# Branch decompositions of $k$-outerplanar graphs

Olav Hjortås
olavh@ii.uib.no

Cand. scient. thesis


Department of Informatics,
University of Bergen,
Norway

25th July 2005

# Contents

# Acknowledgments

First and foremost, I thank my advisor, Professor Fëdor V. Fomin, for patiently guiding my work. I am also grateful to Professor Jan Arne Telle. He spent a great deal of time with me when I started my graduate studies being a teaching assistant in his course on advanced algorithms. He also had the initial idea of what I have called spanning branch decompositions, which forms the backbone of this thesis.

I wish to thank PhD student Christian Sloper, who has given me much valuable and enjoyable insight into the world of informatics research. I thank Omer Kareem Basit and Gunnar Bakken Sørensen for being there. Finally, I thank my parents for supporting me and believing in me.

# Chapter 1

# Introduction

Graphs represent structures in a very general way. Euler is considered the founder of graph theory, solving the Königsberg bridges problem [Wil96] in 1736. This branch of mathematics was not in much use, however, before the 20th century. A large number of discrete mathematical problems, also from fields as diverse as operational research, chemistry, sociology and genetics, can be represented as problems on graphs. It is therefore of great interest to find ways to solve graph problems quickly. Many problems are NP-hard on graphs in general. Graphs may be very complicated, and several methods have been sought to find structures in graphs so that problems can be solved more easily. Robertson and Seymour invented several ways to decompose graphs [RS91] for proving important properties of graphs, especially Wagner's conjecture. NP-complete problems can often be solved in polynomial time on graphs having a sufficiently simple decomposition. *Tree decompositions* is one of these decompositions. After Robertson and Seymour introduced them, they have proven to be very useful, allowing so-called dynamic programming, running in exponential time with respect to the treewidth of the graph (measuring how closely the graph resembles a tree), but in linear time with respect to the size of the graph itself. Another decomposition is *branch decompositions*, which is the subject of this thesis. Again, the goal is to find some underlying tree structure in the graph. The branchwidth is a measure of how intertwined the graph is when considering the different 'branches' of the tree.

Some graphs can be drawn in the plane without crossings, and we call a graph with such a drawing a plane graph. If we have to 'peel' away the outermost layer $k$ times before we are left with an empty graph, we say that the graph is $k$-outerplanar – it has $k$ layers. A spanning tree of a connected graph G is a graph covering all the vertices of G, but just enough edges necessary to make it connected. Bodlaender [Bod96] has shown how to choose a spanning tree for a $k$-outerplanar graph by some very simple rules, and then modifying this into a tree decomposition of width $3k-1$. This thesis presents a very simple algorithm for instead modifying Bodlaender's spanning tree into a branch decomposition of width $2k+1$.

Branch decompositions, as well as tree decompositions, may be used for dynamic programming. Dynamic programming is a method for reducing the runtime of algorithms

exhibiting the properties of (1) overlapping subproblems and (2) optimal substructure. A problem is said to have overlapping subproblems if the problem can be broken down into subproblems which are reused several times. Optimal substructure means that optimal solutions of subproblems can be used to find the optimal solutions of the overall problem.

In terms of branch and tree decompositions, dynamic programming utilizes the tree of the decomposition. First, the problem is solved locally for the leaves of the tree. Assuming a rooted tree, vertices gradually closer to the root are then considered, and the solution is modified for each step, especially when different 'branches' of a tree meets. The running time of these algorithms increase exponentially as the width of the decomposition increases, so small width is of paramount importance. Tree decompositions have been used for dynamic programming for a long time, while the use of branch decompositions is much newer.

# Chapter 2

# Preliminaries

## 2.1 Elementary definitions

The largest integer no greater than $x$ is denoted $\lfloor x \rfloor$.

**Definition 1.** A function $f : S \to T$ is **injective** if $f(s) = f(s')$ implies $s = s'$, and it is **surjective** (or **onto**) if for every $t \in T$ there exists an $s \in S$ such that $f(s) = t$. If a function is both injective and surjective, it is called a **bijection**.

(Some definitions of graph theory are taken from Wilson [Wil96], which is a standard introductory text on this topic.) A **graph** $G = (V, E)$ is a set of **vertices** $V$ and a set of **edges** $E$, such that each edge is represented by two different vertices, meaning $E \subset V \times V$. Graphs are usually drawn using a dot to represent each vertex and a line between two vertices to represent an edge between them. We will use this convention, and denote such a drawing as an **embedding**. The embedding is not inherent in the graph itself, however, but we sometimes say that 'G is a graph with a given embedding', for instance. We will also use the notation $(G, \Phi)$ meaning the graph $G$ with the embedding $\Phi$. Sometimes, we use the notation $V(G)$ or $E(G)$ to represent the vertex or edge set of $G$, respectively.

Each edge $e = (u, v)$ has two **endpoints**, $u$ and $v$, and we say that $u$ and $v$ are **neighbours** and **adjacent**. The edge $e$ is said to be **incident** to $u$ and $v$. Two edges are **adjacent** if they share a vertex. The **degree** of a vertex is the number of edges incident to it. A **leaf** is a vertex of degree 1. An **internal vertex** is a vertex of degree 2 or more. A **leaf edge** is an edge incident to a leaf. All other edges are called **internal**. The **degree** of a graph $G$, $\deg(G)$, is the maximum degree over all its vertices. The **union** of two graphs $G = (V, E)$ and $G' = (V', E')$ is $G \cup G' = (V \cup V', E \cup E')$. This generalizes to more than two graphs in the obvious way. **Suppressing** all vertices of degree two means that, for any vertex of degree two we delete it and add an edge joining its neighbours and continue this process until no such vertices remain.

We sometimes use rather informal notation for describing graphs with something added or taken away, like in [Die00], a standard book on graph theory. For instance, $G + \{e\}$ is $G$ with the addition of $e$. While less formal, this notation hopefully is more easily understandable than the formal manipulation of vertex and edge sets.

A graph $(V', E')$ is a **subgraph** of a graph $(V, E)$ if $V' \subseteq V$ and $E' \subseteq E$.

Given some graph, a **path** is a finite sequence of edges of the form $(v_0, v_1)$, $(v_1, v_2)$, $\ldots$, $(v_{m-1}, v_m)$, also denoted by $v_0 \to v_1 \to v_2 \to \cdots \to v_m$, in which any two consecutive edges are adjacent, and the vertices $v_0, v_1, \ldots, v_m$ are distinct (except, possibly, $v_0 = v_m$). If we do have $v_0 = v_m$, the path is called a **cycle**. We speak of a **walk from** $v_0$ **to** $v_m$. The number of edges in a path is called its **length**. The **distance** between two vertices is the minimum length of the paths between them. A graph is **connected** if there is a path between each pair of vertices. If a graph is disconnected, we call the different connected parts of it **connected components**, while a connected graph has a single connected component. (A **singleton** – a graph consisting of a single vertex – is considered connected.) A **forest** is a graph without cycles, and a **tree** is a connected forest. A **spanning tree** of a connected graph $G = (V, E)$ is a tree $T = (V, F)$ where $F \subseteq E$. We then speak of the **tree edges** (those in $F$) and the **non-tree edges** (the others). A **maximal spanning forest** of a graph $G$ is the union of spanning trees for each connected component of $G$. For simplicity, we speak of tree edges and non-tree edges for maximal spanning forests also. A **subtree** of a tree $T$ is a tree that is a subgraph of $T$. (Of course, any subgraph of a tree is a tree.) A **minimal subtree** with respect to some property is a tree $T$ such that no **proper** subgraph of $T$ (that is, apart from $T$) satisfies the property.

A **planar graph** is a graph that can be drawn in the plane such that no edges cross, i.e., so that no two edges intersect geometrically except at a vertex to which both are incident. Any such drawing is a **plane drawing**. A **plane graph** is a planar graph with some plane drawing.

Let $e = (x, y)$ be an edge of a graph $G = (V, E)$. By $G/e$ we denote the graph obtained from $G$ by **contracting** the edge $e$ into a new vertex $v_e$, which becomes adjacent to all the former neighbours of $x$ and of $y$. Formally, $G/e$ is a graph $(V', E')$ with vertex set $V' = V \setminus (x, y) \cup v_e$ (where $v_e$ is the 'new' vertex, i.e., $v_e \notin V$) and edge set

$$E' = \Big\{(v, w) \in E \mid \{v, w\} \cap \{x, y\} = \emptyset\Big\} \cup \Big\{(v_e, w) \mid (x, w) \text{ or } (y, w) \text{ is in } E \setminus \{e\}\Big\}.$$

If a graph $H$ can be obtained from a subgraph of $G$ by zero or more edge contractions, then $H$ is a **minor** of $G$.

If $G$ is a planar graph, then any plane drawing of $G$ divides the set of points of the plane not lying on $G$ into regions, called **faces**. There is always exactly one unbounded face, it is called the **infinite** (or **exterior**) face. There are also zero or more **interior** faces. Vertices and edges are **on** a face if they are at its border. Note that while vertices can be on an arbitrarily large number of faces (at least one), an edge is on either one or two faces.

We use the symbol $K_n$ for a graph having $n$ vertices and edges between each pair of vertices. Such a graph is called a **complete graph**. $K_n$ has $n(n-1)/2$ edges. The smallest examples are $K_1$ (a single vertex) and $K_2$ (an edge with its two endpoints).

## 2.2 Decompositions

The next definition is based on the one in Alber's PhD thesis [Alb02].

**Definition 2.** Let $(G = (V, E), \Phi)$ be a plane graph.

1. The **layer decomposition** of $(G, \Phi)$ is a disjoint partition of $V$ into sets $L_1, \ldots, L_r$, which are recursively defined as follows:
   - $L_1$ is the set of vertices on the infinite face of $G$.
   - $L_i$ is the set of vertices on the infinite face of $G - L_1 - \cdots - L_{i-1}$.
2. The set $L_i$ is called the $i$**th layer** of $(G, \Phi)$.
3. The number $k$ of different non-empty layers is called the **outerplanarity of** $(G, \Phi)$, denoted by $\mathrm{out}(G, \Phi)$. The embedding $\Phi$ is $k$**-outerplanar**.
4. The smallest outerplanarity of $(G, \Phi)$ over all planar embeddings $\Phi$ is called the **outerplanarity of** $G$, denoted by $\mathrm{out}(G)$. We also say that $G$ is $k$**-outerplanar**.

Bodlaender's definition of $k$-outerplanarity in [Bod96] is a bit vague:

> *An embedding of a graph $G = (V, E)$ is 1-outerplanar, if it is planar, and all vertices lie on the exterior face. For $k \geq 2$, an embedding of a graph $G = (V, E)$ is $k$-outerplanar, if it is planar, and when all vertices on the outer face are deleted, then a $(k-1)$-outerplanar embedding of the resulting graph is obtained. A graph is $k$-outerplanar, if it has a $k$-outerplanar embedding.*

Certainly many graphs are both $k$-outerplanar and $(k+1)$-outerplanar by this definition. It can even be interpreted that a $k$-outerplanar graph is also $(k+n)$-outerplanar for any positive $n$; that is, $k$-outerplanarity indicates only an upper limit on the number of layers. I have chosen to use a definition which assigns a unique outerplanarity to each graph.

**Peeling** a plane graph is removing all vertices on its infinite face. A plane graph is $k$-outerplanar if it has to be peeled $k$ times before the **empty graph** $(\emptyset, \emptyset)$ is attained. It is also easy to see that peeling a non-empty graph reduces its outerplanarity by one. We have the following trivial result:

**Lemma 3.** *Given a plane graph $G$ with $k$ layers, every layer except possibly layer $k$ contains a cycle.*

*Proof.* We will show that this is true for an arbitrary layer $\ell < k$. Each non-innermost layer prevents vertices inside this layer[1] from being on the same layer. Since this barrier is 360 degrees, it must contain a cycle. □

While layer decompositions are straightforward, we now turn towards two more advanced decompositions.

[1] That is, the vertices in $G - L_1 - \cdots - L_{\ell-1}$.

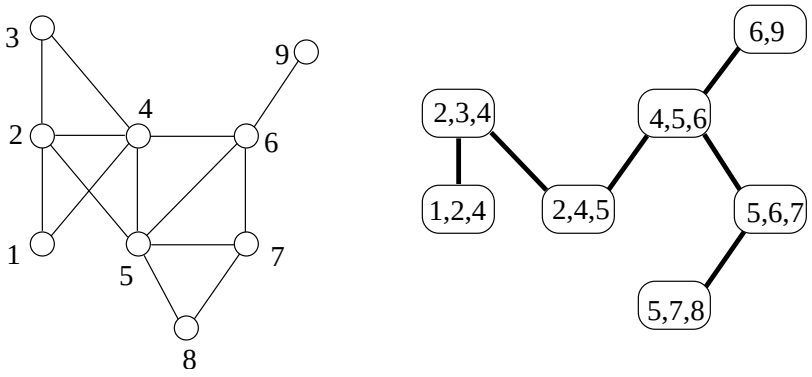


Figure 2.1: A graph $G$ (left) and a tree decomposition for $G$ (right). The boxes represent the bags. The boxes and the edges between them make up the tree. The three conditions of Definition 4 are met: *1.* The bags contain all eight vertices of $G$. *2.* The endpoints of each edge (say, vertices 2 and 4 of $(2,4)$), are in at least one common bag (say, $\{2,4,5\}$). *3.* For each vertex in $G$, the bags containing this vertex are connected (the bags containing, say, vertex 5, are connected).

**Definition 4.** A **tree decomposition** $\mathcal{X}$ of a graph $G = (V, E)$ is a pair

$$(\{X_i \mid i \in I\}, T = (I, F)),$$

where each $X_i$ is a subset of $V$ called a **bag**, and $T$ is a tree, such that:

1. The union of all the bags is $V$. That is, $\cup_{i \in I} X_i = V$.
2. For each edge $(u, v) \in E$, there is a node $X_i$ of $T$ such that both $u$ and $v$ are in $X_i$.
3. For each vertex $v \in V$, $\{i \in I \mid v \in X_i\}$ induces a subtree. That is, the bags that contain $v$ are connected.

The **width** of $\mathcal{X}$ is defined to be the size of its largest bag minus one. The **treewidth** of a graph $G$, $\mathrm{tw}(G)$, is the minimum width over all tree decompositions of $G$. A tree decomposition of $G$ is **optimal** if its width is equal to the treewidth of $G$.

A **ternary tree** is a tree where each vertex has degree 1 or 3. (Do not confuse this with a 3-ary tree.)

**Definition 5.** A **branch decomposition** of a graph $G = (V, E)$ is a pair $(T, \sigma)$, where $T$ is a ternary tree and $\sigma$ is a bijection from $E$ to the set of leaves of $T$. This means that each leaf of $T$ **represents** an edge of $E$. We call $T$ the **branch tree**. Notice that the removal of an edge $e$ of $T$ partitions the leaves, and we say that it colours the corresponding edges of $G$ with two colours, $\alpha$ and $\beta$. The **middle set** $\omega(e)$ is the set of vertices of $G$ which have incident edges of both colours. We say that **collisions** occur at these vertices. The size of $\omega(e)$ is called the **order** of $e$. The **width** of a branch decomposition is the maximum order over all edges $e$ of $T$. The **branchwidth** of $G$, $\mathrm{bw}(G)$, is the minimum width over all

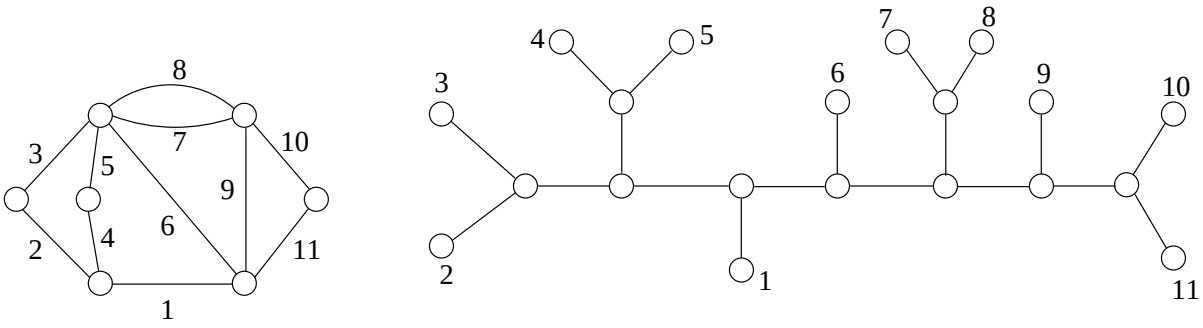


Figure 2.2: Example of a graph (left) and its branch decomposition (right). The bijection is indicated by the labels.

branch decompositions of G. (If $G$ has one or zero edges, it has no branch decomposition, and its branchwidth is zero.) A branch decomposition of $G$ is **optimal** if its width is equal to the branchwidth of $G$.

Note that branch decompositions are usually not defined using the concept of colour, but merely speak of partitionings or paths instead. Hopefully, the reader will agree that the concept of colour makes the definition easy to follow, especially when accompanied by a figure using colours (or black and grey, to be exact). Also, note that this definition explains how to deal with graphs with one or zero edges. So did Robertson and Seymour's original definition, but all other definitions known to me, skip this detail. The following is an example of a traditional definition of branchwidth (from [Bod96]):

> *A* branch decomposition *of a graph $G = (V, E)$ is a pair $(T = (I, F), \sigma)$, where $T$ is a tree with every node in $T$ of degree one or three, and $\sigma$ is a bijection from $E$ to the set of leaves in $T$.*
>
> *The* order *of an edge $f \in F$ is the number of vertices $v \in V$, for which there exist adjacent edges $(v, w), (v, x) \in E$, such that the path in $T$ from $\sigma(v, w)$ to $\sigma(v, x)$ uses $f$.*
>
> *The* width *of branch decomposition $(T = (I, F), \sigma)$ is the maximum order over all edges $f \in F$. The* branchwidth *of $G$ is the minimum width over all branch decompositions of $G$.*

See Figures 2.2 and 2.3 for an example of a graph and its branch decomposition. (This figure is taken from [RS91] where branch decompositions were originally defined.) The bijection is indicated by the labels. Indeed, we could define a branch decomposition without the bijection, saying that each edge of the graph should appear as a leaf in the tree.

When making a branch decomposition for a graph by hand, one should remember to make the structure of the tree resemble the structure of the graph. We attempt to place leaves representing nearby edges close to each other. If we are not careful about this, so that nearby edges are represented by distant leaves, these edges can easily be coloured differently when removing an edge, and we get many collisions.

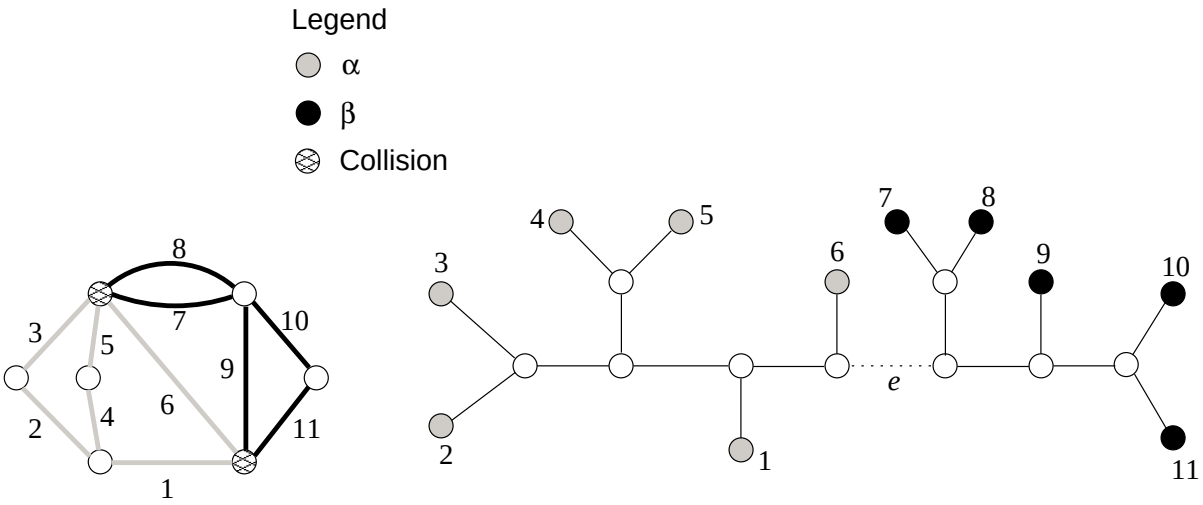


Figure 2.3: On the right, we remove an edge $e$ of the branch tree. This colours the leaves with colours $\alpha$ and $\beta$, shown here as grey and black. Using the bijection, the edges of the graph on the left are coloured correspondingly. Collisions occur where different colours meet. Since there are two collisions here, the order of $e$ is two. There are no edges of higher order than two, so the width of this decomposition is two.

Note that $G$ is disconnected. Disconnected graphs are in general not interesting, only being a nuisance when writing proofs. Branch decompositions by definition require a branch *tree*, not a forest, but each component can be treated separately with respect to collisions. So, assume that we have a graph consisting of several components. Each of these can be considered as seperate graphs, and we can build a branch decomposition for each. It is easy to join the branch trees together, and the bijections do not change. No more collisions appear, so the width of this new decomposition of the whole graph is the maximum of the width of the individual decompositions.

A carving decomposition is similar to a branch decomposition, but it decomposes the vertex set instead of the edge set of a graph:

**Definition 6.** A **carving decomposition** $C$ of $G = (V, E)$ is a ternary tree in which each vertex of $G$ appears as a leaf exactly once. For each ordered pair $(p, q)$ of adjacent nodes in $C$, we denote by $C_{p,q}$ the subtree of $C$ induced by the nodes reachable from $p$, including $p$, without using the edge $(p, q)$. For each subtree $C'$ of $C$, let $L(C')$ denote the set of vertices of $G$ contained, as leaves, in $C'$. We say that the edge $(p, q)$ induces the separation $\{L(C_{p,q}), L(C_{q,p})\}$ of V. We also say that this separtion is **in** $C$. The **cut** associated with a separation $\{S, V \setminus S\}$ of $V$ is the set of edges having one end in $S$ and the other end in $V \setminus S$. The **width** of a separation of $V$ is the cardinality of its associated cut. The **width** of a carving decomposition is the maximum width over all separations of the decomposition. The **carving width** of $G$ is the minimum width over all the possible carving decompositions of $G$.

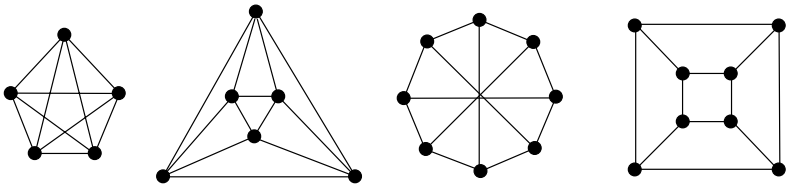

Figure 2.4: Obstruction set for graphs with branchwidth at most 3.

## 2.3 Minors and tangles

A **class** of graphs is a (usually large, often infinite) set of graphs. If every minor of a member of a certain class is itself in the class, this class is **minor closed**. The **obstruction set** of such a class $\mathcal{F}$ is the set of graphs $\mathcal{B}$ which are not members of $\mathcal{F}$, while every minor of a member of $\mathcal{B}$ *is* a member of $\mathcal{F}$. So no member of $\mathcal{F}$ can have the same structure as any member of $\mathcal{B}$.

A graph has branchwidth

**0** if and only if each component of has at most one edge,

**at most 1** if and only if each component has at most one vertex with two or more neighbours,

**at most 2** if and only if it has no $K_4$ minor,

**at most 3** if and only if it has no $K_5$, $M_6$, $M_8$ or $Q_3$ minor (this obstruction set is shown in Figure 2.4.

A **separation** of a graph $G$ is a pair $(G_1, G_2)$ of subgraphs whose union is $G$ while they have no edges in common. The **order** of this separation is the number of vertices they have in common. Let $G$ be a graph and let $\theta$ be a positive integer. A **tangle** in $G$ of **order** $\theta$ is a set $\mathcal{T}$ of separations of $G$, each of order less than $\theta$, such that

1. for every separation $(A, B)$ of $G$ of order less than $\theta$, one of $(A, B)$, $(B, A)$ is in $\mathcal{T}$
2. if $(A_1, B_1), (A_2, B_2), (A_3, B_3) \in \mathcal{T}$, then $A_1 \cup A_2 \cup A_3 \neq G$
3. if $(A, B) \in \mathcal{T}$ then $V(A) \neq V(G)$.

We refer to these as the first, second and third (tangle) axioms. The **tangle number** of $G$, denoted $\theta(G)$, is the maximum order of tangles in $G$ (or 0, if there are no tangles).

There is a relatively simple lemma stating necessary and sufficient criteria for a set of separations to be a tangle:

**Lemma 7.** *Let $\mathcal{T}$ be a set of separations of a graph $G$, each of order less than $\theta$, satisfying the first and second tangle axioms. Then $\mathcal{T}$ is a tangle if an only if $(K_e, G \setminus e) \in \mathcal{T}$ for every edge $e$, where $K_e$ is the graph formed by $e$ and its ends.*

The following theorem states the important result that except for very small graphs, branch width equals the tangle number.

**Theorem 8.** *For any graph having one or more edges,* $\max(\mathrm{bw}(G), 2) = \theta(G)$.

The following relation holds between tree width and branch width:

**Lemma 9.** *For any graph G, we have* $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq \lfloor (3/2)\mathrm{bw}(G) \rfloor$.

All these results are due to Robertson and Seymour [RS91].

## 2.4 Asymptotic notation

There are a few references to running time for algorithms in this thesis. Running time is measured relative to the size of the input. We are usually concerned with the **worst case**: the maximum running time possible for a given input size. Generally, running time depends greatly on the computer, operating system and implementation and compilation of algorithms. Telling the running time exactly would require careful measurement and analysis of the compiled program, and is not called for in most cases, certainly not in theoretical computer science.

It has for a long time been known that so-called asymptotic notation is the most meaningful way to judge running time roughly. The cornerstone of this is $O$-notation:

**Definition 10.** For a given function $g(n)$, we denote by $O(g(n))$ the set of functions $\{f(n) :$ there exist positive constants $c$ and $n_0$ such that $0 \leq f(n) \leq cg(n)$ for all $n \geq n_0\}$.

To indicate that a function $f(n)$ is a member of $O(g(n))$, we write '$f(n) = O(g(n))$' or '$f(n)$ is $O(g(n))$.' We are concerned with how long the running time may be for input of size $n$. We measure running time in the number of steps made by a computer on this input. If an algorithm runs in time proportional to $n$, we say that its running time is $O(n)$, and that it runs in **linear time**. This way, it is easy to distinguish its running time from, for instance, $O(n^2)$ (**quadratic time**) and $O(n \log n)$, commonly occuring for sorting algorithms. If there exists a $k$ such that an algorithm runs in time $O(n^k)$, we say that in runs in **polynomial time**.

The term *NP-complete* indicates that a problem has been proven to belong to a class of difficult problems, known as NP. The term *NP-hard* indicates that a problem is at least this difficult. This is part of complexity theory, and the reader is referred to [Sip97] for an introduction.

# Chapter 3

# Branch decompositions of $k$-outerplanar graphs

## 3.1 Face-vertex walks

Outerplanarity, as given in Definition 2, is a relatively well known invariant of graphs. Tamaki introduced a new invariant, face-vertex height, in his paper on branch decompositions of planar graphs [Tam03]. Below, we show that the face-vertex height is roughly twice the outerplanarity of a graph.

**Definition 11.** Let $G$ be a graph with a planar embedding. A **face-vertex walk** of $G$ is a sequence $x_0 \to x_1 \to \cdots \to x_k$ of faces and vertices of $G$ such that, for $1 \leq i \leq k$,

1. if $x_{i-1}$ is a face then $x_i$ is a vertex and vice versa, and
2. $x_{i-1}$ and $x_i$ are incident.

We say that face-vertex walk $x_0 \to x_1 \to \cdots \to x_k$ is of length $k$ and is from $x_0$ to $x_k$. The **face-vertex height** of a vertex or face $x$ of $G$ is the length of the shortest face-vertex walk from the outer face of $G$ to $x$. The **face-vertex height** of $G$, denoted $\alpha_G$, is the maximum face-vertex height over all vertices and faces in $G$.

Tamaki gives the following result:

**Theorem 12.** *Given a biconnected planar graph $G$ with a planar drawing, a branch decomposition of $G$ with width at most $\alpha_G$ can be constructed in linear time.*

Carving decompositions and medial graphs are used to prove this. The invariant $\alpha_G$ can easily be related to the outerplanarity of $G$:

**Lemma 13.** *Let $G$ be a planar graph. If $\alpha_G \geq 2$, then $G$ is $\lfloor \alpha_G/2 \rfloor$-outerplanar.*

*Proof.* We prove this by induction on $\alpha_G$, which is defined to be $\max_x \alpha_x$. If $\alpha_G = 2$, then such a maximum $\alpha_x$ corresponds to a walk $x_0 \to x_1 \to x$, where $x_0$ is the infinite face.

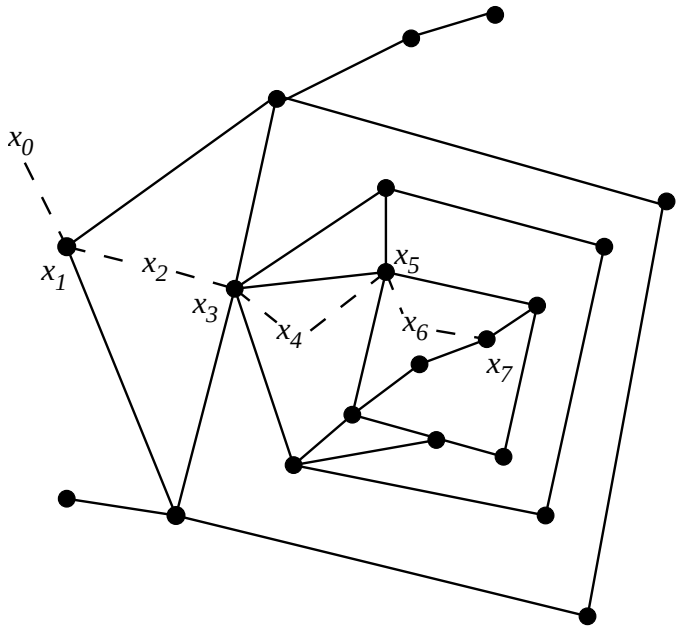


Figure 3.1: One of the shortest face-vertex walks from the outer face ($x_0$) to $x_7$ is indicated by a dashed line. In this particular graph $G$, a vertex has the maximum height, resulting in an odd $\alpha_G$. If a face has the maximum height in a graph, $\alpha_G$ will be even.

Clearly, $G$ is $\lfloor 2/2 \rfloor$-outerplanar, thus the implication of the lemma is true for this basis step.

We let the induction hypothesis be that the implication is correct for $2 < \alpha_G \leq n$. We want to show that the implication is correct for $\alpha_G = n+1$. Let $G$ be any graph such that $\alpha_G = n+1$. We assume that a corresponding embedding is given. Let $G'$ be the resulting graph when removing all vertices on the outer face of $G$.

Let $x$ be a vertex or face of $G$ such that $\alpha_x$ is maximum. Then, $\alpha_x$ corresponds to a walk $x_0 \to x_1 \to x_2 \to \cdots \to x_n \to x$ of length $n+1$. In $G'$, $x_2$ is the infinite face. Therefore, $\alpha_x$ there corresponds to $x_2 \to x_3 \to \cdots \to x_n \to x$, which is of length $n-1$. By the induction hypothesis, the outerplanarity of $G'$ is $\lfloor (n-1)/2 \rfloor$. Because $G$ only contains one more layer, the outerplanarity of $G$ is $\lfloor (n-1)/2 \rfloor + 1 = \lfloor (n-1)/2+1 \rfloor = \lfloor (n-1+2)/2 \rfloor = \lfloor (n+1)/2 \rfloor$, which is what we wanted to show. So the implication is proven by induction. □

## 3.2 The spanning tree approach

We now proceed to the core of this thesis. A new, simple algorithm is given, and we prove that it builds a branch decomposition of low width.

**Definition 14.** Given a graph $G$ and a spanning forest $T$ for $G$, the **load** of a tree edge $e$, denoted $\text{load}(e)$ or $\text{load}_{G,T}(e)$, is the set of non-tree edges $(x, y)$ such that the unique path between $x$ and $y$ in $T$ uses $e$. (If the cardinality of $\text{load}(e)$ is $k$, we may say that the load

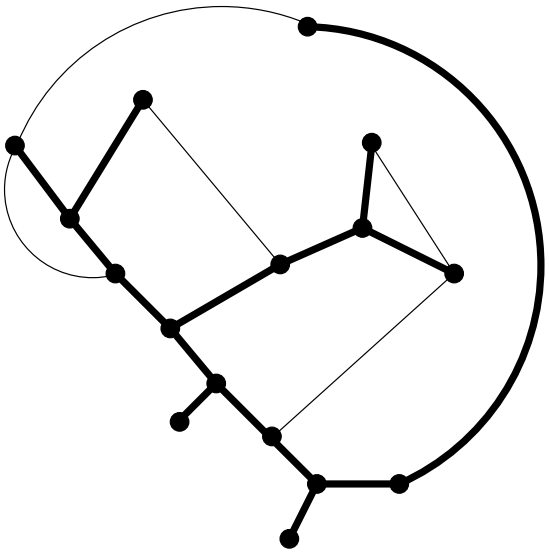

Figure 3.2: This graph $G$ consists of a spanning tree $T$ (shown as thick edges) and some other edges (thin edges).

of $e$ is $k$.) Further, $\text{load}(G,T)$ is the maximum cardinality of $\text{load}(e)$ over all tree edges $e$.

Note that the path in this definition plus the edge $(x,y)$ constitutes a cycle, which we call the **fundamental cycle** of $e$.

**Algorithm 15.** Given a connected graph $G=(V,E)$ and a spanning forest $T=(V,F)$; the branch tree $T_b$ and a function $\sigma$ is defined as follows:

1. Let $T_b := T$.

2. For each edge $e=(u,v) \in F$ where both $u$ and $v$ are internal vertices in $G$, add a new vertex $w$ to $T_b$, and replace $e$ by the edges $e_1=(u,w)$ and $e_2=(w,v)$. Add another new vertex $x$ and a new edge $(w,x)$. Define $\sigma(e) := x$.

3. For each edge $e=(u,v) \in F$ where $u$ is a leaf in $G$, define $\sigma(e) := u$.

4. For each edge $e=(u,v) \in E-F$, add a new vertex $x$ and a new edge $(u,x)$ to $T_b$. Define $\sigma(e) := x$.

5. Delete the leaves of $T_b$ which do not represent any edge, i.e. each leaf $v$ for which there is no edge $e \in E$ such that $\sigma(e)=v$.

6. Suppress all vertices in $T_b$ of degree two.

The pair $(T_b,\sigma)$ is a a **spanning branch decomposition for $G$ given $T$**.

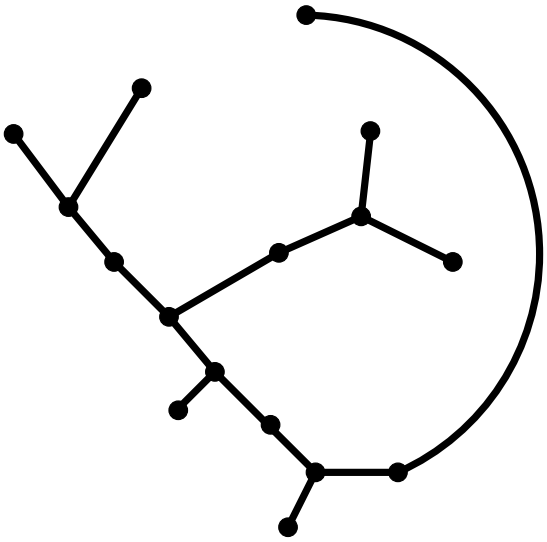

Figure 3.3: **Step 1.** The algorithm is given $G$ and $T$ as input and builds a branch tree $T_b$with some bijection. Initially, $T_b$ is defined as $T$.

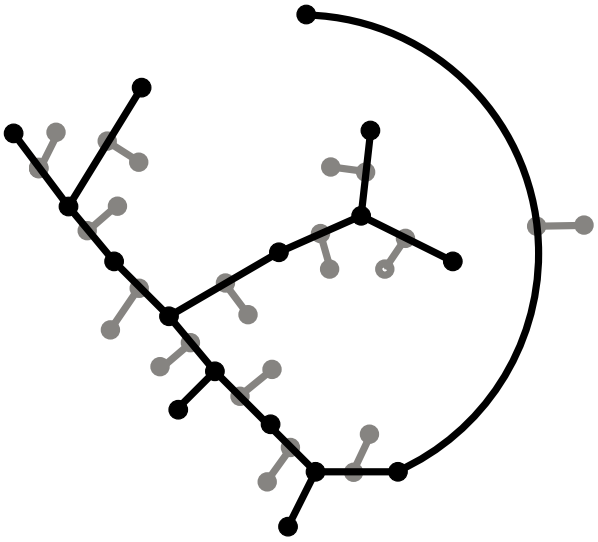

Figure 3.4: **Step 2.** A new edge is added on each edge which is internal in $G$ (note: not in $T_b$). Each new vertex represents the edge of $G$ it was added to.

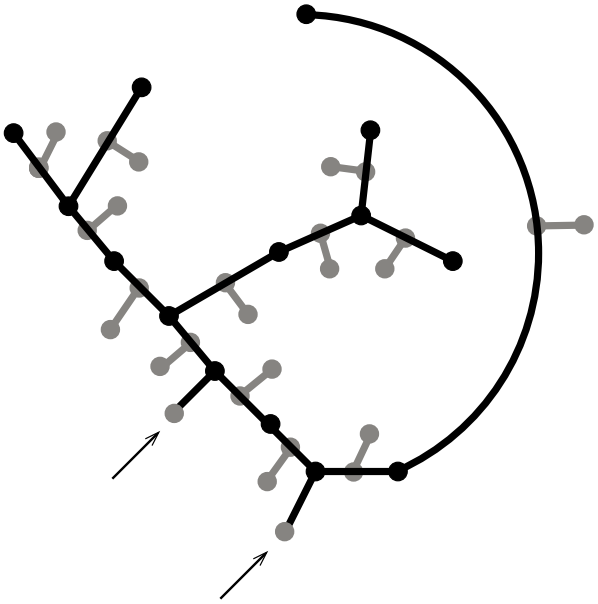

Figure 3.5: **Step 3.** Each leaf edge in $G$ is represented by its adjacent leaf in $T_b$. Here, there were two leaf edges. Their adjacent leaves (see arrows) are coloured gray to indicate that they now represent edges.

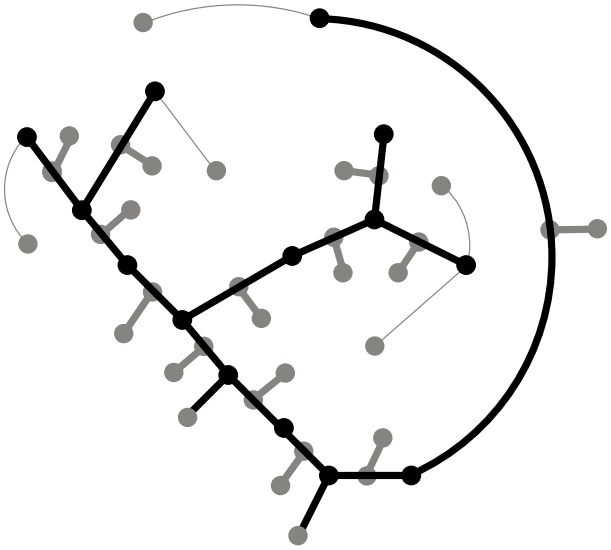

Figure 3.6: **Step 4.** Each non-tree edge $e$ is considered. First, a new vertex is added to represent $e$. Then, it is made a leaf by adding an edge between it and one of the two endpoints of $e$.

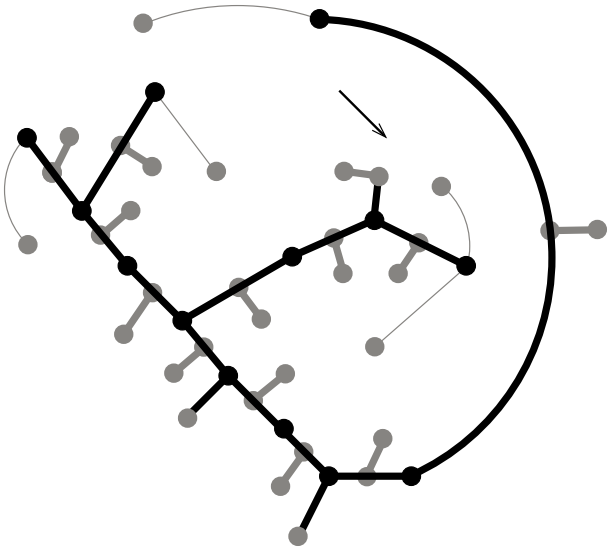

Figure 3.7: **Step 5.** We need to make sure that $T_b$ is a genuine branch tree. We remove leaves which do not represent edges. Here, we had only one such leaf (see arrow), and it was (necessarily) coloured black.

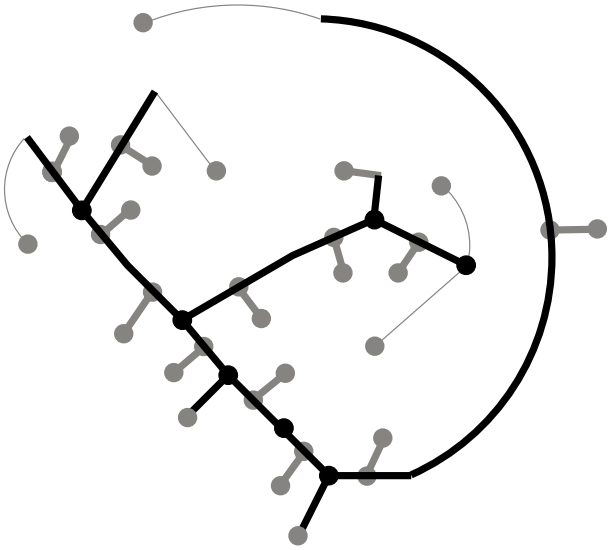

Figure 3.8: **Step 6.** We suppress all vertices of degree 2.

A graph $G$ and its spanning tree $T$ is given in Figure 3.2. The next pages show the algorithm running on this pair, building the branch tree with a bijection. Gray vertices and edges are those added to $T_b$. In step 3, some vertices in $T$ are made representatives; these vertices are also gray. Note that the only 'non-representing' leaves in step 5 are some of the leaves of $T$.

Will this algorithm run on any given graph? It comes up with a so-called spanning branch decomposition, but is this really a branch decomposition? The answer to both questions is 'yes', and we prove this now.

**Lemma 16.** *Given a connected graph $G = (V, E)$ of degree at most 3 and a spanning tree $T$ for $G$, there exists a spanning branch decomposition $(T_b, \sigma)$ for $G$ given $T$, and it is a branch decomposition for $G$.*

*Proof.* First, we prove that such a pair $(T_b, \sigma)$ as given in Algorithm 15 must exist: Steps 1–4 are obviously feasible. (Note that in step 4, there is a choice of which edge to add. This means that branch trees given by some fixed $G$ and $T$ need not be isomorphic.) Step 5 is also feasible, observing that there must be a finite number of 'ghost' leaves, even though more may appear during the elimination process. Finiteness makes step 6 work also. So a pair $(T_b, \sigma)$ must exist.

Now, we need to prove that $(T_b, \sigma)$ is indeed a branch decomposition for G, meaning (1) that $T_b$ is a ternary tree, and (2) that $\sigma$ is a bijection from $E$ to the set of leaves of $T_b$.

1. $T_b$ is based on a spanning tree, and none of the steps makes $T_b$ disconnected or introduces cycles, so it remains a tree.

   We have $\deg(G) \leq 3$, so after step 1, $\deg(T_b) \leq 3$ holds. It is clear that steps 2, 3, 5 and 6 do not violate this inequality; nor does step 4, since for each vertex $u$, each edge added to $u$ in $T_b$ corresponds to a unique non-tree edge incident to $u$ in $G$. Thus, $\deg(T_b) \leq 3$. Since there are no degree 2 vertices left after step 6, and $T_b$ remains connected, all its vertices must be of degree 1 or 3.

2. For each edge $e \in E$, $\sigma(e)$ is defined to be some leaf of $T_b$. This occurs in steps 2–4. *Step 5 ensures that $\sigma$ is surjective (onto), and step 6 does not affect $\sigma$.* Each edge considered in steps 2 and 4 is assigned a brand new leaf. Each edge $(u, v)$ considered in step 3 is assigned the leaf $u$ (in $T_b$), and no other edges are assigned this leaf. *This makes $\sigma$ injective and, again, step 6 does not affect it.* So $\sigma$ is a bijection between the given sets.

□

For a later proof, we need to show that the algorithm does not drastically alter the structure of the tree.

**Lemma 17.** *Given a spanning branch decomposition $(T_b, \sigma)$ for some $G$ given some $T$, and three edges $f$, $g$ and $h$ of $T$, the following statement holds:*

*If, in $T$, $g$ is on the path between $f$ and $h$, then the following holds in $T_b$: the neighbour of $\sigma(g)$ is on the path between the neighbour of $\sigma(f)$ and the neighbour of $\sigma(h)$.*

*Proof.* Note that, for any edge $e$, $\sigma(e)$ has exactly one neighbour. This is true for all branch decompositions, because $\sigma(e)$ is defined to be a leaf.

We will consider if the statement of the lemma is true for the resulting $T_b$ after different steps of Algorithm 15. It is clearly true after steps 2 and 3. Step 4 does not affect it because this deals with edges not in $T$. The statement remains true after step 5, since that step only removes leaves that represent no edge. It also remains true after step 6. □

We are ready to state the core lemma, the proof of which is the most difficult. It links the *order* of an edge (in a branch decomposition) to the *load* of the corresponding edge of $G$. This eventually enables us to make a statement about branch width vs. load.

**Lemma 18.** *Let $(T_b, \sigma)$ be a spanning branch decomposition of some connected $G$ given some $T$, and let $e$ be an edge in $T$. We have $|\omega(e_\ell)| \leq |\text{load}(e)| + 1$ for $\ell = 1, 2$.*

*Proof.* Recall that removing $e_1$ from $T_b$ colours the edges of $G$ with two colours, $\alpha$ and $\beta$. We will first prove that all the tree edges on one side of $e$ are coloured $\alpha$ and all the tree edges on the other side of $e$ are coloured $\beta$: Let $f$ and $g$ be edges of $T$ such that $g$ is on the path between $f$ and $e$ in $T$. (If two such edges do not exist, the claim is obviously true.) Lemma 17 implies that $\sigma(g)$ is in the same partition of the tree as $\sigma(f)$, so $f$ and $g$ must have the same colour. This proves the first claim.

So, for a particular edge $e$, we may visualize $G$ as follows: $e$ is in the middle, and on its left is a subgraph $A$ of tree edges coloured $\alpha$ and on its right subgraph $B$ of tree edges coloured $\beta$. There are also some edges $\mathcal{C}_e$ of either colour between $A$ and $B$. For each such edge $(x, y)$, consider the path from $x$ to $y$ in $T$. It must contain $e$, because $e$ is the only tree edge between $A$ and $B$. This means that $\mathcal{C}_e = \text{load}(e) \cup \{e\}$, since $\text{load}(e)$ does not contain any other edges.

Now, $\omega(e_1)$ is defined to be the vertices which have incident edges of both colours. According to the visualization above, these must be incident to $\text{load}(e) \cup \{e\}$. Each edge in $\mathcal{C}_e = \text{load}(e) \cup \{e\}$ is of a particular colour, so it can at most make a contribution of one new vertex to $\omega(e_1)$. So $|\omega(e_1)| \leq |\text{load}(e) \cup \{e\}| \leq |\text{load}(e)| + 1$. This bound is also given for $|\omega(e_2)|$ by the same argument. □

Now that the order of individual edges has been settled, it is easy to conclude on the width of the whole decomposition.

**Lemma 19.** *Let $(T_b, \sigma)$ be a spanning branch decomposition for some $G$ given some $T$. Then the width of $(T_b, \sigma)$ is at most $\max\{2, \text{load}(T, G) + 1\}$.*

*Proof.* We will consider the order of the edges defined in the different steps in Algorithm 15:

1. All the edges of $E$ are considered in steps 2–4 exactly once.

2. Lemma 18 implies that $|\omega(e_\ell)| \leq \text{load}(T, G) + 1$ for $\ell = 1, 2$.

   This step also adds an edge $(w, x)$ to the new vertex $x$. This is obviously a leaf edge. In general, the removal of a leaf edge of a branch tree gives the edge corresponding to the leaf a colour of its own, so there are at most two collisions because an edge has two endpoints. So the order is at most two. Consequently, $|\omega((w, x))| \leq 2$.

3. The edge $e$ is also a leaf edge, so $|\omega(e)| \leq 2$.

4. The edge $(u, x)$ is also a leaf edge, so $|\omega((u, x))| \leq 2$.

Steps 5 and 6 do not increase the order of any edge. The result follows. □

The spanning forest is built by the following simple procedure, from the inside out. First, a spanning forest is found for the innermost layer. This forest is then extended one layer at a time, towards the infinite face. All edges between the different layers are included in the forest.

We consider graphs that lack cycles everywhere except on the infinite face.

**Lemma 20.** *Let $T$ be a spanning forest of some plane graph $K$ such that all non-tree edges are on the infinite face. Then,* $\text{load}(K, T) \leq 2$.

*Proof.* It suffices to show that the load of an arbitrary tree edge $e$ is no more than two.

First, let us prove that each interior face $f$ of $K$ contains exactly one non-tree edge. Assume the opposite, meaning that $f$ contains the non-tree edges $(u, w)$ and $(x, y)$, both on the infinite face, where $w$ and $x$ may be the same vertex. (Zero non-tree edges on the face would imply that $T$ had a cycle.) Removing these edges would disconnect the graph. The spanning forest $T$ must therefore contain at least one of them, but we assumed that they were both non-tree edges. Contradiction! So each internal face $f$ of $K$ contains exactly one non-tree edge, and therefore the load of $e$ contains at most one edge incident to each face. In other words, each face contains at most one member of $\text{load}(e)$. We know that $e$ is incident to at most two internal faces. Consequently, the internal faces incident to $e$ contribute at most two members to its load.

Next, we will prove that only incident faces can contribute to the load. Assume the opposite: $(x, y) \in \text{load}(e)$, where $(x, y)$ is not on either face incident to $e$. In other words, there is no face to which both $(x, y)$ and $e$ are incident. This means that there is a border (possibly several) between these two edges. This border consists of one or more edges, none of which can be tree edges, or else the forest would have a cycle including this border and $e$. So the border consists on non-tree edges only. This border is inside the fundamental cycle of $e$, so it is not on the infinite face. But all non-tree edges are on the infinite face. Contradiction! So only incident faces can contribute to the load, and we learnt in the last paragraph that they contribute at most two members. Thus, the load of $e$ is at most 2. □

(This proof is based on a lecture note by Biedl [Bie04, Claim 1].)

We now need some results given in [Bod96]. There, Bodlaender uses the term "edge remember number" ($er$) of a graph. This is the same as the *load* defined above, and we will

use this term. Lemmas 21 and 22 correspond to Lemma 79 and 82, respectively, in [Bod96], while Lemma 23 corresponds to Lemmas 80–81. The proofs are based on those in [Bod96], but considerably more detail is given here. The exception is Lemma 22, whose proof is based on [Bie04].

Adding another layer to a graph increases its load by at most 2. In other words:

**Lemma 21.** *Let $G = (V, E)$ be a plane graph. Let $H = (V, E')$ be the graph obtained from $G$ by removing all edges on the infinite face. Let $T' = (V, F')$ be a maximal spanning forest of $H$. Then there exists a maximal spanning forest $T = (V, F)$ of $G$ such that* $\text{load}(G, T) \leq \text{load}(H, T') + 2$.

*Proof.* Consider the graph $K = (V, (E - E') \cup F')$, which is a forest except that it preserves edges on the infinite face of $G$. Let $T = (V, F)$ be a maximal spanning forest of K such that $T' \subseteq T$. That is, T is obtained by adding edges from $E - E'$ to $T'$.

We will now show that if an arbitrary non-tree edge $a$ is in $\text{load}_{G,T}(e)$, then $a \in \text{load}_{H,T'}(e)$ or $a \in \text{load}_{K,T}(e)$. This will suffice in order to prove that $\text{load}(G, T)$ is at most $\text{load}(K, T) + \text{load}(H, T')$. Let us examine the different cases: The edge $a$ is either on the infinite face or not. If it is, then $a \in \text{load}_{K,T}(e)$, because the only difference between $G$ and $K$ is that $G$ contains some non-tree edges not on the infinite face. If it is not, consider whether the fundamental cycle of $a$ contains any edges on the infinite face. If it does not, $a \in \text{load}_{H,T'}(e)$, because $T'$ differs from $T$, and $H$ differs from $G$, only in that the former contains no edges on the infinite face. If it does, consider the following.[1] With respect to $T$, the edge $a$ has only one fundamental cycle. This cycle includes an edge on the infinite face. Thus it has no fundamental cycle in $T'$, and so $T' + a$ contains no cycle. So $T' + a$ is a larger maximal spanning forest than $T'$ on the same graph. Contradiction! Therefore, this case cannot occur.

It follows that $\text{load}(G, T) \leq \text{load}(K, T) + \text{load}(H, T')$. According to Lemma 20, $\text{load}(K, T) \leq 2$, so $\text{load}(G, T) \leq 2 + \text{load}(H, T')$. □

We need only consider graphs of degree at most 3. Graphs of higher degree can be transformed quite easily.

**Lemma 22.** *For every $k$-outerplanar graph $G$, there exists a $k$-outerplanar graph $H$ of degree at most 3 such that $G$ is a minor of $H$.*

*Proof.* We assume a $k$-outerplanar embedding for $G$ and give one for $H$. Each vertex $v$ of degree $d \geq 4$ and layer $\ell$ is replaced by a chain of $d - 2$ vertices of degree 3 as follows:

Let $w_1$ and $w_2$ be two neighbours of $v$ on layer $\ell$. (We know that $v$ must have one neighbour on the same layer, or else we could 'flip' the inside out and thus reduce the outerplanarity. If $v$ has just one neighbour on that layer, we have $w_1 = w_2$.) Now, we let $w_1$ and $w_2$ be the endpoints of the chain replacing $v$. In $G$, there was a certain ordering of edges incident to $v$, given by the planar embedding. This ordering is maintained in the chain.

[1] Thanks to Frederic Dorn for coming up with how to deal with this case.

After having treated all vertices in $G$ to arrive at $H$, $H$ clearly has maximum degree 3. The chain $C(v)$ is in layer $\ell$ just like $v$, so $H$ is $k$-outerplanar.

We can obtain $G$ from $H$ by contracting all edges between vertices in $C(v)$ for every vertex $v$ which has been replaced by a chain, preserving the order of the edges. In other words, $G$ is a minor of $H$. □

We reach a conclusion regarding graphs of low degree:

**Lemma 23.** *Let $G = (V, E)$ be a $k$-outerplanar graph of degree at most 3. Then there exists a maximal spanning forest $T = (V, F)$ such that* $\text{load}(G, T) \leq 2k$.

*Proof.* We will prove this by induction. First, consider the case $k = 1$. Let $T'$ be the resulting graph when removing all edges on the infinite face of $G$. For the purpose of contradiction, assume that $T'$ has a cycle $C'$. The edges of $C'$ cannot be on the infinite face of $G$, so some edges $C$ must be outside $C'$ in $G$. Let $v$ be a vertex in $C'$ incident to an edge $g$ in $C$. Let $e$ and $f$ be the edges incident to $v$ in $C'$. Now, $g$ can be outside either $e$ or $f$, but not both. So $C$ must contain yet another edge $h$ which is outside the other one. This means that $v$ has at least four incident edges in $G$: $e$, $f$, $g$ and $h$. This contradicts the fact that $G$ has degree at most 3. Therefore, $T'$ has no cycle, i.e., it is a forest. Obviously, $\text{load}(T', T') = 0$. Then, Lemma 21 says that $\text{load}(G, T) \leq 0 + 2 = 2$. This proves the basis step.

We now assume that the lemma holds for $k < \ell$, and show that it then holds for $k = \ell$. The result then follows. Let $H$ be the resulting graph when removing all edges on the infinite face of G. Recall that removing all *vertices* on the infinite face reduces the outerplanarity by one. The only difference is that $H$ has some extra vertices of degree zero or one, and this does not prevent it from being $(\ell - 1)$-outerplanar. Lemma 21 now says that $\text{load}(G, T) \leq 2(\ell - 1) + 2 = 2\ell$. □

**Lemma 24 (Robertson and Seymour).** *If $G$ is a minor of a graph $H$, then* $\text{bw}(G) \leq \text{bw}(H)$.

*Proof.* If $G$ has one or zero edges, its branchwidth is zero by definition, so the inequality holds for any $H$ in this case. Assume that $G$ has at least two edges. Let $(T_b, \sigma)$ be an optimal branch decomposition of $H$. We need to modify this to attain a branch decomposition for $G$. Since $G$ is smaller than $H$, we need only part of $T$ and $\sigma$. Let $S$ be a minimal subtree of $T$ such that every edge of $G$ has a representative in $S$. Let $T'$ be obtained from $S$ by suppressing all vertices of degree two. Let $\sigma'$ be the restriction of $\sigma$ to the edges of $G$. By the same argument as in the proofs of Lemmas 16 and 17, $(T', \sigma')$ is a branch decomposition of $G$, and its width is no greater than the branchwidth of $H$. The result follows. □

(This proof was based on Robertson and Seymour's original proof [RS91].) We can finally give an upper bound for the branchwidth:

**Theorem 25.** *Let $G$ be a $k$-outerplanar graph. Then $G$ has branchwidth at most $2k + 1$.*

*Proof.* First, wlog. we assume that $G$ is connected. By Lemma 22, there exists a $k$-outerplanar graph $H$ of degree at most 3 such that $G$ is a minor of $H$. By Lemma 23, there exists a maximal spanning forest $T$ of $H$ such that $\mathrm{load}(T, G) \leq 2k$. Let $(T_b, \sigma)$ be a spanning branch decomposition for $G$ given $T$. Lemma 19 then tells us that the width of $(T_b, \sigma)$ is at most $\max\{2, \mathrm{load}(T) + 1\} \leq \max\{2, 2k + 1\} = 2k + 1$. So $\mathrm{bw}(H) \leq 2k + 1$. By Lemma 24, $\mathrm{bw}(G) \leq \mathrm{bw}(H) \leq 2k + 1$. □

## 3.3 Face-vertex walks revisited

**Corollary 26.** *For any planar graph $G$,* $\mathrm{bw}(G) \leq \alpha_G + 1$.

*Proof.* If $\alpha_G = \max_x \alpha_x = 1$, the shortest face-vertex walk from the outer face of $G$ to $x$ has length 1. This means that the graph has one face only, so it must be a forest, and we have $\mathrm{bw}(G) \leq 2$.

If $\alpha_G \geq 2$, Lemma 13 states that $G$ is $\lfloor \alpha_G/2 \rfloor$-outerplanar. Then the theorem implies that $\mathrm{bw}(G) \leq 2\lfloor \alpha_G/2 \rfloor + 1 \leq \alpha_G + 1$. □

This result is weaker than Tamaki's result by an additive factor of 1; Tamaki stated that the branchwidth is at most $\alpha_G$.

# Chapter 4

# Conclusion

We have looked at various decompositions of graphs, and concentrated on branch decompositions of $k$-outerplanar graphs. Bodlaender [Bod96] has previously shown how easily a spanning tree with interesting properties can be found for these graphs. He used these properties to make tree decompositions of width no more than $3k - 1$. In this thesis, we have seen how one of these properties, stating an upper limit on what we have called *load*, can be used for making branch decompositions width no more than $2k + 1$ in a simple manner. We have not investigated the running time of this algorithm, but it can probably be implemented to run in linear time.

In March 2003, Tamaki [Tam03] published a heuristic producing a branch decomposition of width at most $\alpha_G$ for a graph $G$. This invariant, $\alpha_G$, was invented by Tamaki in the same paper. In this thesis, we have by simple means shown that $\alpha_G$ is roughly twice the outerplanarity of $G$. This implies that Tamaki's heuristic, given a $k$-outerplanar graph, finds a branch decomposition of width no more than $2k$. This is slightly better than my algorithm. It might be possible to improve Algorithm 15 so that it yields the same width, but this has not been investigated.

# Bibliography


[Alb02] Jochen Alber. *Exact Algorithms for NP-hard Problems on Networks: Design, Analysis, and Implementation.* PhD thesis, University of Tübingen, Oct 2002.

[Bie04] Therese Biedl. Cs762: Graph-theoretic algorithms, lecture 27: $k$-outerplanar graphs, 2004. University of Waterloo. `http://web.archive.org/web/www.student.cs.uwaterloo.ca/~cs762/Notes/old/lecture27.ps`.

[Bod96] Hans L. Bodlaender. A partial $k$-arboretum of graphs with bounded treewidth. Technical Report UU-CS-1996-02, Utrecht University, 1996.

[Die00] Reinhard Diestel. *Graph Theory.* Springer-Verlag, 2000.

[RS91] Neil Robertson and Paul D. Seymour. Graph minors X: obstructions to tree-decomposition. *J. Combinatorial Theory, Series B*, 52:153–190, 1991.

[Sip97] Michael Sipser. *Introduction to the Theory of Computation.* PWS Publishing Company, 1997.

[Tam03] Hisao Tamaki. A linear time heuristic for the branch-decomposition of planar graphs. Research report MPI-I-2003-1-010, Max-Planck-Institut für Informatik, Mar 2003.

[Wil96] Robin J. Wilson. *Introduction to Graph Theory.* Longman, 1996.