweepcycle step we use the right neighbor path of a path P. Recall that we assumed P has no internal vertices incident to the outer face and no chords or separating 2-chords on the right. We first show that every vertex has right neighbors. Then we give the procedure for finding the right neighbor path. Afterwards we show that the right neighbor path is indeed a path (Lemma 9) and some additional properties in Lemmas 10 and 12.

Lemma 7. *Every internal vertex of P has at least one neighboring vertex on the right.*

Proof. Suppose that an internal vertex p_i has no neighbor on the right of the path. Then $\dots p_{i-1}p_ip_{i+1}\dots$ is a partial face border. Since p_i is not incident to the outer face, p_i must be incident to a face of degree 3. Thus, $p_{i-1}p_ip_{i+1}$ is a face. However, this would imply a chord on the right of P as can be seen in Figure 22. Hence, by contradiction p_i must have a neighbor on the right. □

The right neighbors of the internal vertices of P form the *right neighbor path* Q of P. Let us first define a larger list of vertices Q' from which we later remove vertices to get Q. Q' consists of p_1 and those vertices adjacent to p_2 that are in the interval (p_1, p_3) of the clockwise angular order of p_2. Followed by the vertices in the interval (p_2, p_4) of the angular order of p_3. We continue this up to the vertices in the interval (p_{k-2}, p_k) of the angular order of p_{k-1} and finally p_k. We get Q from Q' by removing all subsequent duplicates from Q. In Figure 23 an example of a right neighbor path is given.

To break the proof that Q is a path into two parts we define a *walk* as a path without the constraint that the same vertex does not occur twice. Hence, a walk is still a sequence of vertices that are connected to each other, but vertices may repeatedly occur.

Lemma 8. *The right neighbor path Q is a walk.*

Proof. Let us denote the vertices of Q by $q_1q_2\dots q_\ell$. Let q_i and q_{i+1} be two subsequent vertices of Q'. We show they are either connected or the same vertex. We first consider the case where $1 < i < \ell - 1$. Now there are two sub-cases. Either, (a), q_i and q_{i+1} are vertices adjacent to the same vertex p_j and thus subsequent in the angular order of p_j or, (b), q_i was the last vertex adjacent to p_j and thus q_{i+1} is the first vertex adjacent to p_{j+1} since by Lemma 7 every internal vertex of P has right neighbors. Both cases are depicted in Figure 24.

In case (a) we note that since q_i and q_{i+1} are subsequent in the angular order of p_j q_iq_{i+1} is an edge since p_j is not incident to the outer face and every interior face of G is a triangle.

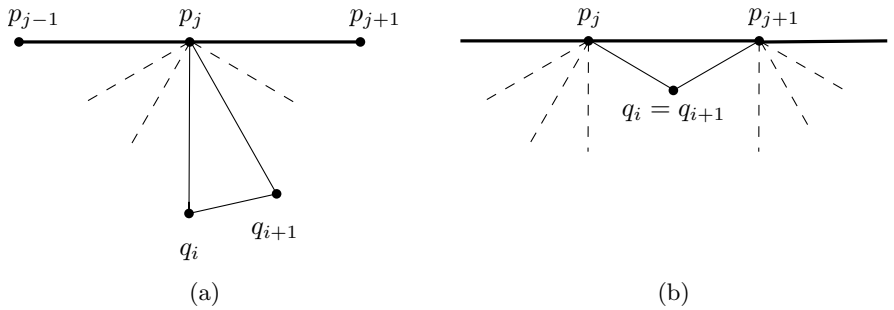


Figure 24: The two main cases of the proof showing that W is a walk.

In case (b) we note that p_iq_i and p_ip_{i+1} are edges subsequent in clockwise order, hence q_ip_{i+1} is also an edge. Hence, q_i is the first vertex adjacent to p_{i+1} subsequent to v_i in the clockwise angular order. Thus, $q_i = q_{i+1}$, that is q_i and q_{i+1} are duplicates.

Now for the cases $i = 1$ and $i = k - 1$. q_1 and q_2 are vertices adjacent to p_2 subsequent in the clockwise angular order of p_2 and hence connected since every interior face is a triangle. In the same way q_{k-1} and q_k are subsequent vertices in the angular order of q_{k-1} and hence connected. This can also be seen in Figure 23.

Since all pairs of subsequent vertices in Q' are connected or duplicates the step removing all duplicates from Q' ensures Q is a walk. □

Lemma 9. *The right neighbor path Q is a path.*

Proof. We already know Q is a walk by Lemma 8. Hence, we only have to show that Q contains no duplicate vertices.

Suppose that Q has a duplicate vertex $q_i = q_j$ with $i < j$. Then this vertex must have been a neighbor to two different vertices in P, since it can not have been added twice while being connected to only one vertex. We denote these vertices p_ℓ, p_k with $\ell < k$. This gives us the situation depicted in Figure 25. Due to the order in which we added vertices to Q', which is preserved by the removal of vertices when we go to Q, we know that any vertices in between q_i and q_j in Q must be one of the following:

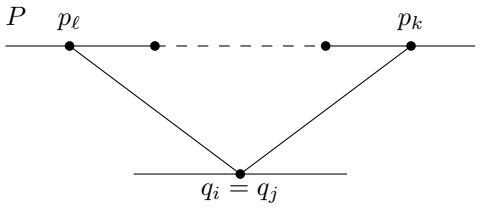


Figure 25: A hypothetical duplicate vertex.

1. Adjacent to p_ℓ and in the interval $(q_i, p_{\ell+1})$ in p_ℓ's angular order.
2. Adjacent to one of $p_{\ell+1}, p_{\ell+2}, \ldots, p_{k-1}$ and to the right of P.
3. Adjacent to p_k and in the interval (p_{k-1}, q_j) in p_k's angular order.

All three cases describe a vertex that lies in the interior of the cycle $q_ip_ip_{i+1}\ldots p_j$. However, since P has no separating 2-chords on the right this cycle must be empty. Therefore, there are no vertices in between q_i and q_j and they must be subsequent duplicates. However, Q is a walk and G is simple so Q has no subsequent duplicates. Hence, Q contains no duplicates at all and is thus, by definition, a path. □

Lemma 10. *The cycle $P \oplus \overleftarrow{Q}$ has no interior vertices.*

Proof. In the construction of the right neighbor path both cases in Figure 24 add a triangle to the interior with all vertices of the triangle in $P \oplus \overleftarrow{Q}$. Hence, the interior of $P \oplus \overleftarrow{Q}$ can be subdivided in a number triangles. Suppose there is an interior vertex in the cycle $P \oplus \overleftarrow{Q}$. Then the triangle containing this vertex is a separating triangle. Hence, $P \oplus \overleftarrow{Q}$ has no interior vertices. □

Lemma 11. *Every internal vertex q of a right neighbor path Q has a left neighbor.*

Proof. Let q be an interior vertex of Q. q Was added as right neighbor of some internal vertex p of P. Since Q does not start or end at p, p must also be a left neighbor of q in Q. □

Lemma 12. *The left of a right neighbor path is chordfree.*

Proof. Suppose that the right neighbor path $Q = q_1 \dots q_k$ has a chord on the left, say between q_i and q_j with $i < j - 1$. Then $q_i q_j$ is an interior edge of $P \oplus \overleftarrow{Q}$. There is a vertex $p_\ell \in P$ such that q_{i+1} is a right neighbor of p_ℓ, hence $p_\ell q_{i+1}$ is an interior edge of $P \oplus \overleftarrow{Q}$. But now we have a crossing between $q_i q_j$ and $p_\ell q_{i+1}$ since both edges run in the interior of $P \oplus \overleftarrow{Q}$ and their endpoints occur alternately in $P \oplus \overleftarrow{Q}$. See figure 26. Since we assume G is planar, there can be no chords on the left of the right neighbor path. □

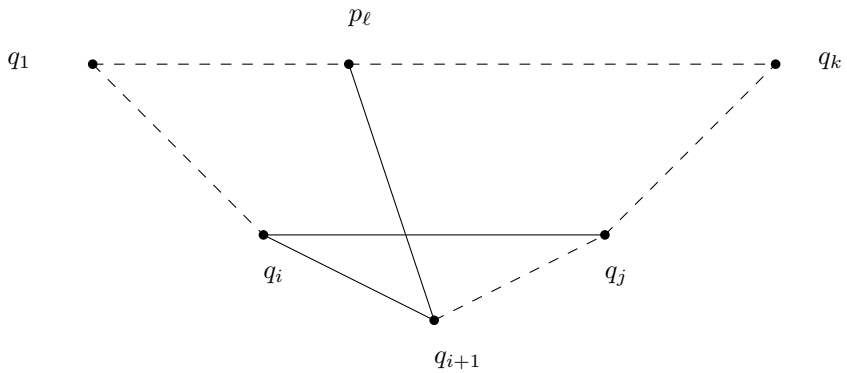


Figure 26: The construction in the proof of Lemma 12.

5.2 SWEEPCYCLE ALGORITHM

The first step of our algorithm is executing a sweepcycle algorithm inspired by, but different from, the sweepcycle algorithm by Fusy [3]. We use $\mathcal{C}$ to indicate the current sweep cycle. We shrink $\mathcal{C}$ by updating it with interior paths. The algorithm finishes when $\mathcal{C}$ has no more interior vertices. When the algorithm finishes, it has produced a regular edge labeling. One of the nicest things about the regular edge labeling is Lemma 18. This lemma states that we can not have the fan handle of a large topfan after a split vertex on a so-called bottom path.

During the algorithm we maintain several invariants on $\mathcal{C}$. The first four are equivalent to those imposed by Fusy. The final invariant is new and allows us to prove Lemma 18.

Invariants 13

(I1) The cycle $\mathcal{C}$ contains the two edges SW and SE.
(I2) $\mathcal{C} \setminus \{\mathrm{S}\}$ has no chords.
(I3) The inner vertex condition holds for all vertices in the exterior of C.
(I4) Every non-pole vertex on the sweepcycle has a red outgoing edge.
(I5) $\mathcal{C} \setminus \{\mathrm{S}\}$ has no separating 2-chords that do not use S.

We initialize the sweepcycle $\mathcal{C}$ with the outer cycle of $\bar{G}$. We denote the vertices of the sweepcycle $\mathcal{C}$ by $\mathrm{S}, v_1 = \mathrm{W}, v_2, \ldots v_{n-1}, v_n = \mathrm{E}, \mathrm{S}$. We repeatedly consider the path $\mathcal{C} \setminus \{\mathrm{S}\}$. In which case we order it from W to E. That the edges SW and SE are always in $\mathcal{C}$ is a result of Invariant 13 (I1).

Each update of the sweepcycle consists of the following three steps.

1. Take the right neighbor walk of a subpath of $\mathcal{C} \setminus \{\mathrm{S}\}$ to get the *candidate path* P.
2. Evade violating chords on P to get the *updating path* P'.
3. Update the sweepcycle with P'.

We repeat these steps until the sweepcycle does not contain anymore interior vertices. At that point we can terminate the algorithm by coloring the edges of the cycle $\mathcal{C}$ blue and its interior edges red.

5.2.1 *Find the right neighbor path*

Recall that we denote the vertices of $\mathcal{C} \setminus \{\mathrm{S}\}$ by $\mathrm{W} = v_1 v_2 \ldots v_{n-1} v_n = \mathrm{E}$. Suppose they are all adjacent to S, then any vertex still in the interior of $\mathcal{C}$ would lie in a separating triangle of G. So we have no interior vertices and hence we can terminate the algorithm as described in Section 5.2.4. In the remainder we assume some vertices from $\mathcal{C} \setminus \{\mathrm{S}\}$ are not adjacent to S.

Since $\mathcal{C} \setminus \{\mathrm{S}\}$ has some vertices incident to S (at least W and E) and some that are not, we can consider maximal subpaths of $\mathcal{C} \setminus \{\mathrm{S}\}$ consisting of vertices adjacent to S. We denote by v_i the last vertex of first maximal subpath of vertices adjacent to S and by v_j the first vertex of the second maximal subpath. As candidate path P we take the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_i, v_j}$. This right neighbor path does indeed exist since all internal vertices of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_i, v_j}$ are interior vertices of G and $\mathcal{C} \setminus \{\mathrm{S}\}$ has no violating chords by Invariants 13 (I2) and 13 (I5). This situation is depicted in Figure 27.

5.2.2 *Evading violating chords*

Recall that a violating chords on the candidate path P can be one of the following two things:

1. Chords
2. Separating 2-chords

The *middle* vertex of a 2-chord is the only internal vertex of that 2-chord (seen as a path). All violating chords are on the right of the candidate path due to Lemma 12 (no chords on the left) and Lemma 10 (no separating 2-chords on the left).

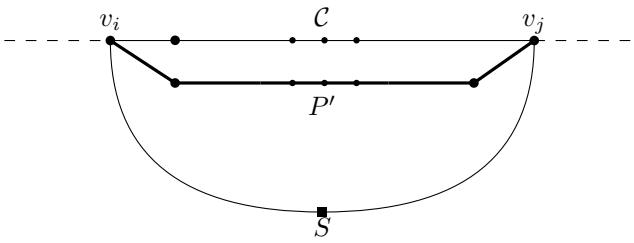


Figure 27: Updating path when P contains no violating chord.

Before we can show how to evade these structures, we first introduce more notation. We orient P from $v_i \in \mathcal{C}$ (the vertex closest to W) to $v_j \in \mathcal{C}$ (the vertex closest to E) and denote its vertices by $p_1 \dots p_r$. The *index* of a vertex $p_m \in P$ is its position in the path, that is, the index of p_m is m. The *start index* of a violating chord C is the index of the first vertex in P that is also in C. Similarly, the *end index* is the index of the last vertex in P that is also in C. The *range* of a violating chord is given by its start and end index. We update the sweepcycle with some *update path* P', depending on the violating chords we find on the candidate path P. This update is described in Section 5.2.3.

While describing how the updating path P depends on the violating chords of P, we show that the following two lemmas hold in every case.

Lemma 14. *The updating path has no violating chords*

Lemma 15. *There are no violating chords, not containing* S *as middle vertex, with one endpoint on the sweepcycle* $\mathcal{C}$ *and one endpoint on the updating path* P'*.*

We have no violating chord. When P has no violating chords, we update the sweepcycle with the entire candidate path P. In this case the update path P' is equal to P.

P' has no violating chord by the definition of this case. Moreover, there are no violating chords with one endpoint on P' and one endpoint on $\mathcal{C}$ since v_i and v_j are both adjacent to S, so we can not have any chords and any 2-chords must have S as middle vertex.

We have a chord on P. Note that we can not have a chord incident to one of the exterior vertices of the candidate path P, that is p_1 or p_r, since any such chord would violate Invariant 13 (I5) of $\mathcal{C}$ as can be seen in Figure 28.

We identify the chords by their ranges. Of the chords with the smallest end index m we consider the one with the largest start index n. We denote this chord by C. Note that this chord can not contain any other chords since such a chord would have either a large start index or a smaller end index. The way in which we find this chord C is illustrated in Figure 29.

What we do now depends on whether a separating 2-chord shows up in the interior of the chord concatenated to the candidate path $P|_{m,n} \oplus \overleftarrow{C}$.

No separating 2-chord. If there is no separating 2-chord in the interior of $P|_{i,j} \oplus \overleftarrow{C}$, we let v_k be the shared neighbor in $\mathcal{C}$ of p_m and p_{m+1} and we let v_ℓ the shared neighbor in $\mathcal{C}$ of p_{n-1} and p_n. The updating path P' is the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}\,|_{v_k,v_\ell}$. See Figure 30.

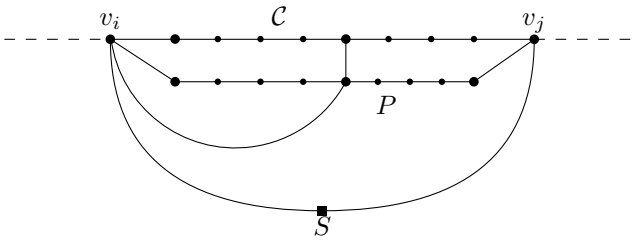


Figure 28: Hypothetical situation where P would have a chord on an exterior vertex.

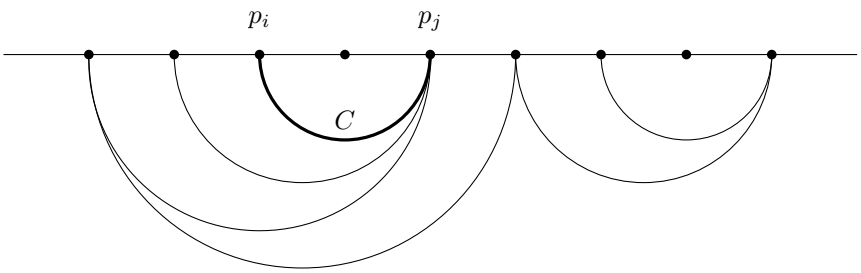


Figure 29: Finding the chord C.

P' is entirely inside a chord containing no more chords and thus can not contain a chord. Moreover, there are no separating 2-chords on $P|_{p_m,p_n}$ so P' can not have separating 2-chords. So Lemma 14 holds in this case. Any violating chord with one endpoint in P' and one in $\mathcal{C}$ has to cross $v_k p_m p_n v_\ell$ so we can not have a chord and any 2-chord has p_m or p_n as middle vertex. But, with this restriction such a 2-chord can not be separating. So Lemma 15 also holds in this case.

At least one separating 2-chord. Let n' be end index of the separating 2-chord with the lowest end index in the interior of $P|_{m,n} \oplus p_{\overleftarrow{m}} p_n$. And let v_k be the shared neighbor on the sweepcycle of p_m and p_{m+1} and let v_ℓ and $v_{\ell'}$ be the shared neighbor on the sweepcycle of p_{n-1} and p_n and of $p_{n'-1}$ and $p_{n'}$, respectively. Then the updating path P' is the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_k,v_{\ell'}}$. See Figure 31.

P' is entirely inside a chord containing no more chords and thus can not contain a chord. Moreover, P' can not have a separating 2-chord since we evaded the end of the first one ending. Just as in the above case we can not have any violating chord with one endpoint in P' and one in $\mathcal{C}$ outside the containing chord $v_k p_m p_n v_\ell \oplus \overleftarrow{\mathcal{C} \setminus \{\mathrm{S}\}|_{v_k,v_\ell}}$. That leaves violating chords with the second endpoint inside the containing chord $v_k p_m p_n v_\ell \oplus \mathcal{C} \setminus \overleftarrow{\{\mathrm{S}\}|_{v_k,v_\ell}}$. Suppose that we have a separating 2-chord, then this would have been a chord of P. This is in contradiction with $p_m p_n$ being a minimal chord. We also can not have a chord since this would break the cycle $P|_{p_m,p_n} \oplus p_n p_m$.

Only separating 2-chords. In this case the candidate path P has no chords, otherwise we would be in the above case. We split this case into two subcases, either we have a separating 2-chord with v_j as end vertex or we have only other separating 2-chords.

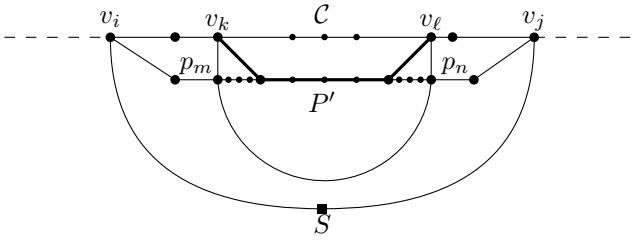


Figure 30: Updating path when P has a chord not containing a separating 2-chord.

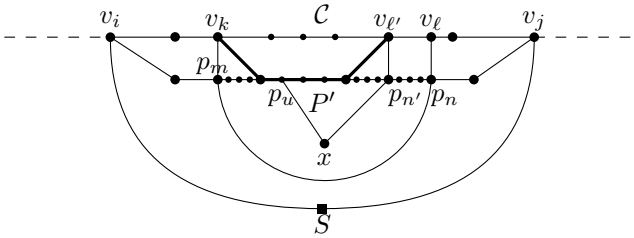


Figure 31: Updating path when P has a chord containing at least one separating 2-chord.

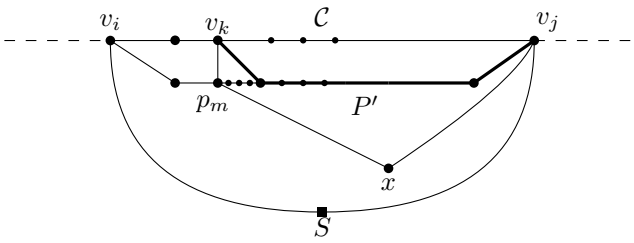


Figure 32: Updating path when P has a separating 2-chord with v_j as end vertex.

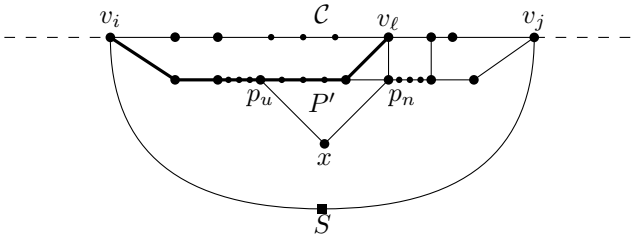


Figure 33: Updating path when P has separating 2-chords none of which have v_j as end vertex.

Any separating 2-chords with v_j as end vertex. We find the smallest separating 2-chord with v_j as end vertex (i.e. the one with the highest start index). Say this separating 2-chord has start index m. Let v_k be the shared neighbor on the sweepcycle of p_m and p_{m+1}. The updating path is the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_k,v_j}$. See Figure 32.

P' starts inside the separating 2-chord adjacent to v_j with the highest start index. P' has no chords since P already had none. However, P' can have separating 2-chords that are not adjacent to v_j. If this is the case, we find the 2-chord with the lowest end index n. Let v_ℓ be the shared neighbor on the sweepcycle of p_n and p_{n-1}. The updating path is the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_k,v_\ell}$.

We have no chords with one endpoint in P' and one in $\mathcal{C}$ since these would have to break $v_j x p_m v_k \oplus \overleftarrow{P'}$ or be adjacent to v_j, which is in violation of Invariant 13 (I5). Any 2-chords with one endpoint in P' and one in $\mathcal{C}$ would have x or any vertex in P as middle vertex. However, the first yields a 2-chord of the candidate path with a higher start range, this is a contradiction. And the second gives a chord of P, this is in contradiction with the current case where P has no chords.

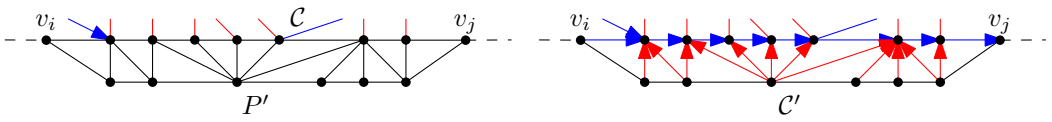
Only other separating 2-chords Find the 2-chord with the lowest end index, say that this is n. Let v_ℓ be the shared neighbor on the sweepcycle of p_n and p_{n-1}. The updating path is the right neighbor path of $\mathcal{C} \setminus \{\mathrm{S}\}|_{v_i,v_\ell}$. See Figure 33.

Any updating path stops before the end of a separating 2-chord and furthermore contains no chords since P already did not.

Any violating chord with one vertex in the updating path and the other on the old sweepcycle must end to the right of the updating path since the updating path starts at v_i, a vertex adjacent to S. Suppose that we have a separating 2-chord then that would have been a chord of the candidate path. This is in contradiction with the assumption that the candidate path had no chords. We also have no chord since such a chord would violate Invariant 13 (I5) of the old sweepcycle. Furthermore, the second-to-last vertex of the updating path has no chords since it would break $v_\ell p_n x p_u$. (see Figure 33).

5.2.3 *Updating*

Once we found the updating path P', we can update the sweepcycle with this path. Let p_a and p_b indicate the two unique vertices of P' that are also part of $\mathcal{C}$. We then let $\mathcal{C} \setminus \{\mathrm{S}\}|_{P'}$ denote the path $\mathcal{C} \setminus \{\mathrm{S}\}|_{p_a,p_b}$. In this section we describe how to update the sweepcycle with an updating path and we show that the update maintains all sweepcycle invariants (Lemma 16). To execute



(a) Before. (b) After.

Figure 34: The update.

the update we color all interior edge of $\mathcal{C} \setminus \{S\}|_{P'} \oplus \overleftarrow{P'}$ red and orient them towards $\mathcal{C} \setminus \{S\}|_{P'}$. We also color all edges of $\mathcal{C} \setminus \{S\}|_{P'}$ blue and orient them from the lower to the higher indices. We then update the sweepcycle to $\mathcal{C}'$ by replacing $\mathcal{C} \setminus \{S\}|_{P'}$ by P' in $\mathcal{C}$. An example of the whole update for an updating path P' can be seen in Figure 34.

Lemma 16. *Updating with a path P' maintains all sweepcycle invariants.*

Proof. Invariant 13 (I1) remains true. Invariant 13 (I3) holds due to the way we colored the edges around the new interior vertices as can be seen in Figure 34. Furthermore, Invariant 13 (I4) holds because every internal vertex of P' has a left neighbor by Lemma 11.

To see that Invariants 13 (I2) and 13 (I5) hold, note that there can be no violating chords with both endpoints in the overlap of the old and new sweepcycle $\mathcal{C} \cap \mathcal{C}'$ by Invariants 13 (I2) and 13 (I5). Since the updating path itself also has no violating chords (Lemma 14), we know any violating chord C has to have one vertex on $\mathcal{P}$ and one vertex on the unchanged part of old sweepcycle $\mathcal{C} \cap \mathcal{C}'$. However, these potential chords can not exist by Lemma 15. Hence, $\mathcal{C}'$ is a valid new sweepcycle. □

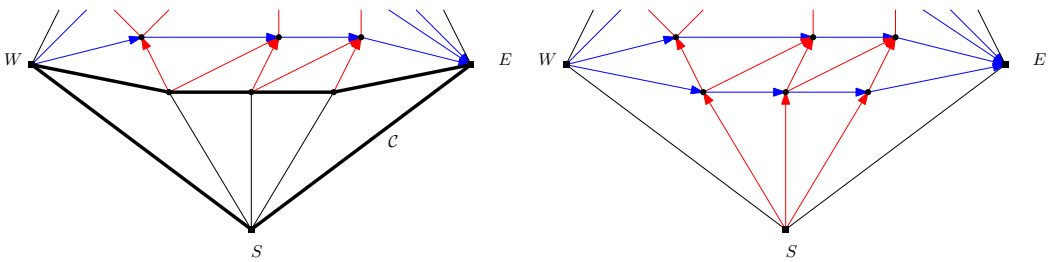
If after the update the new sweepcycle $\mathcal{C}'$ has no interior vertices, we terminate the algorithm; this is described in Section 5.2.4. Otherwise, we start the update loop again by finding a new candidate path.

5.2.4 *Terminating the algorithm*

When the sweepcycle has no more interior vertices, we can not update it anymore. However, at this point it is easy to color the remainder of the graph. All vertices in $\mathcal{C} \setminus \{S\}$ must be adjacent to S since SW and SE are part of $\mathcal{C}$ by Invariant 13 (I1), $\mathcal{C} \setminus \{S\}$ has no chords by Invariant 13 (I2) and $\mathcal{C}$ does not contain any interior vertices. All sweepcycle interior edges are adjacent to S, since otherwise we would have a chord in $\mathcal{C} \setminus \{S\}$ (violating Invariant 13 (I2)).

We color all interior edges of $\mathcal{C}$ red and orient them towards $\mathcal{C} \setminus \{S\}$ and the edges in $\mathcal{C} \setminus \{S\}$ are colored blue and oriented towards E. The termination step can be seen in Figure 35. This last move completes the interior vertex condition for vertices in $\mathcal{C} \setminus \{W, S, E\}$ and also correctly completes the exterior vertex condition.

Lemma 17. *The resulting structure is a regular edge labeling*



(a) Before. (b) After.

Figure 35: The termination step.

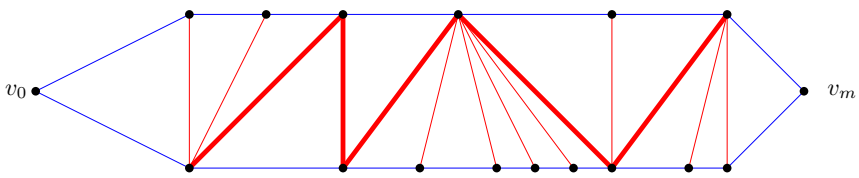


Figure 36: An example face with fans.

Proof. After running the whole algorithm the interior vertex condition holds for all vertices in the graph by Invariant 13 (I3). Furthermore, the poles are also colored correctly due to Invariant 13 (I1). □

5.2.5 *A useful property of this regular edge labelling*

There still is a useful property of this regular edge labeling left to discuss (Lemma 18). Before we can state this property, we first need to introduce fans.

Fans. We want to better describe the interior of blue (or red) faces. Every interior edge of such face goes from one boundary path to the other (otherwise its start or end vertex would violate the interior vertex condition or create a face with a boundary path of length one violating Observation 3). We now describe the edges from the split vertex to the merge vertex of F.

Let $u_0, u_1, \dots u_n$ be the vertices of the top boundary path of F and $v_0, v_1, \dots, v_m$ the vertices of the bottom boundary path. That is $u_0 = v_0$ is the split vertex and $u_n = v_m$ is the merge vertex. Since our graph is a triangulation, u_1v_1 must be an edge. For the second edge in the face we have two options, either u_1v_2 or u_2v_1, otherwise this edge and the previous one would not form a triangle. This argument holds for every subsequent edge, we can either increase the index of the top boundary path or the index of the bottom boundary path. Hence, this face is, for the readers that know this term, a *triangle strip*.

We call a maximal sequence of at least two edges increasing the index on the top boundary path (and thus keeping the index on the upper path fixed) a *bottomfan* and a maximal sequence of at least two edges increasing the index on the bottom boundary path is called a *topfan*. The *size* of such a fan is the number of edges contained in the sequence. By the definition of a fan it has size of at least 2. We use *fans* to refer to both these *types* of fans (i.e. topfans and bottomfans). We call a fan of size 3 or larger a *large fan* and a fan of size 2 a *small fan*.

The *left* of the fan is the part closest to the split vertex and the *right* of the fan is the part closest to the merge vertex.

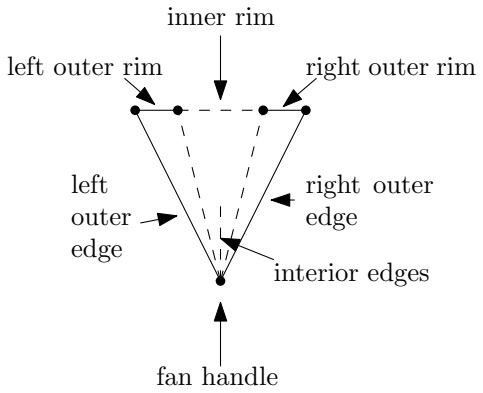


Figure 37: A number of fan-related terms.

In faces, we alternately encounter bottomfans and topfans. If we would have two adjacent fans of the same type, we would just have a single larger fan of that type. In Figure 36 we see a blue face consisting of subsequently a bottomfan of size 3, a topfan of size 2, a bottomfan of size 2, a topfan of size 6, a bottomfan of size 3 and a topfan of size 3.

We introduce some more terminology for fans: *outer edges*, *fan handles* and the *rim* as can be seen in Figure 37. The *fan handle* v is the vertex shared by all edges in the fan. Let H be the induced subgraph of vertices incident to the edges in the fan. H contains no edges not belonging to F since these would lead to separating 3-cycles. The *rim* is the path given by $F \setminus \{v\}$. The *outer rim* are the two extreme edges of this path and the *outer edges* are the edges between the fan handles and the extreme vertices of the *rim*.

A similar discussion can be given for red faces. However, then we have *right fans* and *left fans* instead of bottom and top fans.

The property.

Before finally discussing the property, we introduce one more definition. Recall a *splitvertex* is a vertex with more than one outgoing blue edge. Given a splitvertex v, the *bottom* path is the path that comes in through the first edge in the interval of incoming blue edges in the rotation at v and leaves through the last edge in the interval of outgoing blue edge in the rotation at v. See Figure 38.

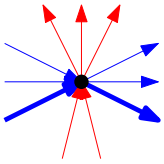


Figure 38: The bottom path of this splitvertex is given in bold.

Lemma 18. *Let v be any splitvertex. Then the subsequent vertex on the bottom path w can not be the handle of a large topfan.*

Proof. There are two possible causes of v being a splitvertex, v is either adjacent to S or v is a splitvertex due to a chord.

If v is a splitvertex because it is adjacent to S, then since w is on the bottom path it also has to be adjacent to S by the definition of the bottom path. Hence, w is not the handle of a large topfan.

If v is a splitvertex due to a chord $vabx$, we can continue the bottom path past w as a bottom path that eventually goes to x since every chord is evaded by a single path from v_k to v_ℓ in the algorithm. We denote this extended bottom path by $\mathcal{P}$. The situation is depicted in Figure 39.

The interior of $vabx \oplus \overleftarrow{\mathcal{P}}$ has no vertices. Suppose there would be such a vertex . Then, since $\mathcal{P}$ is a bottom path the blue path going through this vertex has to start at a and end at b. But this gives a blue face with only one edge on its bottom boundary path violating Observation 3. Since our graph is a regular edge labeling, $vabx \oplus \overleftarrow{\mathcal{P}}$ has no interior vertices.

This also implies all interior edges are red (by the definition of bottom path) and thus that ab is blue otherwise we would get a monochromatic triangle.

Now w can not be connected to any vertex in $\mathcal{P}$ since that would again give a face with a boundary path of length 1 by Observation 3. So w can only be connected to a and b and is thus a topfan of size at most 2. (If it is a topfan at all, since we do not consider topfans of size 1 as topfans.) □

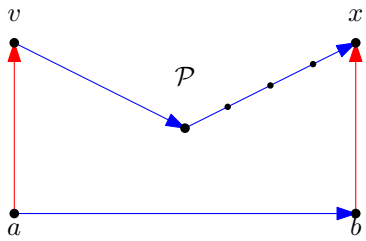


Figure 39: The situation in the proof of Lemma 18.

5.3 FLIPPING BLUE Z'S

The regular edge labeling provided by the sweepcycle algorithm of Section 5.2 is often vertically one-sided but I have not succeeded in proving that this is always the case. We would prefer to get a vertically one-sided regular edge labeling since if we then recolor edges to subdivide large blue faces it is harder to accidentally create many-sided vertical segments. In this section we modify the current regular edge labeling to make it vertically one-sided while maintaining the property of Lemma 18.

A *blue Z* is a path of three blue edges all in the same red face. A Z has a *middle* edge, this is the second edge in this path. If the current regular edge labeling is not one-sided, there must be a blue Z as is shown in Lemma 19.

Lemma 19. *A regular edge labeling is not one sided if and only if it contains a blue Z*

Proof. Consider a regular edge labeling that is not one-sided, then it contains a red face of which both boundary paths are of length at least 3. However, since the interior faces of G are triangles there must then be a blue Z in this face.

If a face contains a blue Z, it can not be one-sided. Since path of length 3 inside a red face must have at least 2 vertices on both boundary paths. □

As long as the regular edge labeling is not vertically one-sided we find such a blue Z and recolor its middle edge as in Figure 41. We call this a *flip* and we will say that this edge is *flipped*. Note that both flips transfer a valid regular edge labeling to another valid regular edge labeling. If the interior vertex condition was fulfilled in Figure 40, then it is also fulfilled in Figure 41.

We repeat these flips until the regular edge labeling is vertically one-sided. Since every flip reduces the number of blue edges by one, this is a finite procedure.

Lemma 20. *The result is a vertically one-sided rectangular edge labeling.*

Proof. We still have a regular edge labeling since the flips maintain a regular edge labeling. By construction, we flip all blue Z's. If we do not have anymore Z's, then the remaining regular edge labeling is vertically one-sided by Lemma 19. □

Lemma 21. *Let v be any splitvertex. Then the subsequent vertex on the bottom path w can not be the handle of a large topfan.*

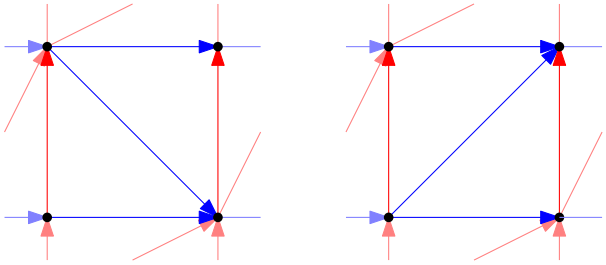


Figure 40: The two possible blue Z's.

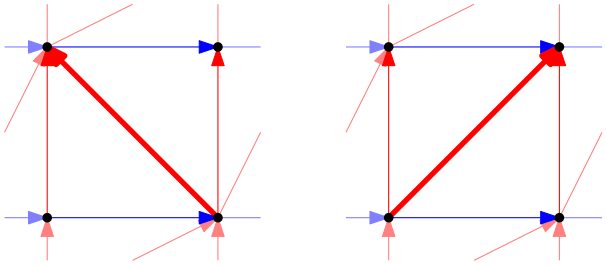


Figure 41: The flip.

Proof. This is the same statement as in Lemma 18. We will show that the operation of flipping Z's does not compromise the validity of this statement.

The flips in this section can only reduce the number of split vertices. Hence, it suffices to show that the statement still holds for all previously existing split vertices.

For a split vertex v adjacent to S we can note that the edge vw will not be flipped because it can not be a middle edge. Hence, w is still on the bottom path and still not the handle of a big topfan.

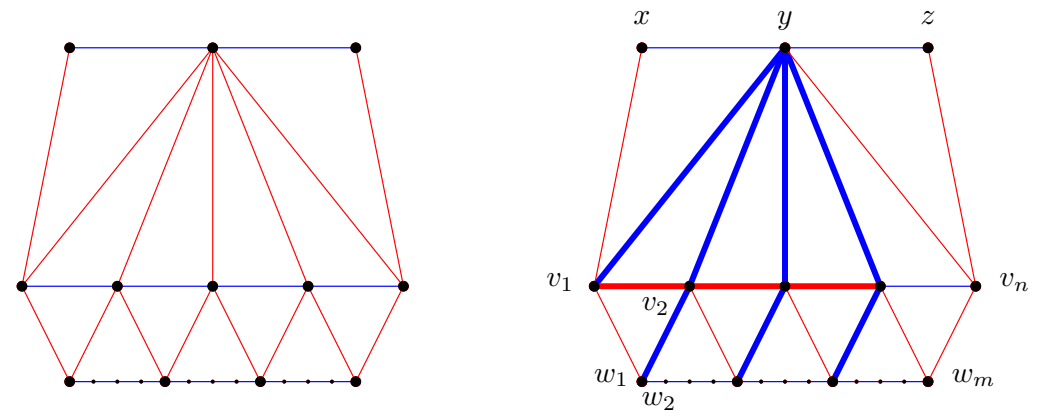
If v is a split vertex due to a chord the edges of $\mathcal{P}$ and ab in Figure 39 can not have been flipped since then we would find a monochromatic red triangle while a flip leads to another valid regular edge labeling. Hence, w is still on the bottom path through v and still can not be the handle of a large topfan. □

5.4 TOPFAN FLIPS

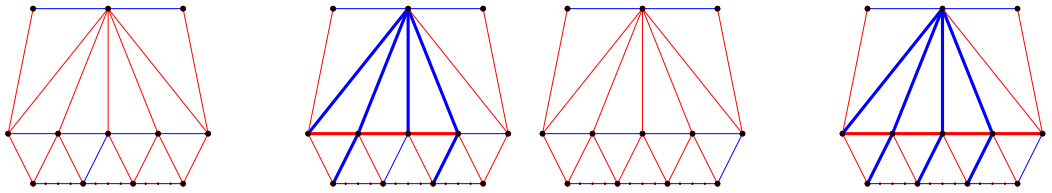
Overview. In Section 5.3, we obtained a vertically one-sided regular edge labeling (Lemma 20), moreover, this regular edge labeling never has a split vertex next to a topfan handle along a bottom path (Lemma 21). Using local recoloring (*flips*) on the topfans we will maintain a one-sided regular edge labeling (Lemma 23) and make sure that large topfans only occur in very specific situations (Lemma 24). It will turn out that we can deal with these specific situations in the final step of the algorithm described in Section 5.5. Our flips differ depending on whether we encounter a split and or merge in the bottom boundary path. Refer to Figure 42 for a first glance at the different kinds of topfan flips.

In what order do we flip topfans. We would like to start at the bottommost face F. We can unfortunately no longer use the creation order since the removal of blue Z's may have changed which faces lie above which other faces, they may even have merged faces. Hence, we have to show that there always is a face F whose whole top boundary path borders faces that are not treated while its bottom boundary path borders no such faces.

Lemma 22. *There is a face F such that the whole top boundary path of F borders faces that are not treated while the bottom boundary path does not border such faces.*

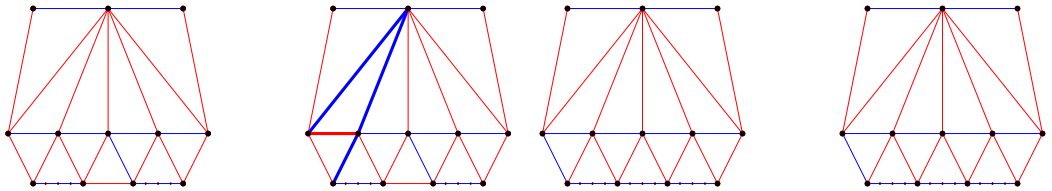


(a) The regular topfan flip.



(b) Topfan flip above a merge.

(c) Topfan flip next to merge. Note the additional red edge.



(d) Before a split we stop.

(e) If the split is on the first vertex, we do not flip at all.

Figure 42: Topfan Flips.

Proof. Let us first remove all treated faces from $\bar{G}$ by removing their blue edges and connecting their red edges with S.

Unless there are no faces remaining, and in that case we are finished, there is at least one *splitvertex* (vertex with at least two outgoing blue edges) and one *merge vertex* (vertex with at least two blue incoming edge) along the directed path formed by the vertices adjacent to S. There can be no merges before the first split, nor splits after the last merge. Because there is nowhere for these blue paths to come from or go, respectively. Hence, somewhere along the path there is a split followed by a merge. This face has a bottom boundary path that is entirely adjacent to S after removing treated faces. The top boundary path borders untreated faces. □

We keep considering the bottommost untreated face, until ther are no faces left. We know that this face is below only untreated face by Lemma 22. Since a topfan flip only affects the current face and faces below it we never have to flip in a face that is affected by the results of a topfan flip.

We do not flip topfans whose fanhandle is adjacent to the merge of the face.

How to flip a topfan. A topfan is above a number of edges of the bottom boundary path of the blue face containing the topfan. These edges are the rim of the fan. We will call a vertex S-*adjacent* if it adjacent to S.

We flip the edges of the topfan along the rim starting at the first vertex and ending at the vertex **before** the first splitvertex or S-adjacent vertex or the vertex **before** the last vertex. This can imply that we do not flip any edges (in the case that the first vertex is a split vertex or S-adjacent).

We will use the notation introduced in figure 42a. For the first vertex v_1 we recolor the adjacent outer edge of the topfan. For subsequent vertices v_i we recolor the rim edge between this vertex and the previous vertex v_{i-1} red and we recolor both edges directly adjacent to this edge in the angular order of v_i blue (if they were not already blue). If we stop flipping before a merge vertex v_{i+1}, we flip an additional edge v_iv_{i+1} along the rim, in order to prevent a blue Z from forming.

Examples. Let us show a few examples of this procedure to improve clarity. If the rim has no merges or splits, we execute the topfan flip depicted in Figure 42a. We color all but the rightmost fan edge blue, color all but the rightmost rim edge red and color the left outer edges of all topfans below this topfan in the face below the current face blue.

If the rim consists of merges and regular vertices, we easily adept a topfan flip to this situation. We simply do not flip the edge merging in as depicted in Figure 42b. A special case is given by a merge on the last vertex on the bottom edges of the topfan. In that case we flip all rim edges (even the last one) to prevent a blue Z from forming. See figure 42c.

Splits are more difficult to handle. We are unfortunately unable to keep flipping once we hit a split hence we stop before we get that far. See Figure 42d. It this happens on the first vertex we do not flip at all, see Figure 42e.

The result. Before the topfanflips, we had a vertically one-sided regular edge labeling. Afterwards we still have a vertically one-sided regular edge labeling, as we will prove in Lemma 23. Moreover, we have no large top-fans except for some controlled cases (Lemma 24).

Lemma 23. *The regular edge labeling is still vertically one-sided after a topfan flip.*

Proof. We take another look at Figure 42. Note that due to Lemma 19 a regular edge labeling is vertically one-sided as long as there is no blue Z. Since the graph was one-sided we can assume that we had no blue Z's. Let us first consider the regular case. Since the edge v_nw_m is red, (otherwise we would have a merge) this change does not produce any blue Z's.

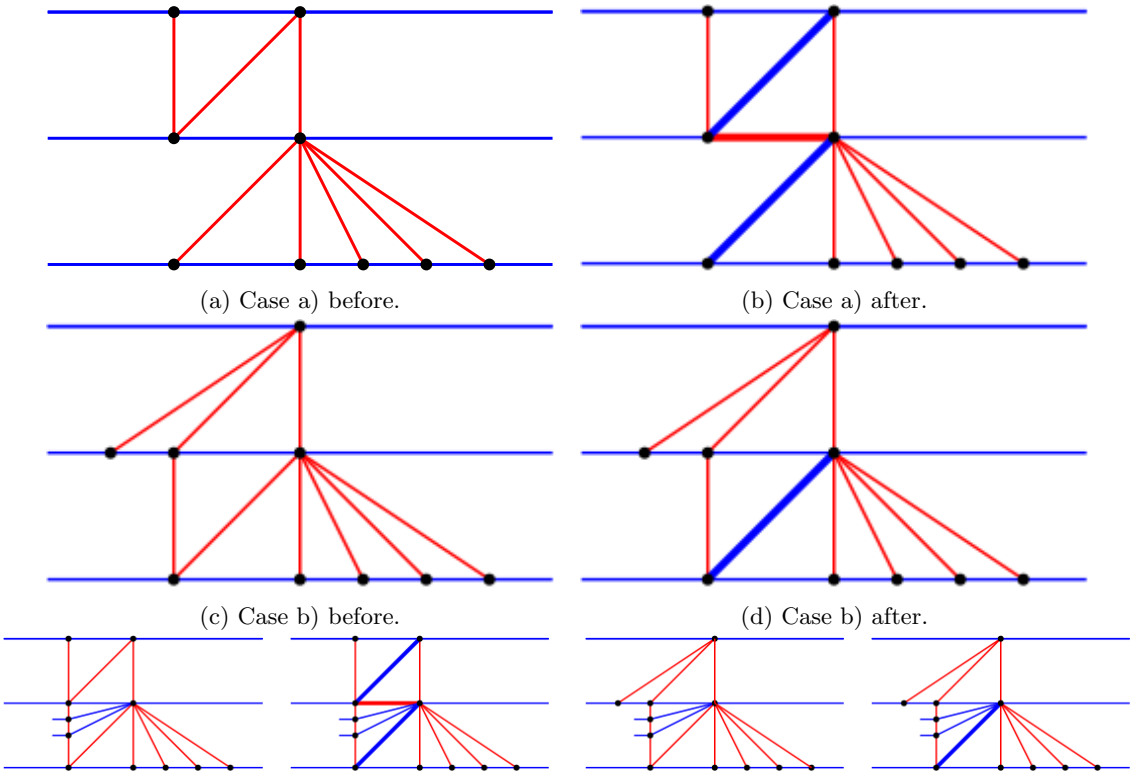
The merge case, due to the clever recoloring, also does not lead to a blue Z. It is clear the split cases do not produce a blue Z either. Since any S-adjacent fan is treated like a split fan, we also do not create Z's in these cases. □

Lemma 24. *In the remaining faces every large topfan is in one of the following two situations:*

1. *This topfan is at the start of the face.*
2. *The left outer rim vertex is a splitvertex.*

Proof. All topfans are manipulated in such a way that they start a new face, or are colored blue entirely, unless the left outer rim vertex is a split. Since in that case we do not flip at all, but then the left outer rim vertex is indeed a split. □

An unfortanate note. After submitting my thesis we discovered that this step is actually incorrect. We say that no blue Z's are formed. Unfortunately, this is not true as can be seen in Figure 42. Instead on has to find another set of flips that maintain the same properties. That is, eliminating all topfans except for those we can handle in the last step. We propose the flips in Figure **??**. But finding the exact working setup is unfortunately left as an exercise for the reader.



(a) Case a) before. (b) Case a) after.

(c) Case b) before. (d) Case b) after.

Figure 43: Topfan Flips.

5.5 BLUE FACE SUBDIVISION

At this point we have a vertically one-sided graph (due to Lemma 23) without large topfans except for the locations provided in Lemma 24. In this section we are going to recolor edges in blue faces to make all of them $d-1$-sided, while at the same time not recoloring so many edges above each other that we create a large red face.

We would like to start at the bottommost face F. Due to Lemma 22 we know that there is always a face whose whole top boundary path of borders faces that are not treated while no part of the bottom boundary path borders such faces.

We now recolor some of the edges of this face if it is too large. We then mark the edges on the top boundary path of this face above the recolored edges as *loaded*. We try to avoid flipping above these edges in future iterations of the algorithm. Then we continue with the next face in the creation order.

Loads. As is mentioned above we mark some edges with so-called *loads*, we refer to these edges as *loaded* in the rest of the section. The exact use of these loaded edges becomes clear in the rest of this section.

It is important to note that if we load any blue edge we regard any other blue edge sharing at least one vertex with this edge to be loaded as well. The occurrence of this phenomenon is called *putting through a load*. An example can be seen in Figure 44 where we flipped the thick edge, and hence marked uv as loaded and because of putting through load also marked uw as loaded.

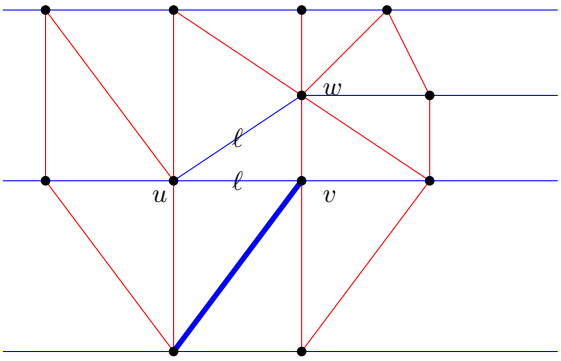


Figure 44: Putting through load.

Step requirements. We flip edges in each face, taking into account loads on the bottom boundary path, such that:

1. We never load the two edges next to a split or merge vertex on the top boundary path.
2. We never load two adjacent edges on the top boundary path.

If we flip edges in line with the step requirements for every face, then the following lemma holds for the bottom boundary path of yet untreated faces.

Lemma 25. *On the bottom boundary path of a face we never find two subsequent loaded edges. Even when we put through loads on splits and merges.*

Proof. A single face would never load two subsequent edges. Hence, the only way to get two subsequent loaded edges is using different faces and thus splits and merges. However, due to never flipping the two edges next to a split or merge we neither get subsequent loaded edges in such a case. □

5.5.1 *Faces without large topfans in the middle*

Lemma 26. *We can subdivide any blue face without large topfans into $d-1$-sided chunks while obeying the load rules above.*

Proof. A worst case example is given in Figure 45. Note that we can flip to the right edge above each edge in the bottom boundary path except if we would end up next to the merge. In that case we flip the left edge above this edge.

We look at the vertex on the bottom fence that is incident to the freshly flipped edge, or if we have not flipped an edge yet the vertex next to the split (and we denote it by v). The following are then the rules for flipping above the edges following v.

1. We do not flip above the edges of the first topfan.
2. We flip above the second edge if it is unloaded.
3. Otherwise, we flip above the third edge.
4. We never flip next to the merge.

When flipping above an edge, we always flip the right edge above that edge. Unless we are on the edge next to the merge, because then we flip the left edge above that edge.

The first edge give us the required separation of loaded edges along the top boundary path. The other items make sure we obey the other rules.

The worst case is given by a large topfan at the start and a combination of the last two items. We would in that case want to flip above the second-to-last edge of the bottom boundary path. But we do not because the next edge is incident to the merge vertex. This gives at worst a topfan and two more edges along the bottom boundary path and hence a $d-3+2 = d-1$-sided face. □

See Figure 46 for a sample execution of the algorithm described in Lemma 26.

5.5.2 *Face encountering a larger topfan*

If we have a large topfan in the middle of the face, then above the left outer edge of this topfan we can not have another topfan that failed to flip its left outer edges due to Lemma 21. This means we can use the following rule: we flip the first edge of a topfan even above a loaded edge. We call such flip a *forced* flip.

We can not have two such forced flips above each other because that would give a situation as in Figure 47. However, that would mean the fan with fanhandle u must be the handle of a topfan that failed its flip and hence v must have been a split vertex. But then by Lemma 21 w can not be the handle of a large topfan.

Since we do not allow a flip above a load (whether it was forced or not) this means that the worst thing that can happen is an ordinary flip followed by a *forced* flip. These two flips can not be followed by any other flip. Hence, the worst case only makes chains of at most 2 blue Z's, that is, a blue path of length 5.

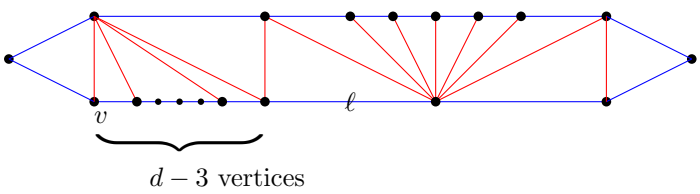


Figure 45: A worst case blue face. We do not flip any edge in this face.

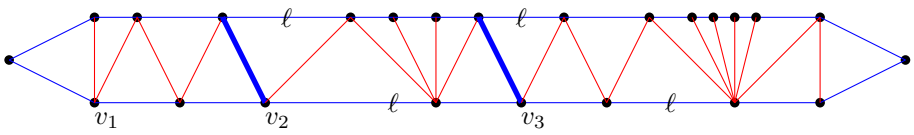


Figure 46: Sample execution of the algorithm.

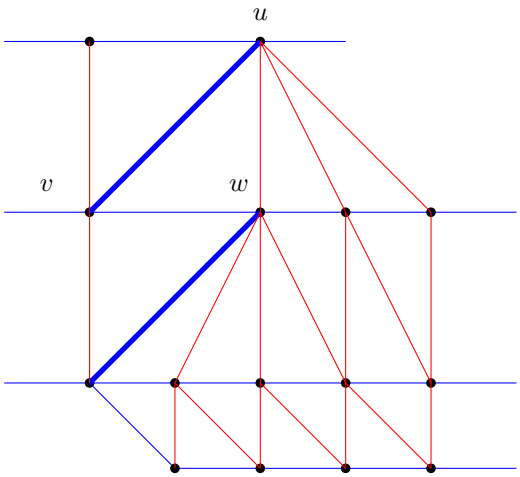


Figure 47: Two forced flips above each other.

5.5.3 *Conclusion*

This concludes the last step of the algorithm. Now all the steps in the algorithm are done all that is left is to show that we indeed generated a $d-1$-sided regular edge labeling.

Lemma 27. *Two blue flips above each other give at worst a red $d-1$-sided face*

Proof. With a *blue fan* we mean a vertex that has multiple incoming or outgoing blue edges on the interior of this red face. The two blue flips above each other give a blue path $\mathcal{P}$ of length 5 inside a red face.

Before creating the Z's in this section, the regular edge labeling was vertically one-sided. That is, before recoloring the two edges in this section there were no paths of length 3 inside the face. This also implies that any Z we now create can have at most one blue fan on the top and one blue fan on the bottom, otherwise we would already have had a 3-path.

So for two Z's we have at most three blue fans. Hence, on one side we have at most one of these. Then the boundary path at this side of the face has at most $d-3+1+1 = d-1$ vertices not counting the split and merge vertex of the red face. □

Then we can now prove Theorem 6.

Proof of Theorem 6 . By construction all blue faces are $d-1$-sided. We have at most two blue flips above each other so red faces are $d-1$-sided by Lemma 27. Hence, we have a $d-1$-sided rectangular edge labeling of $\bar{G}$ corresponding to a d-sided rectangular dual of $\bar{G}$ □

6 *Conclusions and future work*

Results. In this thesis we proved the following two theorems

1. There is a family of graphs G_k that for any $k \in \mathbb{N}$ has members that are not k-sided (Theorem 4).
2. Graphs G that have a corner assignment without separating 4-cycles are $d-1$-sided, where d is the maximal degree of the vertices of G in $\bar{G}$. (Theorem 6)

Open Questions. There are many remaining open problems. One might, for example, hope to show that all corner assignments without separating 4-cycles admit a 2-sided rectangular layout. Failing this, a proof these corner assignments, or their underlying graphs, have a k-sided layout for any constant $k \in \mathbb{N}$ would already be nice.

Another challenging problem is finding an algorithm that has traction on graphs containing (nested) separating 4-cycles. For these graphs there might be an algorithm that gives a d-sided layout, since the family of graphs provided in the proof of Theorem 4 has unbounded maximal degree. However, Theorem 4 prevents us from finding an algorithm generating a k-sided layout for all graphs admitting a rectangular layout, for any constant k.

I also wonder whether the second step in our algorithm, Flipping blue Z's, is necessary since I suspect that result of the sweepcycle algorithm in the first step is vertically one-sided.

Sweepcycle algorithms. I feel the full potential of sweepcycle algorithms is not yet unlocked in this work and the work by Fusy [3]. Since the current algorithm needs many steps to be able to repair those blue faces that are too large, it might be beneficial to concentrate more on the shape of the blue faces, the horizontal segments in the rectangular dual, in the initial sweepcycle algorithm.

One might try to build a better sweepcycle algorithm by directly finding blue faces with short enough boundary paths, hopefully without creating large red faces during its execution by choosing the right invariants. To obtain these invariants one could try to first find a sweepcycle algorithm that provides a regular edge labeling that is horizontally, instead of vertically, one-sided (or horizontally k-sided while maintaining reasonable vertical segments). This way an algorithm obtaining a k-sided layout, for some constant $k \in \mathbb{N}$, for any corner assignment without separating 4-cycles may be possible.

Another approach would be to update the sweepcycle with paths that are not entirely blue (horizontal segments in an dual), but partially blue and partially red (corners in a rectangular dual).

On the other hand, there is also an unexplored world of algorithms working directly on the rectangular dual. Maybe some cross-pollination between such an algorithm and an algorithm manipulating a regular edge labeling will lead to success.

Acknowledgments

I would not have been able to write this thesis without my two daily supervisors Bettina Speckmann and Kevin Verbeek. They always provided suggestions and motivation when I was stuck and helped bring structure in the mess that this thesis first was. I also want to thank Jesper Nederlof and Rudi Pendavingh for making this project possible by guaranteeing that it is sufficiently mathematical. I also want to thank Otfried Cheong for writing the very usefull graphics program Ipe. Moreover, I want to thank Wouter Ligtenberg and Hans Beekhuis for reading through my thesis and finding many mistakes. Last but not least I would like to thank Kaylee, my friends and my family for being an outlet for my frustrations and sharing my joys.

References

[1] David Eppstein, Elena Mumford, Bettina Speckmann, and Kevin Verbeek. "Area-Universal and Constrained Rectangular Layouts". In: *SIAM Journal on Computing* 41.3 (Jan. 2012), pp. 537–564. DOI: 10.1137/110834032.

[2] Éric Fusy. "Transversal structures on triangulations: A combinatorial study and straight-line drawings". In: *Discrete Mathematics* 309.7 (Apr. 2009), pp. 1870–1894. DOI: 10.1016/j.disc.2007.12.093.

[3] Éric Fusy. "Transversal Structures on Triangulations, with Application to Straight-Line Drawing". In: *Graph Drawing*. Springer Science+Business Media, 2006, pp. 177–188. DOI: 10.1007/11618058_17.

[4] Xin He. "On Finding the Rectangular Duals of Planar Triangular Graphs". In: *SIAM Journal on Computing* 22.6 (Dec. 1993), pp. 1218–1226. DOI: 10.1137/0222072.

[5] Goos Kant and Xin He. "Regular edge labeling of 4-connected plane graphs and its applications in graph drawing problems". In: *Theoretical Computer Science* 172.1-2 (Feb. 1997), pp. 175–193. DOI: 10.1016/s0304-3975(95)00257-x.

[6] K. Kozminski and E. Kinnen. "An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits". In: *21st Design Automation Conference Proceedings*. Institute of Electrical and Electronics Engineers (IEEE), 1984. DOI: 10.1109/dac.1984.1585872.

[7] Erwin Raisz. "The Rectangular Statistical Cartogram". In: *Geographical Review* 24.2 (Apr. 1934), p. 292. DOI: 10.2307/208794.

[8] I Rinsma. "Nonexistence of a certain rectangular floorplan with specified areas and adjacency". In: *Environment and Planning B: Planning and Design* 14.2 (1987), pp. 163–166. DOI: 10.1068/b140163.

[9] P. Ungar. "On Diagrams Representing Maps". In: *Journal of the London Mathematical Society* s1-28.3 (July 1953), pp. 336–342. DOI: 10.1112/jlms/s1-28.3.336.

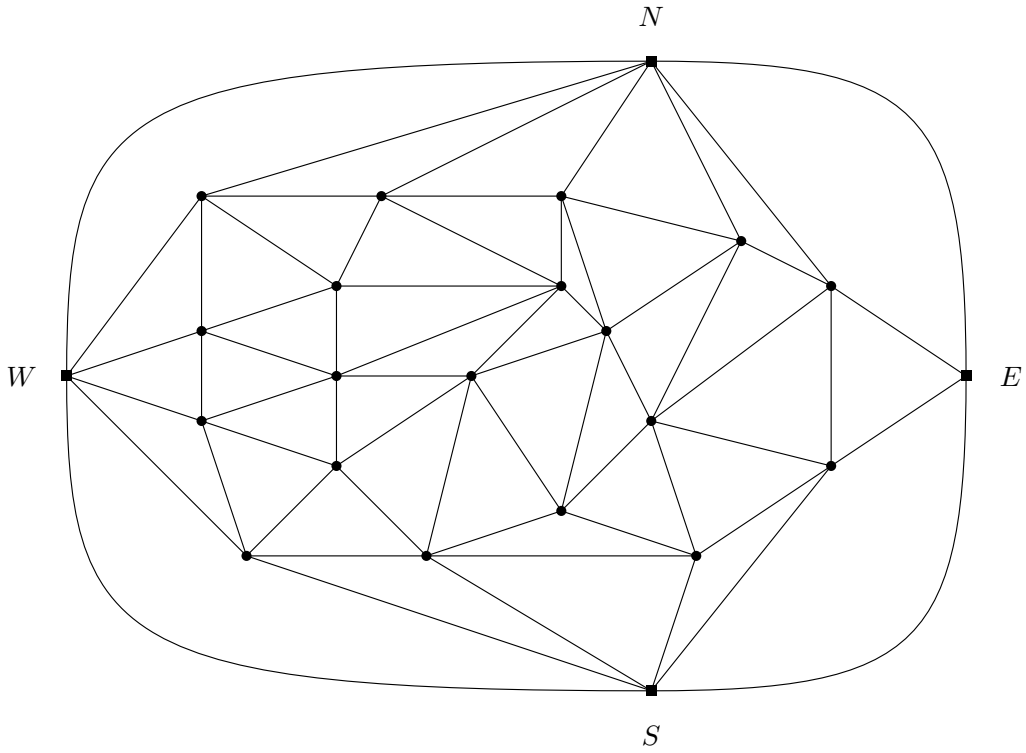
[10] Gary K. H. Yeap and Majid Sarrafzadeh. "Sliceable Floorplanning by Graph Dualization". In: *SIAM Journal on Discrete Mathematics* 8.2 (May 1995), pp. 258–280. DOI: 10.1137/s0895480191266700.

Example execution A

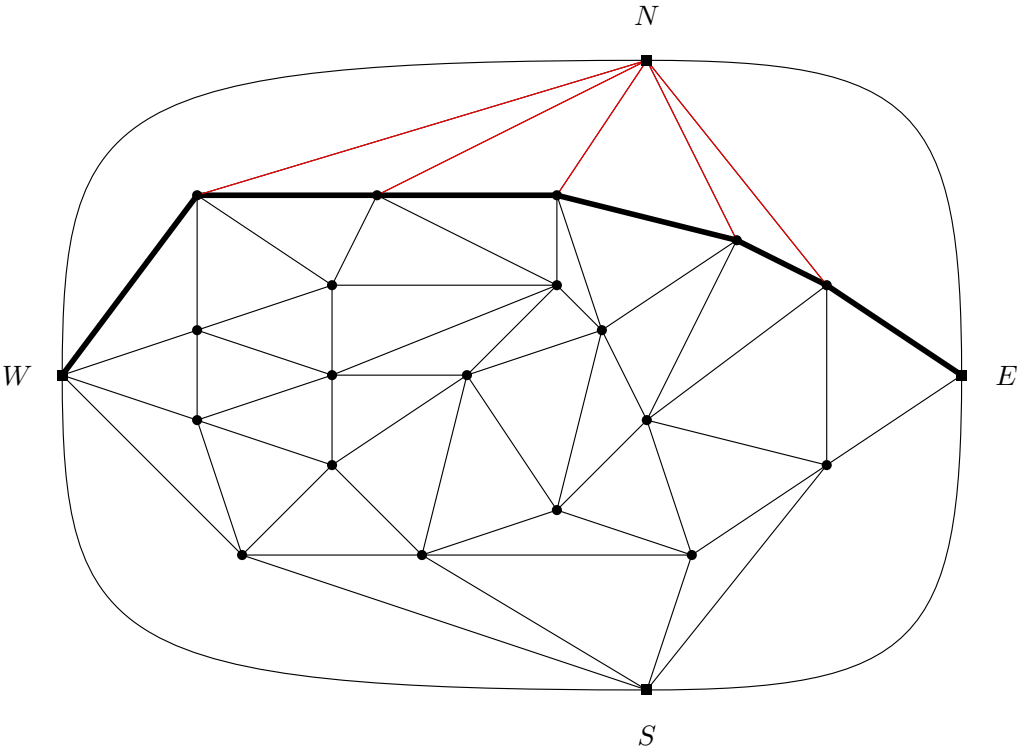
On the four next pages we will give an example execution of our algorithm on a fairly simple graph. We hope this helps the reader to get some intuition of how the algorithm operates. We consider the graph given in Figure 48a this graph has as highest degree vertices several vertices of degree 6. Our algorithm should thus give at least a 5-sided layout, but it actually gives a 2-sided layout in this case.

The application of the algorithm to the graph is straightforward. Figures 48a to 48g are steps in the sweepline algorithm. This graph does not have any blue Z's or topfans so we can skip these steps of the algorithm. Then finally the subdivision of large red faces happen in Figure 48h.

In the captions of the subfigures of Figure 48 more details about each step can be found.

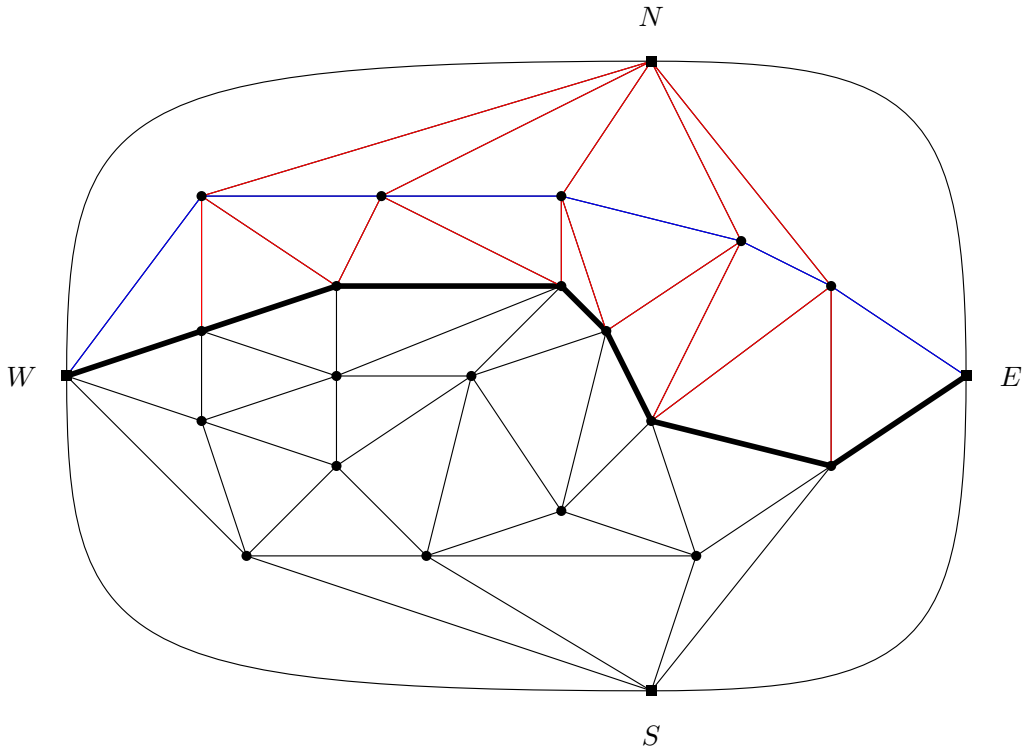


(a) The graph upon which we will execute the algorithm.

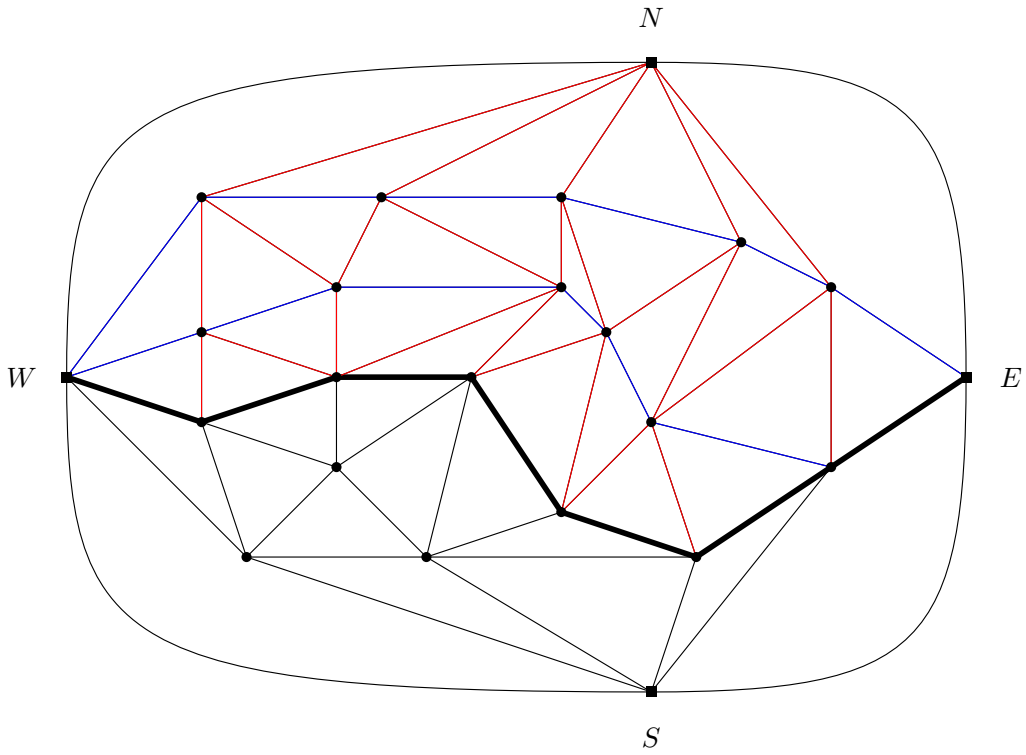


(b) The initial sweepcycle.

Figure 48: The steps of the algorithm.

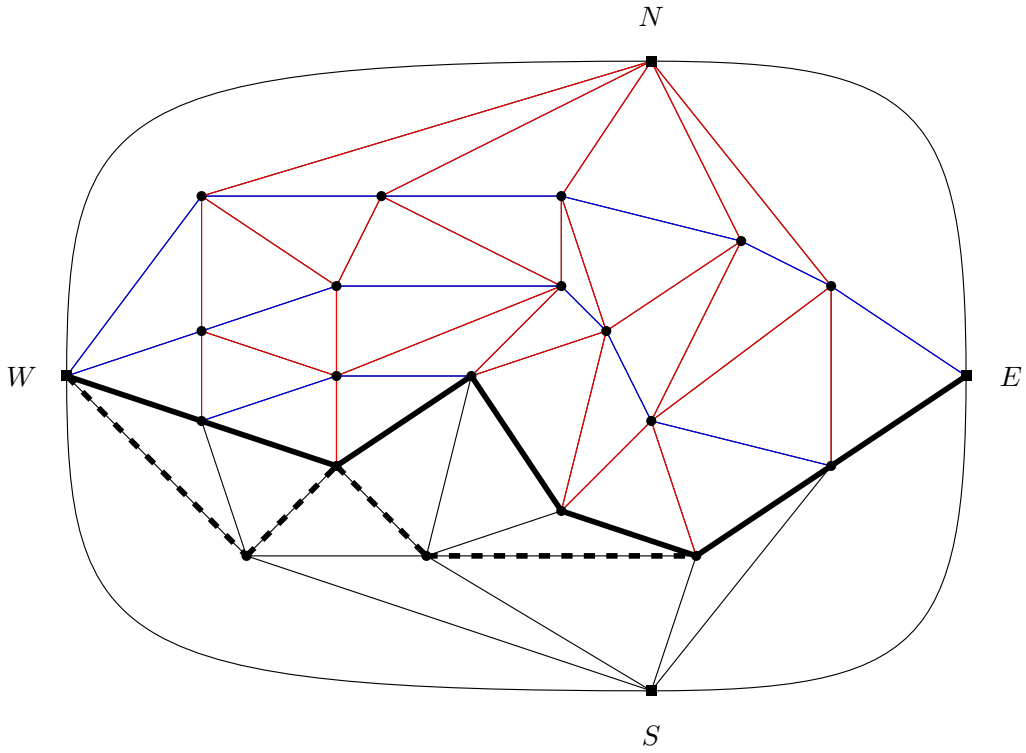


(c) Advancing by one update of the sweepcycle.

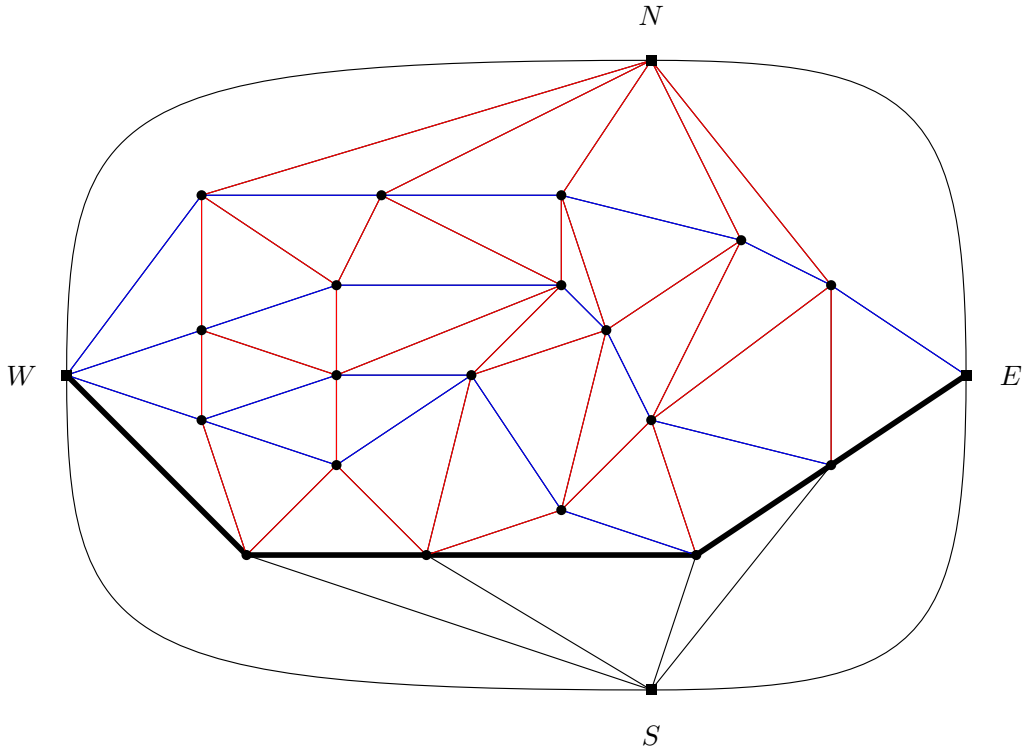


(d) Another update step.

Figure 48: The steps of the algorithm.

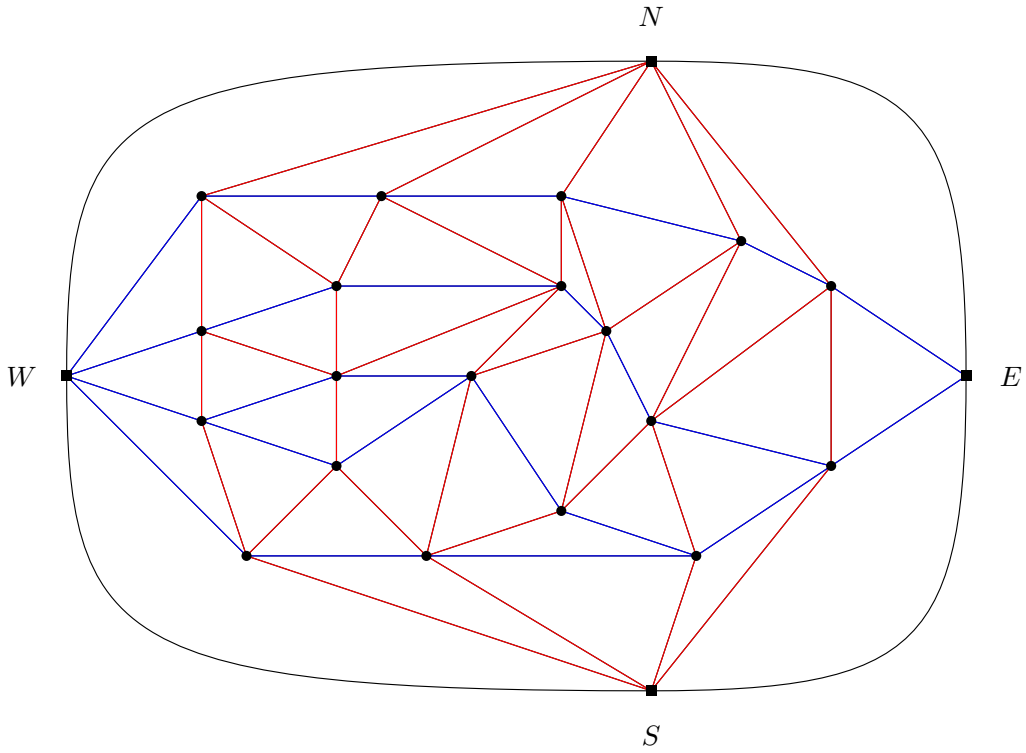


(e) This update the candidate path (dashed) had a chord leading to this smaller update.

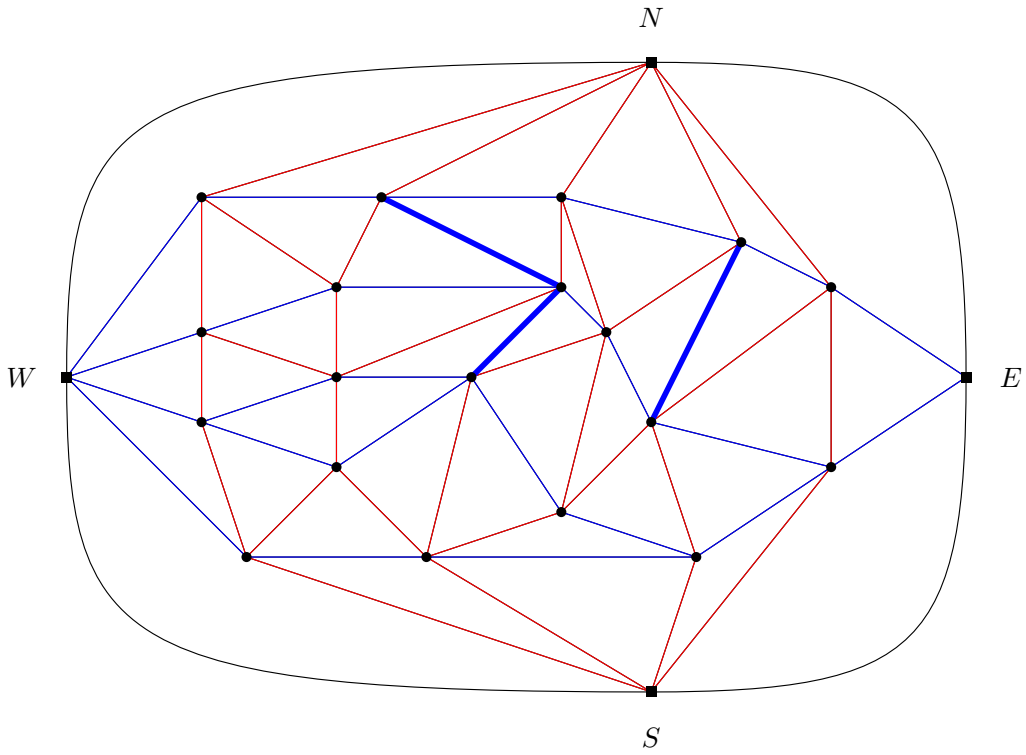


(f) Another update step. This time again without violating chords.

Figure 48: The steps of the algorithm.



(g) Terminating the sweepcycle step of the algorithm.



(h) The flips we make during blue face subdivision.

Figure 48: The steps of the algorithm.