Universität Konstanz

Fachbereich Mathematik und Statistik

Diplomarbeit

Fixed Parameter Algorithms on Planar Graphs

von

Boris Alexander Köpf

Konstanz, Oktober 2002

Aufrichtiger Dank gebührt Frau Prof. Dr. Dorothea Wagner für die Unterstützung und Betreuung bei dieser Arbeit. Bedanken möchte ich mich auch bei Steffen Mecke und Marco Gaertler für wichtige Anregungen und Hilfen. Nicht zuletzt geht grosser Dank an meine Freunde und meine Familie, die mir schwierigen Phasen immer beigestanden haben.

Contents

Chapter 1

Introduction

Fixed parameter algorithms are nothing new. Computer scientists have always specified running times of algorithms in terms of natural problem dependent parameters, in many cases as a function of two or more variables. See for example [13, p. 106].

Classic complexity theory neglects this fact. Time or space complexity of an algorithm are usually formally defined as a function of exactly *one* natural number n: The maximum of the algorithm's running time or space consumption regarding all problem instances of size n. An encoding scheme is used to encipher the information that characterizes an instance of a given problem. This code's length is defined as the instance size - regardless of its semantics.

Parameterized complexity theory (see [11, 12]) formalizes to some extent what algorithm designers have been doing for a long time: It allows for a more differenciated encoding of problem instances and hence for a more detailled theoretical analysis of what can be regarded tractable by computation.

The basic idea is simple: The instance is split in two parts, one called the *main part* and the other the *parameter*. Complexity of an algorithm for such a *parameterized problem* is consequently stated as a function of two variables. If, for a given parameter, it is polinomial in the main part's size, the problem is regarded as *fixed parameter tractable*. The class of fixed parameter tractable problems is intended to consist of all problems that are solvable in practise – as long as the parameter remains "small". Besides from extending the notion of tractability by computation, tailor-made concepts of reduction and completeness provide new playgrounds for complexity theorists.

With this new complexity theoretic viewpoint, many well known results can be given new names. Also, many new algorithm design tools have been developed. These include reduction to a problem kernel, the method of bounded search trees and algorithms for partial $r-$trees, to name just a few. All of those tools yield typical running times of $\mathcal{O}(n^c d^k)$, with positive constants c, d, with n as the main part's size and k as the parameter's size.

In [4], a new tool for designing fixed parameter algorithms was recently developed. It allows the construction of procedures to decide a certain family of $\mathcal{NP}-$complete problems on the important class of *planar graphs*. What is interesting about this new tool is that instead of the above mentioned typical running times it allows the construction of algorithms with *subexponential time complexity*, namely of the form $\mathcal{O}(n^c d^{\sqrt{k}})$ with constants c and d. As was pointed out in [4], this is the first known nontrivial – in terms of subexponential – parameterized complexity result for $\mathcal{NP}-$complete problems, making it an interesting object of investigation.

The aim of this diploma thesis is to present this new method, apply it to new problems and investigate its strength.

Preliminaries and organization: In the first chapter of this work, some fundamental concepts are introduced and revisited. The reader is assumed to have basic knowledge in the areas of algorithm design, complexity theory, graph theory and elementary topology. Chapter 2 will mainly be concerned with the presentation and analysis of the aforementioned design tool for parameterized algorithms. The third part of this thesis investigates the strength of the presented methods. This is done by applying it to some well known $\mathcal{NP}-$complete problems.

Throughout this paper, an attempt has been made to introduce concepts no earlier that necessary, in order to emphasize the motivation behind them. Consequently, a more detailled outline of this thesis' content is given in section 1.4, directly after having introduced the basic toolkit.

To begin with, here are some basic definitions on parameterization.

1.1 Parameterized Complexity

1.1.1 Basic definitions

As it is common practice, we first restrict ourselves to decision problems and language recognition. Let Σ be a finite alphabet. The Kleene closure of Σ is denoted by Σ^*. For a word $x \in \Sigma^*$ we write $|x|$ for the number of characters it contains.

A subset $L \subseteq \Sigma^* \times \Sigma^*$ is called a *parameterized problem*. For $\langle x, k \rangle \in L$, x is called the *main part* and k the *parameter*.

Definition 1.1. Let L be a parameterized problem. We call L *fixed parameter tractable* if there is a function $f : \mathbb{N} \to \mathbb{N}$, a constant $c \in \mathbb{N}$ and an algorithm that correctly decides if $\langle x, k \rangle \in L$ in at most $f(|k|)|x|^c$ steps. We define $\mathcal{FPT}$ as the class of all fixed parameter tractable parameterized problems.

Note that this definition of fixed parameter tractability leaves room for abuse: the function f is not even required to be recursive. There are both weaker and more restrictive variants of this notion (see [11]). In this text, however, all examples will be of the above form, with f being recursive.

For a fixed $k \in \Sigma^*$ we define $L_k = \{\langle x, k \rangle : \langle x, k \rangle \in L\}$ as k's *slice*. If $L \in \mathcal{FPT}$, the question $\langle x, k \rangle \in L_k$ can be decided in polinomial time for any fixed k. This is why a problem in $\mathcal{FPT}$ is said to be *tractable by the slice*.

As it is standard practice, we often state running times in terms of abstract problem specific parameters instead of the length of their encoding. This is done assuming a reasonable encoding scheme is chosen. For details, see [13]. To emphasize this we will then denote problems by Π instead of L.

1.1.2 Optimization problems

One class of problems that allows natural parameterizations is the class of optimization problems. The standard method of turning an optimization problem into a decision problem leads straight to a parameterized problem. The following definition is taken from [13]:

Definition 1.2. An *optimization problem* Π is either a minimization problem or a maximization problem and consists of a triple $(\mathcal{I}_\Pi, \mathcal{S}_\Pi, \nu_\Pi)$ with:

(i) $\mathcal{I}_\Pi$ as the set of *input instances*

(ii) $\mathcal{S}_\Pi(I)$ as a finite set of *feasible solutions* for each $I \in \mathcal{I}_\Pi$

(iii) the *solution value* $\nu_\Pi : (I, S) \mapsto \nu_\Pi(I, S) \in \mathbb{N}$ for every $I \in \mathcal{I}_\Pi$ and $S \in \mathcal{S}_\Pi(I)$

Given an optimization problem Π and an instance I, the aim is to find a feasible solution $S' \in \mathcal{S}_\Pi(I)$ such that $\nu_\Pi(I, S')$ is minimal – or maximal, respectively – among all feasible solutions $S \in \mathcal{S}_\Pi(I)$. An optimal solution is denoted by $opt_\Pi(I)$. We sometimes write $\nu_\Pi(S)$ instead of $\nu_\Pi(I, S)$ when the choice of I is clear from the context.

1.1.2.1 A natural parameterization

The *decision version* of a minimization problem Π can be stated as follows:

$$\textit{Is there a solution } S \textit{ of } \Pi \textit{ on } I \textit{ , such that } \nu_\Pi(I, S) \leq k \textit{ holds?}$$

In the classic sense, an optimization problem Π leads to a decision problem *for every natural number* k. From the parameterized viewpoint it leads to *one* parameterized problem with instances of the form $\langle I, k \rangle$.

For the sake of completeness we note that there are connections between the concepts of approximability and fixed parameter tractability. Using a slightly stronger notion of optimization problems, Cai and Chen proved that the parameterized version of an optimization problem Π is in $\mathcal{FPT}$ once Π allows for a fully polinomial time approximation scheme (FPTAS). For a proof or a definition of FPTAS, see for example [11].

To distinguish between an optimization problem and its parameterized version, we leave out terms like “maximum” or “minimum” in the latter, when the situation is clear from the context.

1.1.2.2 Some examples

Here are some examples of parameterized problems that will be used throughout the text. All of them are obtained from well-known optimization problems following the recipe above. See for example [11, 13]. For the concept of graphs, see section 1.2.

VERTEX COVER

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a set $V' \subseteq V$ with $\|V'\| \leq k$, such that every edge has an endpoint in V'.

INDEPENDENT SET

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a set $V' \subseteq V$ with $\|V'\| \geq k$, such that for all $u, v \in V'$ the relation $(u, v) \notin E$ holds.

DOMINATING SET

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a set $V' \subseteq V$ with $\|V'\| \leq k$, such that every $v \in V$ is either a member of V' or adjacent to some $w \in V'$.

INDEPENDENT EDGE SET

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a set $E' \subseteq E$ with $\|E'\| \geq k$, such that no two edges $e, e' \in E'$ share a common vertex.

EDGE DOMINATING SET

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a set $E' \subseteq E$ with $\|E'\| \leq k$, such that every $e \in E$ is either a member of E' or adjacent to some $e' \in E'$.

$s-t-$PARTITION (see [7])

Instance:	A graph $G = (V, E)$
Parameter:	$k \in \mathbb{N}$
Question:	Is there a partition $V = V_1 \uplus V_2$ with $\lvert V_1\rvert = s$ and $\lvert V_2\rvert = t$ such that there are at most k edges $(v, w) \in E$ with $v \in V_1$ and $w \in V_2$.

Later on we will examine some derivatives of these problems. Their definition will be straightforward: For example PLANAR EDGE DOMINATING SET will be nothing but EDGE DOMINATING SET with the class of instances restricted to planar graphs.

1.2 Graphs and more

In this section we introduce the concepts of graphs, planarity and $r-$outerplanarity and the notion of separators together with some related computational aspects.

Most of this can be regarded as common knowledge and will therefore be dealt with only briefly. If not mentioned otherwise, the upcoming definitions are abbreviated derivatives of definitions in [16]. For a more detailed presentation refer to the original monograph.

1.2.1 Graphs

A *simple graph* $G = (V, E)$ is a finite set V of *vertices* together with a set $E \subseteq V \times V$ of *edges*. If E is symmetric, we sometimes regard E as a set of unordered pairs of vertices and call G *undirected*. When E is irreflexive, we say that G *contains no self–loops*. A *multigraph* $G = (V, E)$ is a set of vertices together with a finite sequence of edges $E = (e_i)_{1 \leq i \leq m}$ with $e_i \in V \times V$ for $i = 1, \ldots, m$. By definition, a multigraph allows multiple edges between two vertices, whereas a simple graph does not.

For both types of graphs, a *subgraph* $H = (V', E')$ of G is a graph with $V' \subseteq V$ and $E' \subseteq E$. For a set $V' \subseteq V$ we denote by $G[V']$ the subgraph *induced* by V', that is, the graph with vertex set V' together with all edges with both endpoints in V'.

As long as nothing else is mentioned, when we speak of a *graph* we mean a *simple undirected graph without self-loops*. Following common practice, we always use n for $|V|$ and m for $|E|$.

For common data structures used to represent graphs in a computational context, refer to [9].

1.2.2 Planarity and $r-$outerplanarity

Let $x, y \in \mathbb{R}^2$. A *x-y-path* is a continuous function $\alpha : [0,1] \to \mathbb{R}^2$ with $\alpha(0) = x$ and $\alpha(1) = y$. A *drawing* of a graph $G = (V, E)$ is an injective function ϕ that maps every vertex $v \in V$ to a point $\phi(v) \in \mathbb{R}^2$ and every edge (v, w) to a $\phi(v) - \phi(w)-$path. We say that two distinct edges $e, e' \in E$ *cross* if $\phi(e)\,[0,1] \cap \phi(e')\,[0,1]$ not only contains common endpoints. We call a drawing ϕ a *planar embedding* if no edges cross. A graph is called *planar* if it has a planar embedding. A *plane graph* (G, ϕ) is a planar graph G together with a planar embedding ϕ.

Elementary topology shows that the image of a planar embedding ϕ, the set $\bigcup_{e \in E} \phi(e)[0,1]$, is a closed set in $\mathbb{R}^2$. Hence, its complement is an open subset of $\mathbb{R}^2$. Its connected components are called *faces*. By a compactness argument one sees that there is exactly one unbounded face. Call this face the *exterior face*.

A very important result due to Euler connects the number of vertices and edges in planar graphs.

Proposition 1. *For a simple planar graph $G = (V, E)$ the following equation holds:*

$$3|V| - 6 \geq |E|$$

For a proof see [16, pp. 256], for example. This result will come in very useful when analyzing running times of algorithms on planar graphs, for it implies $|E| \in \mathcal{O}(|V|)$.

The following definition is taken from [4]:

Definition 1.3. Let $G = (V, E)$ be a planar graph, ϕ an embedding. The *layer decomposition* of (G, ϕ) is a partition of V into sets $L_1, \ldots, L_r$ as follows:

(i) L_1 is the set of vertices on the exterior face of (G, ϕ).

(ii) L_i is the set of vertices on the exterior face of $G\left[V \setminus \bigcup_{j=1}^{i-1} L_j\right]$ for $i = 2, \ldots, r$.

The set L_i is called the *ith layer* of (G, ϕ), its vertices are called *level i vertices.*

The number of layers r is defined as the *outerplanarity* of the embedding. The outerplanarity of G is defined as the minimum outerplanarity over all of its planar embeddings and is denoted by $out(G)$. A graph with outerplanarity 1 is called *outerplanar.*

In the following we will sometimes assume an arbitrary planar embedding is given and write G instead of (G, ϕ). There is an algorithm that, given a planar graph $G = (V, E)$, calculates such an embedding in time $\mathcal{O}(|V|)$. For details, see [8].

1.2.3 Separators and Fragments

The notion of separators is of great importance in the upcoming work. Therefore a separate paragraph is dedicated to them.

Let $G = (V, E)$ be a graph. A subset $S \subseteq V$ is a *separator*, if there is a disjoint partition $V = V_1 \uplus S \uplus V_2$ such that no edge in E joins vertices in V_1 and V_2. A triple (V_1, S, V_2) with that property is called a *separation.* Note that this definition includes border cases such as empty sets S and V_i for $i = 1, 2$.

Following [4], we introduce a name for a special kind of separator. Let G be a plane graph with layer decomposition $(L_i)_{1 \leq i \leq r}$ and $S \subseteq \bigcup_{i=i_0}^{i_1} L_i$ for some indices $1 \leq i_0 \leq i_1 \leq r$. We say that S *separates layers* L_{i_0-1} *and* L_{i_1+1} if there is a separation (V_0, S, V_1) with $\bigcup_{i=1}^{i_0-1} L_i \subseteq V_0$ and $\bigcup_{i=i_1+1}^{r} L_i \subseteq V_1$.

In the upcoming chapters, we will make extensive use of separators. Mostly, we deconstruct a graph $G = (V, E)$ into two subgraphs sharing the nodes in in the separator, with their union containing all edges in E. To keep notation simple, we call a pair of subgraphs $G_1 = (V_1 \cup S, E_1)$ and $G_2 = (V_2 \cup S, E_2)$ with $E_1 \cup E_2 = E$ the *fragments* of the separation (V_1, S, V_2).

1.3 The main ingredients

Our aim is to present the techniques and ideas for designing subexponential fixed parameter algorithms on planar graphs. The key ingredients are twofold:

- a method that allows to design $\mathcal{O}(nc^r)$ algorithms for many $\mathcal{NP}-$complete planar graph problems, where n is the size of the input graph, r its outerplanarity and c a problem-dependent constant.
- a method for deconstructing the main part G of an instance $\langle G, k\rangle$ of a parameterized problem on planar graphs into $\mathcal{O}(\sqrt{k})-$outerplanar components using separators with size bound $\mathcal{O}(\sqrt{k})$.

Before presenting the fusion of the two algorithmic methods in chapter 2, they are first discussed individually in their original context.

1.3.1 Baker's algorithm

The first of the previously mentioned methods was presented by Baker in 1983 and published as a journal article [5] in 1994. It consists of a generic approach to optimization problems using dynamic programming techniques.

Once the method is adapted to a concrete $\mathcal{NP}-$complete optimization problem Π, it allows to finds an optimal solution for a planar instance $G = (V, E)$ in running time $O(nc^r)$, with $n = |V|$, $r = out(G)$ and c as a problem-dependent constant. If we restrict our attention to the class of $r-$outerplanar graphs it thus leads to linear time algorithms.

Different techniques have been developed to overcome the restriction to that very special class of graphs.
In [5], a method to obtain approximation schemes for Π on the class of *planar* graphs is presented. It is based on separating the original instance G into $r-$outerplanar subgraphs for an arbitrary r, and the application of the aforementioned linear time algorithm to every such subgraph. By suitably combining the solutions on the subgraphs, an approximation for the solution of Π on G is obtained.
The approach presented in [4] can be seen as another possibility for extending

Baker's algorithm to the class of planar graphs. Here, the result is a fixed parameter algorithm.

While the fixed parameter approach is discussed in full detail later on, we will give a short glimpse into the the design of approximation algorithms for completeness' sake.

1.3.1.1 Designing approximation algorithms

The details of Baker's algorithm will be presented in sections 2.2 and 2.4.3.1, for the next two paragraphs suppose an algorithm with the above properties is given. We follow the approach of [5] and sketch the idea using a sample problem. Due to the solution's structure, MAXIMUM INDEPENDENT SET suits our purposes best.

Let $G = (V, E)$ be an embedded planar graph with layer decomposition $L_1, \ldots, L_p$ and let $V' \subseteq V$ with $|V'| = M$ be a maximum independent set on G. Observe that, due to planarity, every L_j is a separator, separating $\bigcup_{i=1}^{j-1} L_i$ from $\bigcup_{i=j+1}^{p} L_i$.

Choose an arbitrary $r \in \mathbb{N}$. We split the index set $I := \{1, \ldots, p\}$ into $r+1$ disjoint sets $I_q := \{i \in I : i \equiv q \bmod r+1\}$ for $q = 0, \ldots, r$ and partition V into $C_0, \ldots, C_r$ with $C_q := \bigcup_{i \in I_q} L_i$ accordingly. Note that there must be at least one index j_0 with

$$|C_{j_0} \cap V'| \leq \frac{M}{r+1} \tag{1.1}$$

for it would otherwise contradict the maximality of V'.

Suppose now we are given such an index j_0. The graph $G[V \setminus C_{j_0}]$ has components of outerplanarity bounded by r. Observe that V' cuts down to - not necessarily maximum - independent sets on the $r-$outerplanar components. By (1.1), their union has a size of at least $M\frac{r}{r+1}$. On the other hand, the union of *arbitrary* independent sets on each of the disjoint components forms an independent set on G. Taking the union of *maximum* independent sets thus leaves us with an independent set on G with a size of at least $M\frac{r}{r+1}$.

To find the desired approximation we do the following: for every $j = 0, \ldots, r$ we calculate a maximum independent set for each of the $r-$outerplanar graph components. Taking the union of those sets for a fixed j we obtain an inde-

pendent set on G. We accept the independent set of maximum cardinality among all j's as an approximation for the maximum independent set on G.

This leads to an approximation algorithm with running time $\mathcal{O}(nrc^r)$ and approximation ratio of at most $\frac{r+1}{r}$. For a precise definition of these concepts, see for example [13].

A similiar approach can be used to design approximation schemes for problems such as MINIMUM VERTEX COVER (see [13]), MINIMUM DOMINATING SET and many more. For details, refer to [5].

1.3.2 Alber's approach

The second key ingredient was developed in [4]. A parameterized problem on planar graphs fulfilling the so called *Layerwise Separation Property* exhibits very useful characteristics: for any main part $G = (V, E)$ of a *yes*–instance $\langle G, k\rangle$, we can find a pairwise disjoint family of separators $(S_i)_{1\leq i\leq s}$ for some $s \in \mathbb{N}$, with $|S_i| \in \mathcal{O}(\sqrt{k})$ for $1 \leq i \leq s$.
With their help, G can be split into a sequence $(G_i)_{0\leq i\leq s}$ of $\mathcal{O}(\sqrt{k})-$outerplanar subgraphs, with G_i and G_{i+1} sharing only the nodes in S_{i+1} for every $i = 0, \ldots, s$.

Having obtained this separation, the approach in [4] offers two possibilities: one is based on the fact that the *treewidth tw* (see e.g. [6]) of a planar graph G can be related to its outerplanarity by

$$tw(G) \leq 3out(G) - 1 \tag{1.2}$$

Such a tree decomposition can be found in $\mathcal{O}(out(G)|G|)$ time.
Using the above separation and a tree decomposition for every component G_i according to equation (1.2), one can easily construct a tree decomposition of size $\mathcal{O}(\sqrt{k})$ for the entire graph G. Many well known $\mathcal{NP}-$hard graph problems allow for $\mathcal{O}(n^d c^r)-$algorithms with constants d and c, as soon as the instances allow for a a tree decomposition of size r. Putting all of this together one obtains $\mathcal{O}(n^d c^{\sqrt{k}})-$algorithms. For details on the construction, see [4], for an overview of the concept of tree decompositions, refer to [6].

Another possibility for using the separation is by directly applying algorithms based on bounded outerplanarity to each of the components G_i - and piecing together the obtained solutions. One important tool for solving prob-

lems on $\mathcal{O}(\sqrt{k})-$outerplanar graphs is the previously mentioned algorithm by Baker.

While the tree decomposition based approach allows for a mathematically elegant description, the approach relying on [5] is geometrically motivated and more direct. For this work, we decided to focus on the combination of the approaches in [4] and [5].

1.3.3 Combining both approaches

As Baker's algorithm is tailor-made for optimization problems, and Alber's approach deals with parameterized problems, we will from now on exclusively be dealing with parameterized versions of optimization problems according to section 1.1.2.

First, we take a look at the difficulties that occur when combining the methods introduced in 1.3.1 and 1.3.2. When applying the outerplanarity-based approach to an instance $\langle G, k\rangle$ of a parameterized problem Π, one is confronted with solving the restricted problem on the $\mathcal{O}(\sqrt{k})-$outerplanar subgraphs G_i of the original graph G for $i = 0, \ldots, s$.

To ensure that solutions of Π on G_i and G_{i+1} "fit" together on the common nodes S_{i+1}, certain boundary conditions have to be fulfilled. In the case of Vertex Cover, for example, one will have to look for minimum vertex covers V_i' of G_i and V_{i+1}' of G_{i+1} *under the constraint* that $V_i' \cap S_{i+1} = W = V_{i+1}' \cap S_{i+1}$ holds for a certain node set $W \subseteq S_{i+1}$.

Thus, not the original problem Π has to be solved, but a closely related problem Π' - the *constrained* version of Π. For some problems Π, this difficulty can be overcome by a slight modification of the graph structure. In the example of Vertex Cover, a dummy neighbor is attached to every $v \in W$ in G_i and G_{i+1}. As either v or its dummy neighbor must be contained in in any vertex cover of the thus obtained graphs G_i' and G_{i+1}', it can easily be seen that the size of a minimum vertex cover on G_i' equals the size of a *constrained* minimum vertex cover on G_i. Having made this transformation, Baker's algorithm for Vertex Cover can be applied without modification.

However, the situation is not that simple in all cases. In [4] it was already pointed out that in the case of Dominating Set there is no known modification of the constrained version to an equivalent unconstrained one - and

therefore no way of using Baker's algorithm as a "black box". The same situation seems to occur for Edge Dominating Set, Minimum Maximal Matching and many more.

Because of these difficulties we decided not to follow the approach in [4] and opened the black box. In adapting Baker's algorithm to constrained problems, we found that there are many similarities in the dynamic programming techniques that are underlying both algorithms. Consequently, we tried to work out these similarities and chose a unified approach to present both algorithms.

This point of view has three main advantages: firstly, we overcome the difficulties that arise when dealing with constrained problems in the above way. Secondly, we are able to state a sufficient condition for a parameterized problem to be handled by *both* of these algorithms. Thirdly, we are able to precisely characterize what is meant by *optimal substructure property*, "one of the hallmarks of dynamic programming" (see [9]) in our case.

1.4 Contents and organisation

In section 2.1 we present the notion of *Layerwise Separation Property* introduced in [4]. In 2.1.4, we examine methods to obtain small separators that cut instances into subgraphs of bounded outerplanarity.
In section 2.2, we present the recursive decomposition of $r-$outerplanar graphs into small subgraphs, so called *slices*, due to [5]. We give an interpretation of the original ideas by the consequent use of small separators.
Subsequently, section 2.3 develops the common theoretical viewpoint for both approaches. We will examine when and how we can construct a problem's solution on G from solutions on subgraphs of G.
After the description of the decomposition and the theoretical foundations, the pseudocode is given in section 2.4.
In section 2.5, this approach is analyzed in terms of correctness and running time. Finally, chapter 3 is concerned with investigating the strength of the presented concepts.

Our contributions Our own contributions are spread throughout the text. First, we tried to find a mathematically sound, yet not too formal method

of description. This resulted in a thoroughly reworked mode of presentation, including some missing proofs, especially in sections 2.2 and 2.4. Secondly, we worked out how to get hold of the separators in section 2.1.4, as in [4] only their existence is proven. Thirdly, the considerations of the strength of the presented concepts made in chapter 3 consist of original work

Fourthly, and most importantly – the concept of *dissectability* and the resulting consequences developed in section 2.2. Where in [5] there are no general considerations on the applicability of the concepts, the ideas in [4] are hidden behind abundant formalism. Both approaches rely on dynamic programming techniques and therefore require that the regarded problems have the *optimal substructure property* (see e.g. [9]). However, this fact comes out only implicitly. The notion of *dissectability* is a simple explicit formalization of this property in the context of graph optimization problems. We will see that this transparent notion provides a sufficient condition for the applicability of both algorithms.

Chapter 2

Designing subexponential FP-algorithms

2.1 From planar to $\mathcal{O}(\sqrt{k})-$outerplanar graphs

The key notion that allows us to prove that, given a parameterized problem Π, we can find the desired partition for every instance $\langle G, k\rangle$ – with both the size of the separators and the outerplanarity of the obtained components bounded by $\mathcal{O}(\sqrt{k})$ – is the so called *Layerwise Separation Property* introduced by Alber et. al. in [4].

The first part of section 2.1.1 is concerned with defining these concepts. With the definitions at hand, Alber's theorem on the existence of suitable separators can be stated and proven. In the subsequent section 2.1.4, we show how we can algorithmically get hold of those separators.

2.1.1 Different kinds of separations

This section is essentially contained in [4], aside from differences in the mode of presentation and the use of the notion of *fragments* defined in section 1.2.3.

First, we give a name to a family of separators $S_1, \ldots, S_s$ with the property that every S_j is contained in at most w subsequent layers $L_{i_j}, \ldots, L_{i_j+w-1}$ of a graph's layer decomposition $L_1, \ldots, L_r$. For ease of notation, we always define $L_j = \emptyset$ for $j < 1$ and $j > r$.

Definition 2.1 (Separation of layers). Let $G = (V, E)$ a plane graph with $(L_i)_{1 \leq i \leq r}$ as its layer decomposition. Furthermore, let $w \in \mathbb{N}$.
A *separation of layers of width* w is a sequence $(S_j)_{1 \leq j \leq s}$ of subsets of V with the following two properties:

(i) there is sequence of layer indices $0 \leq i_1 \leq \cdots \leq i_s \leq r$ such that $S_j \subseteq \bigcup_{q=i_j}^{i_j+w-1} L_q$.

(ii) S_j separates layers L_{i_j-1} and L_{i_j+w}. for every $1 \leq j \leq s$

Not every separation will serve our purposes. Our aim is to obtain $\mathcal{O}(\sqrt{k})-$ outerplanar components of an instance $\langle G, k \rangle$, all separated by separators with size bounded by $\mathcal{O}(\sqrt{k})$. Thus we need a restriction on the size of the separating sets as well as a bound for the difference of any two succeeding layer indices i_j and i_{j+1}. These are the defining properties of a *balanced separation of layers*. We recycle the notation from 2.1.

Definition 2.2 (Balanced). Let $\mathcal{S} = (S_j)_{1 \leq j \leq s}$ be a separation of layers of width w. We say that $\mathcal{S}$ is $\alpha - \beta-$*balanced*, if the following properties hold

(i) $|S_j| \leq \alpha\beta$

(ii) $i_{j+1} - i_j \leq \frac{\alpha}{\beta} + w$ and $i_1 \leq \frac{\alpha}{\beta} + w$

(iii) $i_j + w \leq i_{j+1}$

for every $j = 1, \ldots s$, where we define $i_{s+1} := r$.

It should be noted that the variable β serves as a trade-off parameter and will only be used for running time considerations. For better understanding, imagine $\beta = 1$.

To see how such a balanced separation can be utilized, suppose we are given a plane graph G together with its layer decomposition $(L_i)_{1 \leq i \leq r}$. Let $(S_j)_{1 \leq j \leq s}$ be an $\alpha-\beta-$balanced separation of width w. Use S_1 to separate G into a fragment $G_0 = (V_0 \cup S_1, E_1)$ with $\bigcup_{q=1}^{i_1-1} L_q \subseteq V_0$ and a graph $G'_1 = (S_1 \cup V'_1, E_2)$ with $E_1 \cup E_2 = E$ as its complementary fragment in the sense of paragraph 1.2.3 on page 14.

Reiterate this process with separators S_j of G'_{j-1} for $j = 2, \ldots, s$ and set $G_s := G'_{s-1}$ to obtain the sequence of graph fragments $G_0, \ldots, G_s$. By *(ii)*

we see that $out(G_j) \leq \frac{\alpha}{\beta} + 2w$ holds. With the help of *(iii)* we see that the separators are pairwise disjoint, and by *(i)* of definition 2.2 we see that their size is bounded by $\alpha\beta$.

Next we define yet another kind of separation, the so called *layerwise separation* credited to Alber et al. (see [4]). This concept plays a central role in what comes next, for it provides an easily proved sufficient condition for the existence of balanced separations. It is called *layerwise*, because the sequences $i_1, \ldots, i_s$ and $1, \ldots, r$ in definition 2.1.1 coincide, i.e. there is a separator for every layer.

Definition 2.3 (Layerwise separation). Let $G = (V, E)$ be a plane graph with $(L_i)_{1 \leq i \leq r}$ as its layer decomposition. A *layerwise separation of width w and size σ* of G is a separation of layers $(S_i)_{1 \leq i \leq r}$ of width w with:

(i) $S_j \subseteq \bigcup_{q=j}^{j+w-1} L_q$

(ii) $\sum_{j=1}^{r} |S_j| \leq \sigma$

Now we state a theorem that allows us to conclude from the existence of layerwise separations that of balanced separations of layers. We give its proof, because it contains ideas on how to find the desired separators. They will be taken up in section 2.1.4.

Theorem 2.1 (Alber et al.[4]). *Let $G = (V, E)$ be a planar graph, ϕ an embedding . Suppose (G, ϕ) admits a layerwise separation of width w and size σ. Then, for every $\beta > 0$, there exists a $\sqrt{\sigma} - \beta-$balanced separation of layers of width w.*

Proof. Let $L_1, \ldots, L_r$ be the layer decomposition. Partition the index set $I := \{1, \ldots, r\}$ into sets $I_q := \{i \in I : i \equiv q \bmod w\}$ for $q = 0, \ldots, w - 1$. Observe that by the size bound in *(ii)* of definition 2.3 there is a $q_0 \in \{0, \ldots, w-1\}$ such that

$$\sum_{j \in I_{q_0}} |S_j| \leq \frac{\sigma}{w} \tag{2.1}$$

holds. We thin out the set I_{q_0} set by defining $J := \{j \in I_{q_0} : |S_j| \leq \beta\sqrt{\sigma}\}$. The obtained separation $(S_j)_{j \in J}$ clearly fulfils conditions *(i)* and *(iii)* of the definition of a balanced separation in 2.2.

For item *(ii)* of definition 2.2 let $J = \{i_1, \ldots, i_{|J|}\}$ in nondecreasing order. Observe that the number of elements in I_{q_0} between two subsequent elements i_j and i_{j+1} of J is given by the equation

$$\frac{i_{j+1} - i_j}{w} - 1$$

Because of the definition of J, all of the corresponding separators have sizes greater than $\beta\sqrt{\sigma}$. By equation (2.1) we obtain

$$\left(\frac{i_{j+1} - i_j}{w} - 1\right)\beta\sqrt{\sigma} \leq \frac{\sigma}{w}$$

and so the desired

$$i_{j+1} - i_j \leq \frac{\sqrt{\sigma}}{\beta} + w$$

□

2.1.2 The Layerwise Separation Property

Until this point we have related layerwise separations with balanced separations of layers only for particular plane graphs. To design a decision algorithm for a parameterized problem Π we need balanced separations for every instance. In fact, it is enough to ensure such separations can be found for every *yes*–instance of Π, regardless of the embedding we use. If an instance does not allow for such a separation we then know it must be a *no*–instance and we can simply reject it. The concept of *Layerwise Separation Property* presented in [4] is the formal expression of this idea.

Definition 2.4 (Layerwise Separation Property). A parameterized problem Π on planar graphs is said to have the *Layerwise Separation Property (LSP)* of width w and *size-factor* d if, for every *yes*–instance $\langle G, k\rangle$ and every embedding ϕ of G, we can find a layerwise separation of width w and size dk.

2.1.3 The recipe

The presented ideas can be utilized to design algorithms in the following way: first, one proves the Layerwise Separation Property for the problem Π. Given a problem instance $\langle G, k\rangle$ together with an embedding ϕ, one uses the next chapter's algorithm to find a $\sqrt{dk}-\beta-$balanced separation for an arbitrary β. If this fails, we reject $\langle G, k\rangle$, for the Layerwise Separation Property together with theorem 2.1 implies that we are dealing with a *no*–instance. If we succeed, we go on using the methods of the subsequent chapters. They help us to finally decide whether to reject or accept $\langle G, k\rangle$.

2.1.4 Finding the Separators

In this section we take a look at methods that actually find a $\sqrt{\sigma}-\beta-$balanced separation of layers, assuming the existence of a layerwise separation of size σ.

Recycling the ideas and notation of the proof of theorem 2.1, our aim is to find an index q_0 that fulfils equation (2.1). To do so we simply test all index sets I_q with $q = 0, \ldots, w-1$ as follows: for every $i \in I_q$ we determine whether there is a $\sqrt{\sigma}\beta-$sized vertex set separating L_{i-1} and L_{i+w}. If two succeding indices with that property lie more than $\sqrt{\sigma}/\beta + w$ apart, q cannot be the q_0 we are looking for. If we eventually find q_0 with the desired property, we are done and can continue with the algorithms of the subsequent chapters. If our search fails, we return NIL.

2.1.4.1 Balanced Separation

The above idea is captured in a method called BALANCED SEPARATION. Called with arguments α, β, w and the layer decomposition, it tries to find a $\alpha-\beta-$balanced separation of layers of width w. It makes calls to a method called SEPARATOR, handing over $\alpha\beta$ and $(L_i)_{j-1\leq i\leq j+w}$ for some j. As we will see later on, SEPARATOR decides whether there is a vertex set of size $\alpha\beta$ separating L_{j-1} from L_{j+w} – and returns it in case it exists. Note however, that the separators obtained by these methods do not necessarily coincide with the ones in the existence proof of theorem 2.1.

BALANCED SEPARATION$((L_i)_{1\leq i\leq r}, \alpha, \beta, w)$

```
1  for q = 1 to w
2  do S_q ← ∅
3      j ← q, j_0 ← q
4      while j ≤ r and j − j_0 ≤ α/β + w
5      do if SEPARATOR((L_i)_{j−1≤i≤j+w}, αβ) returns S_j ≠ ∅
6            then S_q ← S_q ∪ {S_j}
7                 j_0 ← j
8         j ← j + w
9      if r − j_0 ≤ α/β + w
10       then return S
11 return NIL
```

The remaining problem is to determine an algorithm SEPARATOR that finds a minimum sized set of vertices that separate L_{j-1} from L_{j+w}. It can be solved using standard network flow methods. To be able to make use of them we first introduce some notation, most of which is taken from [9].

2.1.4.2 Some basic definitions on networks

A *flow network* $N = (G, s, t, c)$ is a *directed* graph $G = (V, E)$ with a *capacity* function $c : E \to \mathbb{R} \cup \{\infty\}$ and two distinguished vertices $s, t \in V$. We call s the *source* and t the *sink* of the network.

A $s - t-cut$ or simply *cut* (S, T) in N is a partition $V = S \uplus T$ with $s \in S$ and $t \in T$. The *capacity* $c(S, T)$ of a cut is defined as the sum of capacities of edges that leave S, that is, $c(S, T) := \sum_{v \in S} \sum_{w \in T} c(v, w)$. A *minimum cut* on N is a cut with minimum capacity among all cuts on N.

By a $s - t-path$ in a (directed or undirected) graph $G = (V, E)$ we mean a sequence $s = v_1, \ldots, v_p = t$ of vertices in V, such that $(v_i, v_{i+1}) \in E$ for $i = 1, \ldots, p-1$. A $s - t-separator$ in a (directed or undirected) graph $G = (V, E)$ with $s, t \in V$ is a subset $S \subseteq V \setminus \{s, t\}$, such that there is no $s - t-$path in $G[V \setminus S]$.

2.1.4.3 Separators and cuts

In this section we will describe how the problem of finding a set S of vertices that separate layers L_{j-1} and L_{j+w} is transformed to a maximum flow problem. Firstly, observe that finding such a set is equivalent to finding a minimum sized $s - t-$separator in the (undirected) graph obtained from $G\left[\bigcup_{q=j-1}^{j+w} L_q\right]$ by identifying all nodes in L_{j-1} and naming them s, and identifying all nodes in L_{j+w} to a new node t while conserving all edges. For simplicity of notation, assume from now on that we are given an undirected graph $H = (V, E)$ with two nodes $s, t \in V$.

Firstly, we create a flow network $\overline{H}$ for H with the property that the minimum cuts in $\overline{H}$ correspond to the minimum sized $s - t-$separators in H. Utilizing the Max-Flow-Min-Cut, duality a simple modification of the well-known Ford-Fulkerson-algorithm for finding maximum flows does the job.

We start off by creating the *Even-Tarjan-Reduct* $\overline{H}$ of H as described in [14].

In the first step we introduce two directed edges $(v,w),(w,v)$ for every undirected edge $(v,w) \in E$. In the second step, we split every vertex $v \in V$ into two vertices v', v'' connected by a directed edge (v', v''). The edges that led into vertex v are attached to v', the edges that parted from v are attached to v''.

Formally: $\overline{H} := (\overline{V}, \overline{E})$ with $\overline{V} := \{v' : v \in V\} \cup \{v'' : v \in V\}$ and $\overline{E} := \{(v', v'') : v \in V\} \cup \{(v'', w') : (v,w) \in E\} \cup \{(w'', v') : (v,w) \in E\}$

We give $\overline{H}$ flow network structure by defining $c(e) = 1$ for every edge of the form $e = (v', v'')$, with $v \in V \setminus \{s,t\}$ and $c(e) = \infty$ otherwise. We set s' as the source, t'' as the sink of the network $\overline{H}$. Due to the construction, there are close connections between H and $\overline{H}$. We point will out some of the important ones.

Proposition 2. *The $v - w-$paths of H are in 1-1 correspondence with the directed $v' - w''-$ paths of $\overline{H}$ by*

$$(v, v_1 ..., v_r, w) \longleftrightarrow (v', v'', v_1', v_1'', ..., v_r', v_r'', w', w'')$$

Proof. The definition of $\overline{H}$ shows that the right hand side defines a $v' - w''-$ path if the left hand side is a $v - w-$path in H. On the other hand, every $v' - w''-$path in $\overline{H}$ is of that form, because the only edge leaving a vertex u' – respectively entering a vertex u'' – is (u', u''), and all other edges in the $\overline{H}-$path directly correspond to edges in H. □

Proposition 3. *A minimal $s - t-$separator of cardinality d in H induces a cut of capacity d in $\overline{H}$*

Proof. Let $v_1, ..., v_d$ be a minimal $s - t$ separator in H. We show that the edge set $C := \{(v_1', v_1''), \ldots, (v_d', v_d'')\} \subseteq \overline{E}$ forms a cut. We define

$$S =: \{v \in \overline{V} : \text{there is a } s' - v - \text{ path in } \overline{H} \setminus C\}$$

and $T := \overline{V} \setminus S$. By the definition of S it is clear that the only edges going from S to T can be the the ones in C. It is also clear that $s' \in S$ and $v_1'', \ldots, v_d'' \in T$. Suppose now that $v_i' \notin S$ for an index $i \in \{1, \ldots, d\}$. That is, there is no path connecting s' and v_i' without using an edge of the set $C \setminus (v_i', v_i'')$ and therefore, according to proposition 2, no $s - v_i-$path in H without a vertex $\{v_1, \ldots, v_{i-1}, v_{i+1}, ..., v_d\}$. Then, $\{v_1, \ldots, v_{i-1}, v_{i+1}, ..., v_d\}$ is also a $s - t-$separator, contradicing the minimality assumption.

To show $t'' \in T$, assume there is a $s' - t''$ path in $\overline{H}$. According to proposition 2 it reduces to a $s - t-$path in H, contradicting the assumption that $v_1, \ldots, v_d$ is a separator. □

Proposition 4. *A cut (S, T) in $\overline{H}$ of finite capacity d induces a $s-t-$separator of cardinality d in H.*

Proof. Because $c(S, T) < \infty$ and the fact that edges in $\overline{H}$ have either infinite or unit capacity, we can assume $(v'_1, v''_1), \ldots, (v'_d, v''_d)$ are the edges that lead from S to T. In particular, every $s' - t''$-path contains at least one of them. Making use of proposition 2 we see that the set $\{v_1, ..., v_d\} \subset V$ is a $s - t$ separator in H. □

Proposition 5. *The minimum cuts in $\overline{H}$ correspond to minimum size $s-t-$separators in H via the above connection*

Proof. This follows directly from propositions 4 and 3. □

Having made this reduction, only the problem of determining minimum cuts remains.

2.1.4.4 Cuts and flows

We use the well-known Max-Flow-Min-Cut-duality to determine a minimum cut in $\overline{H}$. First, we define what we mean by a flow in a network N.

A *flow* in N is a function $f : E \to \mathbb{R}_{>0}$ that satisfies the following conditions

(i) for all $(u, v) \in E : f(u, v) \leq c(u, v)$

(ii) for all $v \in V \setminus \{s, t\} : \sum_{(u,v)\in E} f(u, v) = \sum_{(v,u)\in E} f(v, u)$

We note that

$$\sum_{(s,v)\in E} f(s, v) - \sum_{(v,s)\in E} f(v, s) = \sum_{(v,t)\in E} f(v, t) - \sum_{(t,v)\in E} f(t, v)$$

holds and define the left hand side of this equation as the *value* $|f|$ of the flow f. A *maximum flow* on N is a flow with maximum value among all flows on N.

First, we state the famous Max-Flow-Min-Cut-theorem as in [9]. It shows an important duality between cuts and flows in a network and thus allows us to get hold of our set of separators using a slightly modified version of the well-known Ford-Fulkerson algorithm for finding maximum flows.

To state the theorem and the algorithm we need some more notation, taken from [16]. Let $N = (G, s, t, c)$ with $G = (V, E)$ be a simple flow network. Let $p = (s = v_1, v_2, ..., v_{r-1}, t = v_r)$ be an *undirected* $s-t$ path, that is, a path in the underlying undirected graph. A pair (v, w) of successive nodes in p is called a *forward edge*, if $(v, w) \in E$, and *backward edge* if $(w, v) \in E$. Given a flow f, we call an undirected path p an $f-$*augmenting path* if $f(v, w) < c(v, w)$ for every forward edge and $f(v, w) > 0$ for every backward edge in p. Let p be a $f-$augmenting $s - t-$path. Define

$$\delta(v, w) = \begin{cases} c(v, w) - f(v, w) & \text{if } (v, w) \text{ is a forward edge} \\ f(v, w) & \text{if } (v, w) \text{ is a backward edge} \end{cases}$$

and let $\Delta := \min\{\delta(v, w) : (v, w) \text{ on } p\}$.

Define $f_p(v, w) := \Delta$, if (v, w) is a forward edge and set $f_p(v, w) := -\Delta$ if (v, w) is a backward edge. Extend it to a function on E by setting $f_p(v, w)$ to 0 if neither (v, w) nor (w, v) lie on p. The function $f + f_p$, defined by $(f + f_p)(v, w) := f(v, w) + f_p(v, w)$ for every $(v, w) \in E$, clearly yields a flow on N with $|f + f_p| = |f| + \Delta$ (see also [16], lemma 4.3.3). If we say we *augment* f *along* p we mean the transition from f to $f + f_p$.

The following theorem and algorithm show that the notion of augmenting paths can be used for finding maximum flows and minimum cuts.

Theorem 2.2 (Max-Flow-Min-Cut-Theorem). *Let $N = (G, s, t, c)$ be a flow network and f a flow in G. The following conditions are equivalent:*

(i) f is a maximum flow

(ii) N contains no f-augmenting paths

(iii) $|f| = c(S, T)$ for a minimum cut (S, T) of N.

Moreover, given a maximum flow f, the set S of all vertices v that can be reached by an augmenting $s - v$ path in in N induces a min-cut (S, T).

The proof of this theorem can for example be found in [9]. The last remark is not explicitly stated but a direct consequence from the proof.

By this further reduction the only thing left to do is to determine a maximum flow and use breadth-first-search to obtain the set S. To find a maximum flow we use a slight modification of the well-known Ford-Fulkerson algorithm. A generic version of that method is stated in [9] as follows:

FORD-FULKERSON-METHOD(G, s, t, c)
1 initialize flow f to 0
2 **while** there exists an f augmenting path p
3 **do** augment flow f along p
4 **return** f

We modify this method according to our needs. Because we are dealing with integral edge capacities, in every pass of the while- loop flow is augmented by at least 1. As we are only interested in vertex separators of a certain size d or smaller, we can stop after at most d passes of the loop. If then we cannot find any augmenting $s-t-$paths, we use breadth-first-search to determine the vertices that can still be reached from s. By the remark in the Max-Flow-Min-Cut theorem we have thus found a minimum cut and by proposition 5 a minimum sized $s-t-$separator. If after d passes we still find augmenting $s-t-$paths we return NIL. Note that checking if there is an augmenting path p in line 2 can be done by breadth-first-search in time $\mathcal{O}(|V|)$. For details, see [9].

2.1.4.5 The algorithm Separator

Now we are ready to state the SEPARATOR-algorithm:

SEPARATOR$((L_i)_{j-1\leq i\leq j+w}, d)$
1 let $H \leftarrow G\,[(L_i)_{j-1\leq i\leq j+w}]$
2 identify vertices in L_{j-1} with s
3 identify vertices in L_{j+w} with t
4 construct the Even-Tarjan-Reduct$(\overline{H} = (\overline{V}, \overline{E}), s', t'', c)$
5 **for** each edge $(v, w) \in \overline{E}$
6 **do** initialize flow f to 0
7 $\quad d_0 \leftarrow 0$
8 **while** $d_0 \leq d$ and there exists an $f-$augmenting $s-t-$path p
9 **do** augment f along p
10 $\quad d_0 \leftarrow d_0 + 1$
11 **if** $d_0 = d + 1$
12 $\quad$ **then return** NIL
13 determine the endpoints $v'_1, ..., v'_{d_0-1}$ of $f-$augmenting paths
14 **return** $\mathcal{S} := \{v_1, ..., v_{d_0-1}\}$

2.1.4.6 The analysis

Because of the above considerations and the fact that we are mainly dealing with standard methods we only give a brief analysis. The following proposition holds:

Proposition 6. *Let G be a plane graph that allows a layerwise separation of width w and size σ. Let $(L_i)_{1\leq i\leq r}$ be its layer decomposition and let $\beta > 0$. Then* BALANCED SEPARATION$((L_i)_{1\leq i\leq r}, \sqrt{\sigma}, \beta, w)$ *returns a $\sqrt{\sigma}-\beta-$ balanced separation of layers of width w after $\mathcal{O}(|G|w\sqrt{\sigma}\beta)$ steps.*

Proof. The correctness of SEPARATOR follows from proposition 5 together with the Max-Flow-Min-Cut-theorem and the correctness of the Ford-Fulkerson-method. This, together with the proof of theorem 2.1 implies the correctness of the method BALANCED SEPARATION.

The time consumption of a call to SEPARATOR is in $\mathcal{O}(n'\sqrt{\sigma}\beta)$, where n' is the number of vertices of the subgraph it is called on. BALANCED SEPARATION

calls SEPARATOR at most w times on every node, yielding the claimed result.

□

2.2 Decomposing r-outerplanar graphs

In this section we describe the decomposition of $r-$outerplanar graphs into trivial subgraphs. Apart from the differing mode of presentation, it is essentially taken from [5].

In part 2.2.1 we construct a forest that captures the structure of a given $r-$outerplane graph G. In 2.2.2 we make use of this forest-structure by recursively defining a sequence of subgraphs – named *slices* – of G. They have the property that neighboring slices share only a few common *boundary nodes* and that their union yields the entire graph G. Subsequently, in chapter 2.2.3, we interpret this recursive definition in terms of a decomposition of G via separators, thereby providing a common viewpoint for this and last chapter's considerations and a basis for the theory developed in section 2.3.

To get a feel for this decomposition, consider a very special example of a $r-$outerplanar graph: imagine a graph consisting of r nested circles. The idea is to cut this graph into slices like a pie: from the center to the exterior face such that two neighboring slices share only one node from each of the r layers.

However, the shape of general $r-$outerplanar graphs may be much more complicated. To capture this structure, the so-called *structure trees* are used

2.2.1 Structure trees

Let G be an embedded r-outerplanar graph and $(L_i)_{1\leq i\leq r}$ its layer decomposition. Observe that $G[L_i]$ is outerplanar for $i = 1, \ldots, r$. Each of the connected components of $G[L_i]$, the *level i components*, is contained in one face of the graph $G[L_{i-1}]$ for $i = 2, \ldots, r$.

Insert dummy edges to connect $G[L_1]$ and all level $i > 1$ components contained in a common face. Furthermore, introduce a dummy edge as a copy of every cut-edge (i.e. an edge whose removal disconnects the component) of such a component. This construction leaves us with a graph in which every level $i-1$ face contains at most one, $2-$edge–connected but not necessarily simple, level i component. This modification simplifies the construction of the structure trees. During the calculation, dummy edges will simply be ignored. Finally, keep a *fixed triangulation* of the region between every

level i component and its enclosing face in mind. Such a triangulation can be obtained in linear time (see [5]).

We start by constructing a labeled and rooted tree $\mathcal{T}_H$ for every such 2−edge–connected outerplanar component H. After that, we use the triangulation to relate the vertices of $\mathcal{T}_H$ with the tree of the enclosing face's component, capturing the global structure of G's embedding.

To be able to linguistically separate the different structures involved, we follow the example in [5] and call the vertices of the tree $\mathcal{T}_H$ *vertices*, whereas graph H's vertices will be called *nodes*. We use variables u, v, w for vertices and x, y, z for nodes.

2.2.1.1 The construction of $\mathcal{T}_H$

For this section, let H be a 2−edge–connected, outerplane graph.

Call edges adjacent to H's exterior face *exterior edges*. An edge that is not exterior is called an *interior edge* Firstly, introduce a vertex for every exterior edge and a vertex for every interior face of H. Call the first ones *edge vertices* and the latter ones *face vertices*. Connect every edge vertex to the vertex of the face it is adjacent to, and connect vertices of faces sharing a common edge. Observe the close relationship between the *weak dual* – i.e. the graph obtained from the dual graph by removing the exterior face's vertex – of a plane graph and the graph just constructed. A slight modification of a result on the weak dual of outerplanar graphs stated in [16, p. 254] shows that our graph under construction is actually a forest.

If it consists of more than one tree one easily sees that, due to the connectivity of H, must be faces meeting in cut-nodes of H (i.e. nodes whose removal disconnects H). Successively introduce edges between such face vertices belonging to different trees until the forest becomes connected. Call the tree thus obtained $\mathcal{T}_H$. Denote the forest of all such trees of graph G by $\mathcal{T}_G$.

For a small example, see figure 2.1. The black vertices, together with the dashed edges, depict the structure tree of the graph given by the white nodes.

Next, we will label $\mathcal{T}_H$'s vertices recursively and define a natural ordering on the children of every face vertex. The ordering and labeling will be with respect to a root vertex that has yet to be chosen. Details of the root choice follow subsequently.

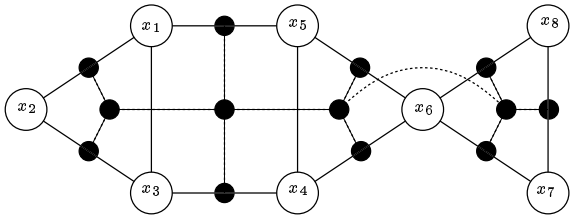

Figure 2.1: Construction of the structure tree

2.2.1.2 The ordering

Pick a face vertex v to be the root of $\mathcal{T}_H$. Choose an arbitrary child w of v to be its first descendant. Number the children of v in order of appearence in a counterclockwise walk around v's face starting with w. In this walk, face vertices correspond to the edge – or node, in the case of cut-nodes – they share with v's face. Recursively repeat this procedure on every child u of v that is a face vertex, with the only modification being that the first child of u is now fixed to be the next one to the edge or node corresponding to u – again in counterclockwise order.

We have thus defined a linear ordering $\leq$ on the children of every face vertex. This ordering naturally extends to the leaves of the tree. When we talk about the *succeeding sibling* or the *succeeding leaf* we mean the next vertex or leaf with respect to $\leq$ or its natural extension.

2.2.1.3 The labeling

Now we assign directed edges as labels to $\mathcal{T}_H$'s vertices. Label each edge vertex by the edge it represents, directed in the direction of the counterclockwise walk around its the face of its ancestor. Label a face vertex v with (x, y), with x being the first node in the label of v's first child and y as the second node in the label of v's last child. From now on we sometimes identify a tree vertex v by its label (x, y) and write $v = (x, y)$.

The result of the labeling procedure of the graph in figure 2.1 is depicted in 2.2. The vertex of the leftmost face was chosen as root, its first child is the edge (x_1, x_2).

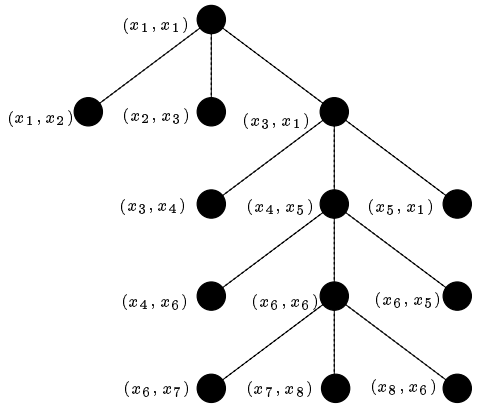

Figure 2.2: The labeled structure tree

2.2.1.4 Root choice

Up to now we have defined a tree for every layer component with vertices labeled and ordered recursively with respect to an arbitrarily chosen root. In this part we describe how this root is chosen in order to relate the trees of nested layer components according to our needs. The choice might not seem compelling at first sight. When defining slices, however, it will become clear that it was made in a way so that the slice boundaries of the root vertex of a component are natural extensions to those of the enclosing face's vertex.

Choose an arbitrary face vertex v to be the root of $G[L_1]$'s tree. Let H be a layer $i > 1$ component and suppose that the structure tree for all layer $i-1$ components are already constructed, rooted and labeled. Let $v = (x, y)$ the face vertex of the level $i-1$ face that encloses H. If $x \neq y$, let z be the node in H that is adjacent to both x and y in the chosen triangulation. If $x = y$, let z be an arbitrary example of the nodes with that property. Choose the root vertex u of $\mathcal{T}_H$ as the vertex of the face which node z lies on and choose its first child as the vertex of the edge in H leaving z in counterclockwise order. The labeling conventions assure that $u = (z, z)$ holds - and this is precisely what we want because of the reasons sketched above and worked out in detail in the next chapter.

2.2.1.5 Some properties of structure trees

We will point out some interesting attributes of the trees just constructed.

Remark 2.2.1. Let v be a face vertex of $\mathcal{T}_H$. If it is the root vertex, its label is of the form (x, x) for some node x. If $v = (x, y)$ with $x \neq y$ then (x, y) is the edge shared by v's and its parent's face. (x, y) is directed in the direction of a *clockwise* walk around v's face and a *counterclockwise* walk around its ancestor's face.

Remark 2.2.2. Let v be a face vertex with children $v_1, ..., v_t$. The *children's* labels together with v's inverted label represent a counterclockwise walk around v's face. The labels of the *leaves* of $\mathcal{T}_H$ represent a directed counterclockwise walk around the exterior face of H.

The last two comments give an idea of how the structure of the trees can be used to define the slices mentioned earlier. To define a boundary running from a level i component H to its enclosing face in level $i-1$, we have to relate the *leaves* of $\mathcal{T}_H$ to the *children* of the enclosing face's vertex. To do this we can use $\leq$ to process both the leaves and the children in order of appearance.

2.2.2 The slices

In this section we will recursively define the slices the $r-$outerplane graph G is composed of. We do this by defining left and right boundaries ∂_{l} and ∂_{r} for every tree vertex v. These boundaries run from the level of v to the exterior face, containing exactly one node from every level subgraph. Using the ordering relation $\leq$ the slice is defined as the node set that lies between the left and right boundary nodes.

What is meant by *left* and *right* for a vertex $v = (x, y)$ will be defined in precision later on. Intuition coincides with the definition when the viewer is imagined in the same plane as the graph, looking in the direction of the edge (x, y) from outside the face the edge lies on.

In the following we use the same fixed triangulation between the level subgraphs and their enclosing faces as before.

2.2.2.1 Boundary definition

It would be sufficient to give mappings from level i to level $i-1$ *nodes* for $i = 2, ..., r$ in order to define the boundaries of slices and use the ordering to define the slice as the subgraph included within two such boundaries. The possible presence of cut-nodes makes this approach impossible for they allow no such unique assignment.

The solution is to use the trees and the information their labels carry to give a mapping from the set of level i *vertices* to level $i-1$ vertices for every $i = 2, \ldots, r$. Having defined such a mapping, we use it to give a definition of boundaries mentioned above.

For a level i component H_i and its enclosing level $i-1$ component H_{i-1}, these mappings of nodes will be of the form $\Delta_{\mathrm{l}} : V(\mathcal{T}_{H_i}) \to V(\mathcal{T}_{H_{i-1}}) \cup \{\star\}$ and $\Delta_{\mathrm{r}} : V(\mathcal{T}_{H_i}) \to V(\mathcal{T}_{H_{i-1}}) \cup \{\star\}$, respectively.

For their definition, let $v_1, \ldots, v_t$ be the leaves of $\mathcal{T}_{H_i}$ and $w_1, \ldots, w_q \in V(\mathcal{T}_{H_{i-1}})$ the children of the enclosing face's vertex w. Pick two successive leaves $v_j = (x_j, x_{j+1})$ and $v_{j+1} = (x_{j+1}, x_{j+2})$. There is a node $z(j) \in V(H_{i-1})$ and an edge $(x_{j+1}, z(j))$ in the chosen triangulation such that the edges $(x_{j+1}, x_j), (x_{j+1}, z(j))$ and (x_{j+1}, x_{j+2}) appear in counterclockwise order around node x_{j+1}. Call such a node $z(j)$ *dividing point* for v_j and v_{j+1}.

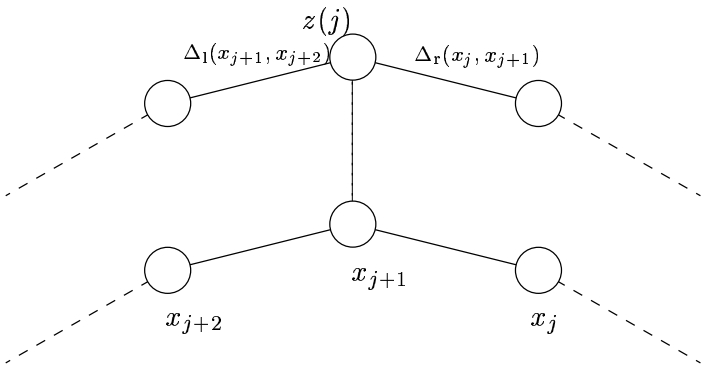

Figure 2.3: definition of Δ_{l} and Δ_{r}

Observe that the dividing points can always be chosen such that $z(p)$ appears before $z(p+1)$ in a counterclockwise walk around the enclosing face for $p = 1, \ldots, t-1$.

Inductively set $w(j) := (z(j), b)$ as the vertex of $w_1, \ldots, w_q$ with $w(j) \geq w(j-1)$ which represents the edge leaving $z(j)$, and $w(j) := \star$ if there is no such edge. Let $\overline{w}(j) \in \{w_1, \ldots, w_q\}$ be the predecessor of $w(j)$ with respect to $\leq$, and $\overline{w}(j) := \star$, if there is none. Note that either $w(j) \neq \star$ or $\overline{w}(j) \neq \star$ holds. Furthermore, set $w(0) = w_1$ and $w(t) = w_q$.

Definition 2.5. We define $\Delta_{\mathrm{l}}(v_j) = w(j-1)$ and $\Delta_{\mathrm{r}}(v_j) = \overline{w}(j)$ for every $j = 1, ..., t$. If u is a face vertex, u_1 is its first and u_s its last child, set $\Delta_{\mathrm{l}}(u) = \Delta_{\mathrm{l}}(u_1)$ and $\Delta_{\mathrm{r}}(u) = \Delta_{\mathrm{r}}(u_s)$

An example for the definition of Δ_{l} and Δ_{r} is depicted in figure 2.3.

The definitions of Δ_{l} on v_1 and Δ_{r} on v_t mark the extreme left and right boundaries of any of the face's children, in accordance with the choice of the root in $\mathcal{T}_{H_i}$. Now we can recursively define the left and right boundary sets:

Definition 2.6. For $v = (x, y)$ we define $\partial_{\mathrm{l}}(v)$ as:

(LB1)	$\{x\}$	if v is a level 1 vertex
(LB2)	$\{x\} \cup \partial_{\mathrm{l}} \Delta_{\mathrm{l}}(v)$	if v is a level i vertex with $i > 1$ and $\Delta_{\mathrm{l}}(v) \neq \star$
(LB2')	$\{x\} \cup \partial_{\mathrm{r}} \Delta_{\mathrm{r}}(v)$	if v is a level i vertex with $i > 1$ and $\Delta_{\mathrm{l}}(v) = \star$

The set $\partial_{\mathrm{r}}(v)$ is defined by (RB1), (RB2) and (RB2') similarly: simply substitute x by y, l by r and vice versa.

The definition of (LB1) and (LB2) are quite intuitive, the definition of (LB2') deals with the border case and will become clear later.

Before giving a definition of slices, we prove some important properties about boundaries.

Proposition 7. *The following statements are true:*

(i) $\partial_{\mathrm{l}}(v) = \partial_{\mathrm{l}}(v_1)$ *and* $\partial_{\mathrm{r}}(v) = \partial_{\mathrm{r}}(v_t)$ *if* v *is a face vertex and* v_1, v_t *are its first and last child, respectively.*

(ii) $\partial_{\mathrm{r}}(v_j) = \partial_{\mathrm{l}}(v_{j+1})$ *for two successive vertices* v_j *and* v_{j+1}.

(iii) $|\partial_{\mathrm{l}}(v)| = i$ *and* $|\partial_{\mathrm{r}}(v)| = i$ *for every level* i *vertex* v.

(iv) for a face vertex u *and an enclosed component with root* $v = (x, x)$ $\partial_{\mathrm{l}}(u) \cup \{x\} = \partial_{\mathrm{l}}(v)$ *and* $\partial_{\mathrm{r}}(u) \cup \{x\} = \partial_{\mathrm{r}}(v)$ *hold.*

Proof. Assertion *(i)* follows directly from face vertex labeling conventions and the definition of Δ_{l} and Δ_{r} on face vertices.

Claim *(ii)*: Because of *(i)* we can restrict ourselves to successive leaves. We prove *(ii)* by induction on the level index. For neighboring level 1 leaves $v_j = (x_j, x_{j+1})$ and $v_{j+1} = (x_{j+1}, x_{j+2})$ a simple application of (LB1) and (RB1) does the trick. For consecutive level $i > 1$ leaves v_j and v_{j+1} labeled as above and $\Delta_{\mathrm{l}}(v_{j+1}) \neq \star \neq \Delta_{\mathrm{r}}(v_j)$ we ascertain by definition 2.5 that $\Delta_{\mathrm{l}}(v_{j+1})$ and $\Delta_{\mathrm{r}}(v_j)$ are adjacent level $i-1$ vertices. Using induction hypothesis and (LB2) and (RB2) proves the claim. In the case that, for example, $\Delta_{\mathrm{r}}(v_j) = \star$ holds, let $w_1, \ldots, w_t$ be the children of the enclosing face's vertex. The definitions

of Δ_r and Δ_l show that $\Delta_l(v_{j+1}) = w_1$ in this case. Then also $\Delta_l(v_j) = w_1$ holds and with (RB2') we get $\partial_r(v_j) = \{x_{j+1}\} \cup \partial_l \Delta_l(v_j) = \{x_{j+1}\} \cup \partial_l \Delta_l(v_{j+1}) = \partial_l(x_{j+1}, x_{j+2})$. The same arguments hold for Δ_l.

For assertion *(iii)* recall that Δ_l and Δ_r are mappings from level i to level $i-1$ vertices. The sets $\partial_l(v)$ and $\partial_r(v)$ thus contain exactly one node from each of the i level subgraphs.

For claim *(iv)* let $v_1, ..., v_t$ be the leaves of the enclosed component with root $v = (x, x)$ and $u_1, ..., u_s$ the children of u. Note that the labels of v_1 and v_t are of the form (x, x_2) and (x_t, x), respectively. From the definition of Δ_l it follows that $\Delta_l(v_1) = u_1$ and so

$$\partial_l(v) \overset{(i)}{=} \partial_l(v_1) \overset{\text{LB2}}{=} \{x\} \cup \partial_l \Delta_l(v_1) = \{x\} \cup \partial_l(u_1) \overset{(i)}{=} \{x\} \cup \partial_l(u)$$

The same argumentation holds for Δ_r.

□

2.2.2.2 Slice definition

In this section we recursively define a node set $\mathrm{sl}(v)$, called the *slice*, for every tree vertex $v \in V(\mathcal{T}_G)$. In [5], the slices are defined as *subgraphs* of G and are – with one exception – equal to the subgraphs of G induced by their node set $\mathrm{sl}(v)$. Yet, from the theoretical viewpoint to be developed in section 2.3, it will turn out to be useful not to be restricted to those induced subgraphs but to admit more general *slice subgraphs* $G_v \subseteq G[\mathrm{sl}(v)]$. Their carrying node sets, however, remain invariant. Consequently, we define slices as the corresponding invariant *node sets* and leave the precise – and, in our approach, problem dependent – definition of the slice subgraphs to next chapter's considerations.

In addition to the theoretical advantages mentioned earlier, this approach allows for a more compact description of slices.

Let $v = (x, y)$ be a level i vertex. For $i > 1$ define $\mathcal{N}(v)$ as the set of all level $i-1$ vertices w with $\Delta_l(v) \leq w \leq \Delta_l(v)$ and $\mathcal{N}(v) = \emptyset$ if $\Delta_l(v) = \star$ or $\Delta_r(v) = \star$

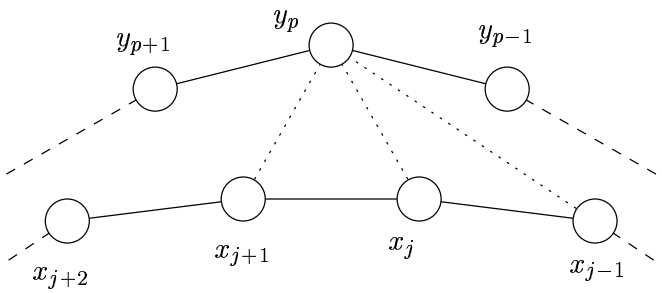

Figure 2.4: a triangulation leading to a degenerated slice

Definition 2.7. We define *the slice* $\mathrm{sl}(v)$ of a vertex v as:

(S1)	$\{x, y\}$	if v is a level 1 leaf
(S2)	$\{x, y\} \cup \bigcup_{u \in \mathcal{N}(v)} \mathrm{sl}(u)$	if v is a level $i > 1$ leaf and $\mathcal{N}(v) \neq \emptyset$
(S2')	$\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v)$	if v is a level $i > 1$ leaf and $\mathcal{N}(v) = \emptyset$
(S3)	$\bigcup_{j=1}^{t} \mathrm{sl}(v_j)$	if v is a level i face vertex without enclosed level $i+1$ component and children $v_1, ..., v_t$
(S4)	$\mathrm{sl}(u)$	if v is a level i face vertex and u is the root vertex of the enclosed level $i+1$ component

Before we prove important properties of the correlation of slices and their boundaries, we give a explanation for the definition parts (LB2') and (S2'). Consider the example depicted in figure 2.4.

The nodes x_{j-1}, x_j, x_{j+1} and x_{j+2} depict elements of the component enclosed by the face which y_{p-1}, y_p and y_{p+1} belong to. Let y_p be the dividing point for both $(x_{j-1}, x_j), (x_j, x_{j+1})$ and $(x_j, x_{j+1}), (x_{j+1}, x_{j+2})$ This choice of dividing points leads to $\Delta_{\mathrm{l}}(x_j, x_{j+1}) = (y_p, y_{j+p})$ and $\Delta_{\mathrm{r}}(x_j, x_{j+1}) = (y_{p-1}, y_p)$ which then leads to $\mathcal{N}(x_j, x_{j+1}) = \emptyset$. We call a slice with that property *degenerated*. For practical reasons one wants degenerated slices to have the same boundary length as non-degenerated slices. Therefore the slice is defined as consisting *only* of the boundary nodes defined by ∂_{l} and ∂_{r}. Part *(i)* of proposition 8 claims that in the degenerated case the left and right boundaries coincide

on levels $1, \ldots, i-1$. Similar considerations hold for the case $\Delta_{\mathrm{l}}(v) = \star$ or $\Delta_{\mathrm{r}}(v) = \star$. Thus we defined (S2') in the above way only for esthetic reasons.

With the slice and boundary definition at hand prove some important properties in proposition 8. Part *(iii)* should be emphasized, for it provides the connection between the geometric intuition and the formal definitions.

For ease of formulation we introduce some more notation: for a tree vertex $v = (x, y)$, the embeddings of $G[\partial_{\mathrm{l}}(v)]$ and $G[\partial_{\mathrm{r}}(v)]$, together with the edge (x, y) and the edges of the exterior face running from $\partial_{\mathrm{l}}(v)$ to $\partial_{\mathrm{r}}(v)$ form a bounded region in the plane. To see this, note that Δ_{l} and Δ_{r} are defined along the the chosen triangulation. We call the topological closure of that region the *sector* of v. Furthermore, we introduce the *interior* of a slice $\mathrm{sl}(v)$ as $\mathrm{int}(\mathrm{sl}(v)) = \mathrm{sl}(v) \setminus (\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v))$.

Proposition 8. *The following statements hold:*

(i) *if v is a level i vertex with degenerated slice, then $\partial_{\mathrm{l}}(v)$ and $\partial_{\mathrm{r}}(v)$ coincide on levels $1, \ldots, i-1$.*

(ii) *if v is a face vertex, then there is a child u of v with $\partial_{\mathrm{l}}(u) \cap \partial_{\mathrm{r}}(u) = \emptyset$.*

(iii) *the set of nodes in the sector of v equals $\mathrm{sl}(v)$*

(iv) *the slice of a vertex v consists of the slices of all its descendants and their enclosed components.*

(v) *$\mathrm{sl}(v) = V(G)$ if v is the root of the level 1 component.*

(vi) *the triple $(V \setminus \mathrm{sl}(v), \partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v), \mathrm{int}\,\mathrm{sl}(v))$ forms a separation for every tree vertex v.*

Proof. The proof for *(i)* in the case of degeneration because of $\Delta_{\mathrm{r}}(v) = \star$ or $\Delta_{\mathrm{l}}(v) = \star$ follows from parts (RB2') and (LB2') of definition 2.6. If otherwise $\mathcal{N}(v) = \emptyset$ it can be only because $\Delta_{\mathrm{l}}(v) = w_i$ and $\Delta_{\mathrm{r}}(v) = w_{i+1}$ for two successive vertices w_i and w_{i+1}. Then part *(ii)* of proposition 7 applied to w_i and w_{i+1} finishes the proof.

For *(ii)* assume $\partial_{\mathrm{l}}(u) \cap \partial_{\mathrm{r}}(u) \neq \emptyset$ for every child u. By part *(i)*, one sees that once $\partial_{\mathrm{l}}(u)$ and $\partial_{\mathrm{r}}(u)$ intersect, they coincide on all of the surrounding layers, and hence on layer 1. By successive application of part *(ii)* of proposition 7

on all neighboring children and we see that the first childs left boundary and the last childs right boundary, and hence v's left and right boundaries intersect on layer 1, showing that the exterior face consists of a single edge, contradicting the looplessness of G.

We show *(iii)* by induction on the depth of recursion in definition 2.7. For its finiteness, we refer to [5]. The assertion is clearly true if v is a level 1 leaf. If v is a level $i > 1$ leaf with $\mathcal{N}(v) = \emptyset$ it follows from part *(i)* of proposition 8. Assume now the assertion holds for every recursive occurrence of sl in the definition of $\mathrm{sl}(v)$.
Let $v = (x, y)$ be the vertex of a face with no enclosed component and let $v_1, \ldots, v_t$ be its children. By item *(ii)* of proposition 7 the boundaries of two succeeding vertices v_j and v_{j+1} coincide. Using this fact with the hypothesis, the union $\bigcup_{j=1}^{t} \mathrm{sl}(v_j)$ is the set of vertices contained within the region cut out by $\partial_{\mathrm{l}}(v_1) \cup \partial_{\mathrm{r}}(v_t)$, the edges corresponding to $v_1, \ldots, v_t$ and the edges of the exterior face contained in the sector. The face of v contains no component, thus adding the edge (x, y) – if $x \neq y$ – closes the path formed by $v_1, \ldots, v_t$ and yields a new boundary given by the embeddings of $\partial_{\mathrm{l}}(v_1), \partial_{\mathrm{r}}(v_t)$ and (x, y). The left boundariy of v_1 and the right boundary of v_t correspond with the respective ones of v by item *(i)* of proposition 7, whence we obtain what we claimed.
If v is the vertex of a face with enclosed component H, we simply applying induction hypothesis to $u = \mathrm{root}\,\mathcal{T}_H$ and use item *(iv)* of proposition 7.
Finally, if v is a level i leaf we obtain $\bigcup_{u \in \mathcal{N}(v)} \mathrm{sl}(u)$ as the set of nodes contained in the region bounded by the edges corresponding to vertices in $\mathcal{N}(v)$ and $\partial_{\mathrm{l}} \Delta_{\mathrm{l}}(v)$ and $\partial_{\mathrm{r}} \Delta_{\mathrm{r}}(v)$ arguing as in the first case. By definition, $\partial_{\mathrm{l}}(v)$ and $\partial_{\mathrm{r}}(v)$ are prolongations $\partial_{\mathrm{l}} \Delta_{\mathrm{l}}(v)$ and $\partial_{\mathrm{r}} \Delta_{\mathrm{r}}(v)$. This finishes our proof.

The proofs for *(iv)* and *(v)* can be found in [5], deduced by *(iii)* or proven in the same way as *(iii)*.

For *(vi)* observe that due to the planarity of G and the definition of ∂_{l} and ∂_{r} along the triangulation, no edge crosses a slice's boundary. This fact, together with *(iii)*, implies the assertion. □

2.2.3 Slice subgraphs and separators

In this chapter we show how the recursive definition of slices can be interpreted as a consequent decomposition of an $r-$outerplane graph G using separators. While definition 2.7 helps with the algorithmic details, the separator-based viewpoint provides a sound theoretical basis for a problem-independent analysis of both Alber's and Baker's approaches.

While decomposing the graph into trivial subgraphs we define the problem dependent *slice subgraphs* $G_v \subseteq G[\mathrm{sl}(v)]$ for every vertex $v \in V(\mathcal{T}_G)$.

Assume that there is a problem-dependent rule that, for every instance $G = (V, E)$ and every separation (V_1, S, V_2) of G, determines the corresponding fragments G_1 and G_2 (see section 1.2.3).
In practise, only two cases will be of importance. The first is the case that both fragments contain all of the common node set's edges, that is, $G_i = G[V_i \cup S]$ for $i = 1, 2$. The second is the case that every edge is contained in exactly one of the fragments, i.e., $G_1 = G[V_1 \cup S]$ and $G_2 = G[V_2 \cup S] \setminus E(G[S])$.

The root: If v is the root vertex of the level 1 component, define $G_v = G$. Note that this definition is sound, because from part *(vi)* of proposition 8 it follows that $\mathrm{sl}(v) = V(G)$.

Face vertices: Suppose now that $v = (x, y)$ is a level i face vertex and we are given G_v. If $x \neq y$, use separator $\{x, y\}$ to obtain subgraphs $G_v \setminus (x, y)$ and $(\{x, y\}, \{(x, y)\})$. In the case of an enclosed level $i + 1-$component, recursively step down to the innermost component H with $u = \mathrm{root}(\mathcal{T}_H)$. Let $u_1, \ldots u_t$ be the children of u.

The straightforward approach would now be to successively separate the children's slices from the given graph using their boundaries as separators (see part *(vi)* of prop. 8).
To obtain a better running time, however, it will turn out to be useful to split the graph in such a way that a children's slice can be separated by *either* ∂_{l} *or* ∂_{r} instead of both. To obtain this we need to "cut open" the circle formed by H and the enclosing components. Observe that this might fail if we cut out a degenerated slice, for the remaining fragment's node set remains unchanged.

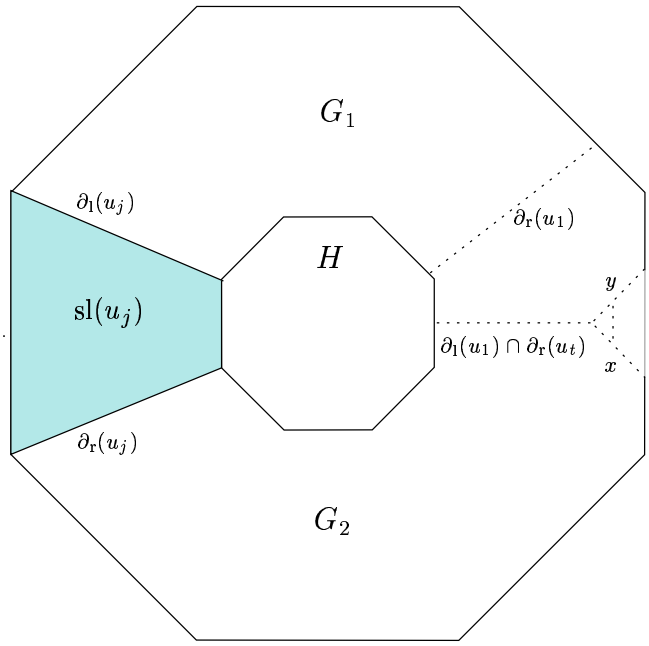

Figure 2.5: decomposition of the slice subgraph of a face vertex

Consequently, pick j with $\partial_{\mathrm{l}}(u_j) \cap \partial_{\mathrm{r}}(u_j) = \emptyset$. Note that by part *(ii)* of proposition 8 such a j always exists. Define G_{u_j} as the fragment of $G_v \setminus (x, y)$ with node set $\mathrm{sl}(u_j)$ using separator $\partial_{\mathrm{l}}(u_j) \cup \partial_{\mathrm{r}}(u_j)$. Let G^c be the remaining fragment. By the choice of j one easily sees that the node set $\partial_{\mathrm{l}}(u_1) \cap \partial_{\mathrm{r}}(u_t)$ is a separator, separating G^c into fragments G_1 and G_2 with $\mathrm{sl}(u_k) \subseteq G_1$ for $k = 1, \ldots, j-1$ and $\mathrm{sl}(u_k) \subseteq G_2$ for $k = j+1, \ldots, t$.
Use the set $\partial_{\mathrm{r}}(u_1)$ as a separator of G_1^c. Take the fragment with node set $\mathrm{sl}(u_1)$ as definition of the slice subgraph G_{u_1}. Repeat this process with the respective remaining fragments for $k = 2, \ldots, j-1$. Apply the same procedure for $k = j+1, \ldots, t$ on G_2. We thus obtain the slice subgraphs for every of u's children. The first steps of this procedure are depicted in 2.5.

Leaves: Suppose $v = (x, y)$ is a level $i > 1$ leaf and $\mathcal{N}(v) \neq \emptyset$ and we are given its slice subgraph G_v.

Firstly, let $\mathcal{N}(v) = \{w_1, \ldots, w_t\} \neq \emptyset$ with $w_i = (z_i, z_{i+1})$. Interpreting rule (S2) as decomposition via small separators is more sophisticated. Fol-

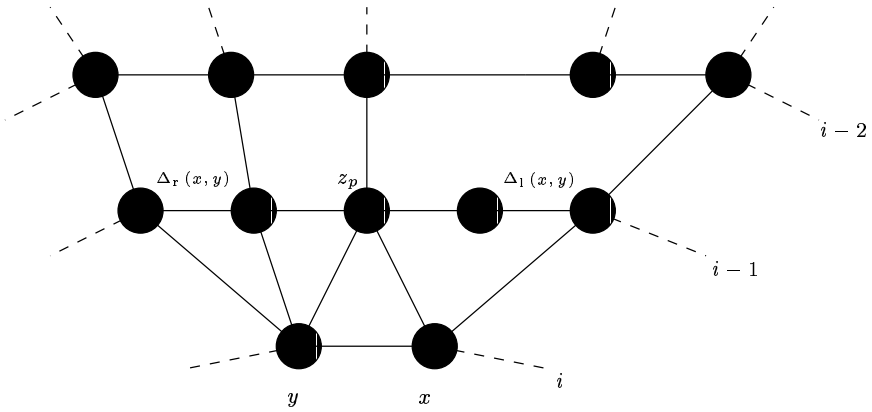

Figure 2.6: a subgraph G_v of a level $i > 1$ leaf v with $\mathcal{N}(v) \neq \emptyset$

lowing the proof of *(vi)* in proposition 8, the natural approach would be to use the node set $S = \{z_i : 1 \leq i \leq t+1\}$ as a separator and decompose the thus obtained fragment corresponding to $\bigcup_{w \in \mathcal{N}(v)} \mathrm{sl}(w)$ in the above described way. Observe however that this approach is doomed to fail, for there is no size bound on S. (For the necessity of such bounds, see chapter 2.5.2)
The solution comes from the planarity assumption. The essential steps of the process described below are depicted in figures 2.6 and 2.7. First, separate G_v into $G_v \setminus (x,y)$ and (x,y) using the separator $\{x,y\}$. This corresponds to the dashed line labeled by 1 in figure 2.7. One easily sees that by planarity there can be at most one level $i-1$ node z that is connected to both x and y. Let p be such that $z = z_p$ holds. If there is no such j, let p be such that $w_p = \Delta_{\mathrm{r}}(v)$.
Starting with $j = t$ and $H^c_{w_{t+1}} = G_v \setminus (x,y)$, we reiterate the following process for all $w_j \in \mathcal{N}(v)$ with $j > p$ in descending order: use the set $\partial_{\mathrm{l}}(w_j) \cup \{y\}$ as a separator of $H^c_{w_{j+1}}$ to obtain fragments H_{w_j} and $H^c_{w_j}$.The first step corresponds to the separation labeled by 2 in figure 2.7.
In an intermediate step we use the separator $\partial_{\mathrm{l}}(w_p)$ to obtain the fragments H_{w_p} and $H^c_{w_p}$.
Repeat this process with $w_j \in \mathcal{N}(v)$ for every $j = p-1, \ldots 2$ in decending order, now using the separator $\partial_{\mathrm{l}}(v) \cup \{x\}$ of the respective remaining graph fragments .
Observe that in this way we obtain graph fragments H_w for every $w \in \mathcal{N}(v)$. To finally obtain the slice subgraphs G_w from the fragments H_w we do the

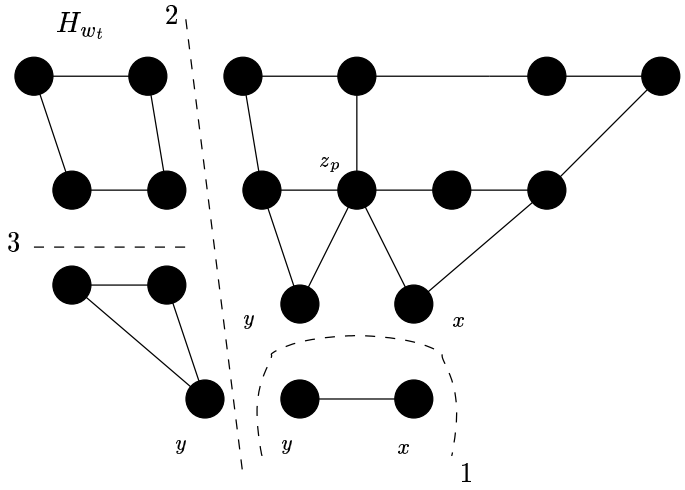

Figure 2.7: decomposing G_v

following: for every $w = (a, b) \in \mathcal{N}(v)$ we use the separator $\{a, b\}$ to obtain the slice subgraphs G_w and fragments with node set $\{x, a, b\}$ or $\{y, a, b\}$, respectively. This step corresponds to the dashed line labeled by 3 in figure 2.7.

Finally, let v be a level $i > 1$ leaf with $\mathcal{N}(v) = \emptyset$ and let G_v be the corresponding graph fragment. By point *(i)* of proposition 8 we can assume $\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v) = \{x, y, x_{i-1}, \ldots, x_1\}$ where x_j is a level j node for $1 \leq j \leq i-1$. Starting with separator x_{i-1} we successively decompose G_v into fragments with node sets $\{x, y, x_{i-1}\}, \{x_{i-1}, x_{i-2}\}, \ldots, \{x_2, x_1\}$.

Remark 2.2.3. The above process describes the successive deconstruction of an $r-$outerplanar graph G into subgraphs with either 2 or 3 nodes, using separators given by slice boundaries.

Now we are done: firstly we a given planar graph into fragments of bounded outerplanarity, then we showed how to decompose each such fragment into trivial subgraphs. In the next chapter we show how this deconstruction can be utilized.

2.3 Problems and subproblems

In the this section we show how to utilize the separator-based deconstruction. Suppose we are dealing with an *optimization problem* Π on graphs. Our aim is to build optimal solutions for an instance G from optimal solutions of Π on subgraphs of G. This can be done by using dynamic programming techniques.

The applicability of dynamic programming techniques to optimization problems strongly depends on the relationship between solutions of the problem Π and its subproblems. The required property is the so-called *optimal substructure property*, the characteristic that an optimal solution of the problem Π contains within it optimal solutions to the subproblems (see e.g. [9]).

In our case, the subproblems will be solutions of the original problem Π on subgraphs of the original instance G. Thus, for applying dynamic programming techniques it is necessary to investigate the relationship between solutions of Π on G and those on G's fragments obtained by separation.

In 2.3.1 we give an example of the difficulties that occur when approaching this problem the naive way. Paragraph 2.3.2 provides a workaround to this problem, together with a formalization of the concept of *optimal substructure* in the context of graph optimization problems. Finally, 2.3.3 is concerned with deducing consequences that will help us to design and analyze algorithms for attacking those problems.

Note by the way that the considerations in this section are in no way restricted to *planar* graphs.

2.3.1 Arising difficulties

Let Π be an optimization problem on graphs. Suppose G is an instance and G_1, G_2 are fragments of G obtained by separation. A simple but nevertheless important observation is the following.

- the union of solutions of Π on G_1 and G_2 does not necessarily form a solution of Π on G.
- cuts of solutions of Π on G with G_i for $i = 1, 2$ do not necessarily induce solutions of Π on G_i.

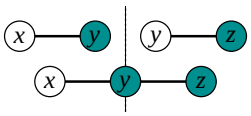

Figure 2.8: unions of INDEPENDENT SETS cease to be independent

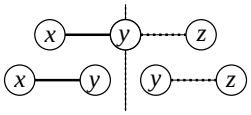

Figure 2.9: cuts of EDGE DOMINATING SETS cease to be dominating

For point *(i)* consider the graph sketched in figure 2.8. The shaded vertices form MAXIMUM INDEPENDENT SETS on the graph fragments, but their union does not yield an independent set.

Item *(ii)* is best illustrated by EDGE DOMINATING SET on the graphs of figure 2.9. The edges sketched with filled lines belong to the dominating set, the dashed edges do not. The intersection of the dominating set with the fragment on the right does not conserve the required property.

From the first example we learn that not any two optimal subsolutions can be combined to give a feasible solution. The second example shows that our problem does apparently not have the optimal substructure property, for the "subsolutions" are not even solutions of the problem on the smaller instances.

Another important observation is that for problems such as INDEPENDENT SET or EDGE DOMINATING SET, cuts of solutions with fragments only locally cease to look like solutions – "close" to the separating nodes. The inconsistency of unions of solutions also seems to be a local phenomenon with the regarded problems.

We handle these problems by characterizing the "boundary behaviour" of the subsolutions, with the boundary being the common nodes of the fragments – the separator. We will furthermore define a notion of "compatibility" that helps us decide whether two subsolutions fit together on the common nodes.

2.3.2 Dissectable problems

2.3.2.1 Constrained problems

In this section we deal exclusively with *graph* optimization problems $\Pi = (\mathcal{I}_\Pi, \mathcal{S}_\Pi, \nu_\Pi)$. By this we mean optimization problems in the sense of definition 1.2, where every instance $I \in \mathcal{I}_\Pi$ is a finite graph and where every feasible solution $S \in \mathcal{S}_\Pi(I)$ is a subgraph of I.
By interpreting a node set as a graph with empty edge set and by regarding an edge set as an edge-induced subgraph, the class of graph optimization problems contains all of the $\mathcal{NP}-$complete optimization problems mentioned in this text.

Our first aim, given a graph optimization problem Π, is to construct a derivative Π_c that has the desired optimal substructure property.

A *constrained version* Π_c of a graph optimization problem Π is an optimization problem whose instances are of the form $(G, W_1, \ldots, W_d)$ for a constant $d \in \mathbb{N}$ with $G = (V, E) \in \mathcal{I}_\Pi$ and $W_i \subseteq V$ for $i = 1, ..., d$. We call $\{W_1, \ldots, W_d\}$ the set of *constraints* on G. As for the unconstrained version, we require $S \subseteq G$ for every $S \in \mathcal{S}_{\Pi_c}(G, W_1, ..., W_d)$. Furthermore we require that $\mathcal{S}_{\Pi_c}(G, \emptyset, ..., \emptyset) = \mathcal{S}_\Pi(G)$ holds and that ν_{Π_c} and ν_Π coincide on that space.

With the connection of ν_{Π_c} and ν_Π, Π_c can be seen as an extention of Π. Membership of a node v in one of the sets $W_1, \ldots, W_d$ will enforce a certain behaviour of every solution $S \in \mathcal{S}_{\Pi_c}(G, W_1, ..., W_d)$ on v. When there are no constraints, we obtain solutions of the original problem Π. We will therefore drop the subscript c when the setting is clear from the context.

A simple example is Constrained Independent Set:

$$V' \subseteq V \text{ is a CIS } :\Leftrightarrow \forall v, w \in V' : (v, w) \notin E \\ \text{and } W_1 \subseteq V' \text{ and } W_2 \cap V' = \emptyset.$$

where we use $|V'|$ as the objective function.

Every $V' \in \mathcal{S}_{CIS}(G, W_1, W_2)$ is forced to contain the set W_1, and have an empty intersection with W_2. An optimal solution is therefore an independent set with maximum cardinality under the opposed constraints.

An intuitive way of describing a set of constraints on a graph $G = (V, E)$ is

by coloring its nodes. Name $C = \{1, \ldots, d\}$ the set of *colors* of a constrained problem and call a mapping of the form $\chi : U \to C$ with $U \subseteq V$ a *coloring*.

It is clear that an arbitrary family of colorings $\chi_j : U_j \to C$ with $j = 1, \ldots, t$ defines a set of constraints by $W_i := \bigcup_{j=1}^{t} \chi_j^{-1}(i)$ for $i = 1, \ldots, d$. On the other hand, it is easy to see that every set of constraints $\{W_1, \ldots, W_d\}$ can be described by a – not necessarily unique – family of colorings. However, note that sets of pairwise disjoint constraints and single colorings are in $1-1$ correspondence by the above equations.

2.3.2.2 Resolving Constraints

Next we provide a concept that allows us to characterize the constraints of every instance by only *one* coloring. For a given set of constraints $\{W_1, \ldots W_d\}$, we need to find a family of *pairwise disjoint* constraints $\{U_1, \ldots, U_d\}$ such that the corresponding spaces of solutions $\mathcal{S}_\Pi(G, W_1, \ldots W_d)$ and $\mathcal{S}_\Pi(G, U_1, \ldots U_d)$ coincide. To obtain such a disjoint family we will define a rule to successively *resolve* multiple constraints on every node until we end up with a disjoint partition.

Note however, that non-disjoint constraints allow inconsistent assignments to one node – for example, a node can at the same time be required to be in the INDEPENDENT SET and *not* be a member of it. In order to be able to resolve these kinds of constraints, we need to extend our color set by an "illegal value" $\star$. Any coloring that maps a node to $\star$ will also be named $\star$.

Let $C' = C \cup \{\star\}$ and let $\cdot : C' \times C' \to C'$ be an associative and commutative mapping with the properties $(i, i) \mapsto i$ and $(\star, i) \mapsto \star$ for every $i \in C'$. Call such a mapping a *resolving rule*. This mapping can be extended to a multiplication of colorings in a natural way: let $G = (V, E)$ be a graph and $V_1, V_2 \subseteq V$. For colorings $\chi_1 : V_1 \to C'$ and $\chi_2 : V_2 \to C'$ we define $\chi_1 \cdot \chi_2$ by

$$(\chi_1 \cdot \chi_2)(v) := \begin{cases} \chi_1(v) & \text{if } v \in V_1 \setminus V_2 \\ \chi_2(v) & \text{if } v \in V_2 \setminus V_1 \\ \chi_1(v) \cdot \chi_2(v) & \text{if } v \in V_1 \cap V_2 \end{cases}$$

Note that the extension of $\cdot$ to colorings is also both commutative and asso-

ciative and that the equation

$$\chi_{\restriction V_1} \cdot \chi_{\restriction V_2} = \chi \tag{2.2}$$

holds for every coloring χ with domain $V_1 \cup V_2$, that is, there is no conflict to be resolved if a node is assigned the same color by two colorings.

Definition 2.8. We say that a constrained graph optimization problem Π with color set C is *resolvable*, if there is a resolving rule $\cdot : C' \times C' \to C'$, such that its extension to colorings fulfils the equation

$$\mathcal{S}_\Pi(G, \chi_1^{-1}(1) \cup \chi_2^{-1}(1), ..., \chi_1^{-1}(d) \cup \chi_2^{-1}(d)) = \mathcal{S}_\Pi(G, \chi^{-1}(1), ..., \chi^{-1}(d))$$

for every graph G and every pair of colorings χ_1, χ_2 with $\chi = \chi_1 \cdot \chi_2$ where we set $\mathcal{S}_\Pi(G, \chi^{-1}(1), \ldots, \chi^{-1}(d)) := \emptyset$ if $\chi = \star$.

A resolving rule for Constrained Independent Set is given by the mapping $\cdot$ induced by $\cdot : \{1, 2\} \mapsto \star$. It is clear that CIS is resolvable with this rule.

As already mentioned, the aim behind this multiplication of mappings is to state an order independent way of obtaining a disjoint family of constraints. In the context of mappings, however, it can be formalized more easily. The concept of overlaying colors may serve as a model. With the above considerations, it is clear that every instance of a *resolvable* constrained graph optimization can be entirely described by a pair (G, χ) consisting of a graph G and a coloring χ.

We are now ready to define the notion of *dissectability*:

2.3.2.3 Dissectable Problems

Let Π be a resolvable constrained optimization problem on graphs and let $C = \{1, ..., d\}$ be the corresponding set of colors.

Definition 2.9. We say that Π is *dissectable*, if there is a permutation Φ of C so that for every instance (G, ψ) and every separation (V_1, S, V_2) of G we can find fragments G_1 and G_2 so that the following conditions hold:

(i) For every σ in $\mathcal{S}_\Pi(G, \psi)$ we find a coloring $\chi : S \to C$ such that $\sigma \cap G_1 \in \mathcal{S}_\Pi(G_1, \psi_{\restriction G_1} \cdot \chi)$ and $\sigma \cap G_2 \in \mathcal{S}_\Pi(G_2, \psi_{\restriction G_2} \cdot \Phi\chi)$

(ii) For every coloring $\chi : S \to C$ there is a constant $c(\chi)$ such that for every $\sigma_1 \in \mathcal{S}_\Pi(G_1, \psi_1 \cdot \chi)$ and for every $\sigma_2 \in \mathcal{S}_\Pi(G_2, \psi_2 \cdot \Phi\chi)$ we have $\sigma_1 \cup \sigma_2 \in \mathcal{S}_\Pi(G, \psi_1 \cdot \psi_2)$, and the equation

$$\nu_\Pi(\sigma_1 \cup \sigma_2) = \nu_\Pi(\sigma_1) + \nu_\Pi(\sigma_2) - c(\chi)$$

holds.

The coloring χ characterizes the boundary behaviour of the solutions, and the permutation Φ models the idea of *compatibility*. In this light, item *(ii)* requires that the union of subsolutions with compatible boundary behaviour results in a feasible solution on the union of the subgraphs. In addition to this, item *(i)* requires that restrictions of solutions are subsolutions, a fact closely related to the optimal substructure property.

2.3.2.4 Some examples

In this section we give two examples of dissectable problems. More can be found in chapter 3

Observe that, in the case $\mathcal{S}_\Pi(G, \psi) = \emptyset$, item *(i)* of definition 2.9 is vacuously true. The same holds for part *(ii)* in a case where $\mathcal{S}_\Pi(G_1, \psi_1 \cdot \chi) = \emptyset$ or $\mathcal{S}_\Pi(G_2, \psi_2 \cdot \Phi\chi) = \emptyset$. When examining our examples, we shall therefore always assume non-empty spaces of solutions.

Independent Set: A simple but important example is CONSTRAINED INDEPENDENT SET. We have already given the definition and observed its resolvability. What remains is to examine whether it fulfils definition 2.9.

Let $G = (V, E)$ be a graph, (V_1, S, V_2) a separation. The fragments corresponding to the separation are defined as $G_1 = G[V_1 \cup S]$ and $G_2 = G[V_2 \cup S]$.

Now for points *(i)* and *(ii)* of definition 2.9. Firstly, we set $\Phi = id$. Let (G, ψ) be an instance, V' a solution and (V_1, S, V_2) a separation. With $\chi : S \to \{1, 2\}$ defined as

$$\chi(v) = \begin{cases} 1 & \text{if } v \in V' \\ 2 & \text{otherwise} \end{cases}$$

point *(i)* is clearly fulfilled.

For part *(ii)* let $V_1' \in \mathcal{S}_\Pi(G_1, \psi_1 \cdot \chi)$ and $V_2' \in \mathcal{S}_\Pi(G_2, \psi_2 \cdot \chi)$. Obviously $V_1' \cup V_2' \in \mathcal{S}_\Pi(G, \psi_1 \cdot \psi_2)$ holds, for no edge joins nodes in V_1 and V_2 and both constrained independent sets coincide on the common nodes due to the restriction given by χ. With the definition of $c(\chi) = |\chi^{-1}(1)|$ we avoid counting these nodes twice. Condition *(ii)* of 2.9 is therefore also fulfilled by *CIS*.

Edge Dominating Set: A more complicated example is EDGE DOMINATING SET. This problem differs from the preceding one, for it is an optimization problem on *edges*.

We state the constrained version using three node predicates - W_1 for "must be adjacent to $e \in E'$ ", W_2 for "is adjacent to a dominating edge from somewhere else" and W_3 for neither of both. Let $G = (V, E)$ be a graph.

$$\begin{aligned} E' \subseteq E \text{ is a } \mathit{CEDS} \ :\Leftrightarrow \forall e \in E : (e \in E' \\ &\text{or } \exists e' \in E' : e \text{ is adjacent to } e' \\ &\text{or } \exists v \in W_2 : e \text{ is adjacent to } v) \\ &\text{and } \forall v \in W_1 \setminus W_2 : \exists e' \in E' : e' \text{ adjacent to } v. \end{aligned}$$

In the case $W_1 = W_2 = W_3 = \emptyset$ we obtain the formulation of EDGE DOMINATING SET. The constraints are resolvable: map any pair containing 2 and

not $\star$ to 2, assign 1 to the pair $(1,3)$ and define the remaining cases according to the requirements for $\cdot$ on page 53.

Let $G = (V, E)$ be a graph, (V_1, S, V_2) a separation. The fragments corresponding to the separation are defined as $G_1 = G[V_1 \cup S]$ and $G_2 = G[V_2 \cup S] \setminus E(G[S])$. In fact, *any* disjoint partition of the edges in $G[S]$ yields suitable graph fragments.

Define $\Phi = \tau_{1,2}$, the transposition that swaps 1 and 2. Let (G, ψ) be an instance, E' a solution and (V_1, S, V_2) a separation. Define $\chi : S \to C$ as

$$\chi(v) = \begin{cases} 1 & \text{if } v \text{ is adjacent to } e \in E' \cap G_1 \text{ or } v \in \psi^{-1}(2) \\ 2 & \text{if } v \text{ is adjacent to } e \in E' \cap G_2 \text{ or } v \in \psi^{-1}(1) \\ 3 & \text{otherwise} \end{cases}$$

Note that χ is not well-defined, for the first two cases might occur simultaneously. In this situation, arbitrarily choose one of the two possible definitions.

Observe that the only edges to be examined for item *(i)* of definition 2.9 are the ones adjacent to nodes in S. Let $v \in S$ and let e be an edge in G_1 with e adjacent to v. If v is colored with 2 by ψ or if v is adjacent to $e' \in E'$, the definitions of χ and $\cdot$ assure that either e' in $E(G_1)$ or that v is colored by 2 in G_1. Thus the CEDS-condition is fulfilled for e in $(G_1, \psi_{\restriction G_1} \cdot \chi)$. If neither $v \in \chi^{-1}(2)$ nor v adjacent to $e' \in E'$, e must be dominated via the other adjacent node, for E' is a *CEDS* in (G, ψ) by assumption. We see that $E' \cap G_1$ is a CEDS in $G_1(\psi_{\restriction G_1} \cdot \chi)$.

Item *(ii)* is also easy to see. Let $E_1' \in \mathcal{S}_\Pi(G_1, \psi_1 \cdot \chi)$ and $E_2' \in \mathcal{S}_\Pi(G_2, \psi_2 \cdot \Phi\chi)$. We have to examine whether $E_1' \cup E_2' \in \mathcal{S}_\Pi(G, \psi_1 \cdot \psi_2)$ holds. The only edges in E where the *CEDS*-condition could be violated are the ones adjacent to nodes $v \in S$ with $\chi(v) = 2$ or $\Phi\chi(v) = 2$. We have $\Phi = \tau_{1,2}$, and so it is ensured by $\cdot$ that v is either adjacent to $e \in E_1' \cup E_2'$ or in $(\psi_1 \cdot \psi_2)^{-1}(2)$. We conclude that $E_1' \cup E_2' \in \mathcal{S}_\Pi(G, \psi_1 \cdot \psi_2)$ holds. The edge sets $E(G_1)$ and $E(G_2)$ are disjoint, we consequently define $c(\chi) = 0$. This concludes this example.

2.3.3 Optimal substructure and dynamic programming

In this chapter we examine why the the structure of dissectable problems is suitable for an algorithmic approach using dynamic programming techniques.

The key to the upcoming algorithm and the corresponding correctness considerations is the following lemma. Its first part states that dissectable problems have the *optimal substructure property*. This means that any optimal solution contains within it optimal solutions to subproblems. From the second part we can easily derive a recipe for an algorithm using dynamic programming techniques.

Lemma 2.3. *Let Π be a dissectable constrained graph optimization problem with color set C, let G be a graph, (V_1, S, V_2) a separation and G_1, G_2 the corresponding graph fragments.*

(i) Let $\sigma \in \mathcal{S}_\Pi(G, \psi)$ be optimal and choose χ as in (i) of definition 2.9. Then $\sigma \cap G_1$ is optimal in $\mathcal{S}_\Pi(G_1, \psi_{\restriction G_1} \cdot \chi)$ and $\sigma \cap G_2$ is optimal in $\mathcal{S}_\Pi(G_2, \psi_{\restriction G_2} \cdot \Phi\chi)$

(ii) $$\operatorname{opt}(G, \psi) = \operatorname*{opt}_{\chi: S \to C} \big(\operatorname{opt}(G_1, \psi_{\restriction G_1} \cdot \chi) + \operatorname{opt}(G_1, \psi \cdot_{\restriction G_2} \Phi\chi) - c(\chi)\big)$$

Proof. Without loss of generality we assume Π is a minimization problem. Assume first that all spaces of solutions are non-empty. For *(i)* assume that for example $\sigma \cap G_1$ is not minimal. Then there is $\sigma' \in \mathcal{S}_\Pi(G_1, \psi_{\restriction G_1} \cdot \chi)$ with $\nu(\sigma') < \nu(\sigma \cap G_1)$. By item *(ii)* of definition 2.9 and equation 2.2 on page 54 we have $\sigma' \cup (\sigma \cap G_2) \in \mathcal{S}_\Pi(G, \psi)$ with the value $\nu(\sigma' \cup (\sigma \cap G_2)) = \nu(\sigma') + \nu(\sigma \cap G_2) - c(\chi) < \nu(\sigma \cap G_1) + \nu(\sigma \cap G_2) - c(\chi) = \nu(\sigma)$. This contradicts σ's minimality.

To show part *(ii)*, for a given χ we choose optimal $\sigma_1 \in \mathcal{S}_\Pi(G_1, \psi_{\restriction G_1} \cdot \chi)$ and $\sigma_2 \in \mathcal{S}_\Pi(G_2, \psi_{\restriction G_2} \cdot \Phi\chi)$. According to definition 2.9, $\sigma_1 \cup \sigma_2 \in \mathcal{S}_\Pi(G, \psi)$ holds, with $\nu(\sigma_1 \cup \sigma_2) = \nu(\sigma_1) + \nu(\sigma_2) - c(\chi)$. With this we can easily see that

$$\operatorname{opt}(G, \psi) \leq \operatorname*{opt}_{\chi: S \to C} \big(\operatorname{opt}(G_1, \psi_{\restriction G_1} \cdot \chi) + \operatorname{opt}(G_2, \psi_{\restriction G_2} \cdot \Phi\chi) - c(\chi)\big) \qquad (2.3)$$

On the other hand, let $\sigma \in \mathcal{S}_\Pi(G, \psi)$ be optimal. Because of definition 2.9, we find a coloring χ with $\sigma \cap G_1 \in \mathcal{S}_\Pi(G_1, \psi_{\restriction G_1} \cdot \chi)$ and $\sigma \cap G_2 \in \mathcal{S}_\Pi(G_2, \psi_{\restriction G_2} \cdot \Phi\chi)$

as well as $\mathrm{opt}(G,\psi) = \nu(\sigma \cap G_1) + \nu(\sigma \cap G_2) - c(\chi)$. With the help of part *(i)* of lemma 2.3, this is transformed to

$$\mathrm{opt}(G,\psi) = \mathrm{opt}(G_1, \psi_{\restriction G_1} \cdot \chi) + \mathrm{opt}(G_2, \psi_{\restriction G_2} \cdot \Phi\chi) - c(\chi)$$

Taking the minimum over all colorings χ, we obtain

$$\mathrm{opt}(G,\psi) \geq \operatorname*{opt}_{\chi: S \to C} \big(\mathrm{opt}(G_1, \psi_{\restriction G_1} \cdot \chi) + \mathrm{opt}(G_2, \psi \cdot_{\restriction G_2} \Phi\chi) - c(\chi)\big) \qquad (2.4)$$

Combining equations 2.3 and 2.4 we obtain the desired equation.

In case $\mathcal{S}_\Pi(G,\psi) = \emptyset$ we define $\mathrm{opt}(G,\psi) = \infty$. With considerations similar to the above we see that the equations still hold.

□

2.4 The algorithms

In this section we give the pseudocode to the algorithms of Baker and Alber. In fact, we present modified versions of the original works, adapted to suit our unified approach. However, the core idea of TABLE and its submethods were first presented in [5], whereas the method DECIDE is strongly inspired by the work in [4].

First, we give an overview of how we intend to piece together the presented concepts – decomposition of an instance $\langle G, k\rangle$ into $\mathcal{O}(\sqrt{k})$-outerplanar subgraphs, further separation into trivial subgraphs and a decomposition-based calculation of solutions. After that we describe the data structures and methods common to both parts. With their help, we can easily state the pseudocode of both algorithms in the subsequent chapter.

For the entire section, assume without loss of generality that Π is a parameterized problem, obtained from a dissectable *minimization* problem with color set C.

2.4.1 Plugging it all together

The version of Baker's algorithm we present is general enough to easily adapt to any dissectable problem - including proof of correctness - and to fit together with the ideas of Alber et al. On the other hand, we try to retain some resemblance to Baker's original work to be able to recycle used data structures and running time analysis made in [5].

2.4.1.1 How it could be done

Section 2.2.3 and part *(ii)* of lemma 2.3 suggest the following approach: for subgraphs $H \subseteq G$ with distinguished node set $U \subseteq V(H)$ we use tables with entries labeled by colorings $\chi : U \to C$. Each entry contains the size of an optimal solution of Π on H under the constraint χ. Starting with tables for trivial fragments as in remark 2.2.3 on page 49, one builds up tables for unions of slice subgraphs, ending up with a table for every $\mathcal{O}(\sqrt{k})-$outerplanar fragment. Determining the value of optimal solutions on unions is achieved by iterating over all colorings of the separating vertices following part *(ii)* of

lemma 2.3. The number of colorings defined on a node set of size r is given by $|C|^r$ – explaining why small separators are worthwile to consider.

Having constructed tables containing solutions on the $\mathcal{O}(\sqrt{k})-$ outerplanar fragments for all possible constraints on the separators obtained BALANCED SEPARATION, we continue this merging process until we end up with a table for the entire graph. This is the straightforward approach suggested by the theory developed above.

In practice, as already mentioned, we use a slightly different method.

2.4.1.2 What we do

To be able to adapt some of the data structures and running time analysis from [5], we give a modification of the original recursive algorithm.

The significant difference is that Baker's original algorithm only treats the case of unconstrained problems. Applied to a graph H, it consequently returns a single number $\mathrm{opt}(H)$ – instead of the desired table with values $\mathrm{opt}(H, \psi)$ for every possible coloring ψ of a distinguished node set.

To obtain a table with values $\mathrm{opt}(H, \psi)$, we do the following: when Baker's algorithm is called, it is given an additional argument – the coloring ψ. The coloring ψ or, more precisely, its restriction to the respective subgraph, is handed down the recursion. When the recursion ends, that is, when the algorithm CREATE builds a table for a trivial subgraph $H' = (V', E')$, we do in fact return a table whose entries contain $\mathrm{opt}(H', \chi\psi_{|V'})$. The entries are labeled by the coloring χ. As we will see, a call to the thus modified algorithm yields a single number, namely $\mathrm{opt}(H, \psi)$. In order to build up our table, all we have to do is to apply the algorithm for every coloring ψ.

For simplicity of notation in the pseudocode, however, we make the coloring ψ a global variable instead of handing it down through the entire recursion.

2.4.2 Basic Tools

2.4.2.1 The Tables

As data structures we use tables T labeled by colorings $\chi : W \to C$ for some node set $W \subseteq V$. We call the node set W the *signature* $\mathrm{sig}\,(T) = W$ of the

table T. Observe that the number of entries $T(\chi)$ is given by $|C|^{|W|}$.

For reasons of convenience, we define a table with signature $\emptyset$ to be a numeric variable.

2.4.2.2 Merge

The key method is named MERGE. It combines optimal solutions on graph fragments to optimal solutions on the union graph. The algorithm – and its correctness – are direct consequences of lemma 2.3.

The input arguments are tables T_1 and T_2, together with node sets S and V_3. The tables T_1 and T_2 are intended to contain optimal solutions on fragments that are separated by the set S. The set V_3 takes the place of the signature of the table to be constructed. With $V_i = \mathrm{sig}(T_i)$ for $i = 1, 2$ we consequently require $V_1 \cap V_2 = S$ and $S \cup V_3 = V_1 \cup V_2$ as preconditions for MERGE.

MERGE(T_1, T_2, S, V_3)
1 Let T_3 be an empty table with signatureV_3
2 $V_1 \leftarrow \mathrm{sig}\,(T_1), V_2 \leftarrow \mathrm{sig}\,(T_2)$
3 **for** all $\psi : V_3 \to C$
4 **do** $m \leftarrow \infty$
5 **for** all $\chi : S \to C$
6 **do** $m \leftarrow \min\{m, T_1(\psi_{|V_1} \cdot \chi) + T_2(\psi_{|V_2} \cdot \Phi\chi) - c(\chi)\}$
7 $T_3(\psi) \leftarrow m$
8 **return** T_3

Note that there is precisely *one* mapping $\chi : \emptyset \to C$. In coherence with the definition of tables, MERGE returns a single value in case $V_3 = \emptyset$.

2.4.2.3 Create

CREATE is the method that builds up tables for the trivial fragments mentioned in remark 2.2.3 of section 2.2.3.

In our approach, CREATE is the *only* problem-dependent method. Thus we will state it as a template and consider the details of the generic part afterwards.

Let $V' \subseteq V$ be the node set of a fragment $H = (V', E')$ corresponding to remark 2.2.3. The coloring ψ is the global variable referred to in section 2.4.1.2

CREATE(V')
1 Let T be an empty table with signature V'
2 **for** all $\chi : V' \to C$
3 **do** $T(\chi) \leftarrow \mathrm{opt}(H, \chi \cdot \psi_{\restriction V'})$
4 **return** T

It is clear that the generic part is the right hand side of the assignment in line 3. There are two problem-specific variables:

- the edge set E'.
- the value $\mathrm{opt}(H, \chi \cdot \psi_{\restriction V'})$

For the first point, all we have to do is to characterize the fragments corresponding to the trivial subgraphs. In the case of DOMINATING SET and INDEPENDENT SET, this task is easy: The fragments are the *induced subgraphs* of the corresponding node set.

The case of EDGE DOMINATING SET must be treated differently: we need to ensure that every edge is contained in exactly *one* fragment. We do this by marking edges. Start with $G = (V, E)$ with unmarked edge set E. Every time CREATE is called on node set V', take as edge set E' all unmarked edges in $E \cap (V' \times V')$. After CREATE has ended, we mark all edges in E'. It is clear that this yields graph fragments according to the considerations on page 56.

The second point can be handled by a brute-force approach: By remark 2.2.3, the set V' contains at most 3 nodes, the corresponding table T's number of entries is therefore bounded by $|C|^3$. We are dealing with problems in $\mathcal{NP}$, so we can calculate $\mathrm{opt}(H, \chi\psi_{\restriction V'})$ on H in constant time for every coloring χ.

It should be emphasized that the method CREATE plays the crucial role in the combination of the algorithms in [5] and [4]. This fact was addressed in paragraph 2.4.1.2 and will become important in section 2.5.1.

2.4.2.4 Extend

Following [5], there are more problem-specific methods to implement. One of them is the method called EXTEND. According to the separation of the slice subgraph of a level $i > 1$ leaf $v = (x, y)$ with $\mathcal{N}(v) \neq \emptyset$ described in section 2.2.3 and depicted in figure 2.7, it extends the table for a level $i - 1$ vertex $w \in \mathcal{N}(v)$ by the node x or y, respectively.

In our approach, however, the method EXTEND is not of an elementary type, for it can be reduced to a simple application of MERGE and CREATE. It is only introduced to provide a connection to the original work in [5] and to improve readability of the pseudocode. It contains a call to the method TABLE, which is being stated in the subsequent paragraph.

The method takes as arguments a vertex $w = (a, b)$ and a node x. It returns a table T for the fragment corresponding to the node set $\mathrm{sl}(w) \cup \{x\}$ with signature $\partial_{\mathrm{l}}(w) \cup \partial_{\mathrm{r}}(w) \cup \{x\}$.

EXTEND$(w = (a, b), x)$
1 $T_1 \leftarrow$ CREATE(a, b, x)
2 $T_2 \leftarrow$ TABLE(w)
3 **return** MERGE$(T_1, T_2, \{a, b\}, \partial_{\mathrm{l}}(w) \cup \partial_{\mathrm{r}}(w) \cup \{x\})$

2.4.3 The pseudocode

2.4.3.1 Table - Baker's algorithm

Baker's algorithm is stated as the recursive method called TABLE. We present a modified version of the original work. These differences are direct consequences of our differing mode of presentation in the previous chapters and the consequent separator-based approach.

TABLE takes as an argument a vertex $v = (x, y)$ of the structure tree $\mathcal{T}_G$ and returns a table T with signature $\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v)$ for the slice subgraph G_v in the case of $v \neq \mathrm{root}(\mathcal{T}_G)$ and the value $\mathrm{opt}(G, \chi)$ in the case of $v = \mathrm{root}(\mathcal{T}_G)$. For increased readability the algorithm is split into two parts, TABLE I and TABLE II. The first is handling the case in which v is a leaf, the second takes care of the case in which v is a face vertex.

TABLE I($v = (x, y)$ a level i leaf)

```
 1  if i = 1
 2     then return CREATE(v)
 3  if N(v) = ∅
 4     then let ∂l(v) = {x, x_{i-1}, ..., x_1}
 5          T ← CREATE(x, y, x_{i-1})
 6          for j = i − 1 to 2
 7          do T ← MERGE(T, CREATE(x_j, x_{j-1}), x_j, {x, y, ..., x_{j-1}})
 8          return T
 9  Let w_1, ..., w_t be the children of the enclosing face's vertex
10  if y is adjacent to z_r with(z_r, z_{r+1}) ∈ N(v)
11     then let p be the least such r
12     else  let p be such that w_p = Δr(v)
13  Let q, s be such that w_q = Δr(v) and w_s = Δl(v)
14  T← EXTEND(w_s, x)
15  for j ← s + 1 to p − 1
16  do T_0 ← EXTEND(w_j, x)
17       T ← MERGE(T, T_0, ∂l(w_j) ∪ {x}, ∂r(w_j) ∪ ∂l(v))
18  T_0 ← EXTEND(w_p, y)
19  T ← MERGE(T, T_0, ∂l(w_p), ∂r(w_p) ∪ ∂l(v))
20  for j ← p + 1 to q
21  do T_0 ← EXTEND(w_j, y)
22       T ← MERGE(T, T_0, ∂l(w_j) ∪ {y}, ∂r(w_j) ∪ ∂l(v))
23  T ← MERGE(T, CREATE(x, y), {x, y}, ∂l(v) ∪ ∂r(v))
24  return T
```

TABLE II($v = (x, y)$ a face vertex)
1 $u \leftarrow v$
2 **while** face (u) has enclosed component H
3 **do** $u \leftarrow \text{root}\,(\mathcal{T}_H)$
4 Let $u_1, ..., u_t$ be the children of u
5 Let j with $\partial_{\text{l}}(u_j) \cap \partial_{\text{r}}(u_j) = \emptyset$
6 $T_1 \leftarrow$ TABLE(u_{j+1})
7 **for** $i \leftarrow j + 2$ **to** t
8 **do** $T_1 \leftarrow$ MERGE$(T_1,$ TABLE$(u_i), \partial_{\text{l}}(u_i), \partial_{\text{l}}(u_{j+1}) \cup \partial_{\text{r}}(u_i))$
9 $T_2 \leftarrow$ TABLE(u_1)
10 **for** $i \leftarrow 2$ **to** $j - 1$
11 **do** $T_2 \leftarrow$ MERGE$(T_2,$ TABLE$(u_i), \partial_{\text{l}}(u_i), \partial_{\text{l}}(u_1) \cup \partial_{\text{r}}(u_i))$
12 **if** $v = \text{root}\,(G)$
13 **then** $B \leftarrow \emptyset$
14 **else** $B \leftarrow \partial_{\text{l}}(v) \cup \partial_{\text{r}}(v)$
15 $T \leftarrow$ MERGE$(T_1, T_2, \partial_{\text{l}}(u_t) \cap \partial_{\text{r}}(u_1), B \cup \partial_{\text{l}}(u_j) \cup \partial_{\text{r}}(u_j))$
16 $T \leftarrow$ MERGE$(T,$ TABLE$(u_j), \partial_{\text{l}}(u_j) \cup \partial_{\text{r}}(u_j), B)$
17 **if** $x \neq y$
18 **then** $T \leftarrow$ MERGE$(T,$ CREATE$(x, y), \{x, y\}, \partial_{\text{l}}(v) \cup \partial_{\text{r}}(v))$
19 **return** T

2.4.3.2 Decide - The frame

In this subsection we state the final algorithm, captured in a method called DECIDE. After the work of the previous chapters, it can be described in compact form. A short description of the idea was given in section 2.4.1.2. Note that in accordance with 2.4.1.2, the variable ψ is visible inside the method TABLE and its submethods, i.e. CREATE and MERGE.

Let Π be the parameterized version of a dissectable graph problem with LSP of width w and size-factor d. Let G be a planar graph. Choose $\beta > 0$ as an arbitrary trade-off-parameter.

```
DECIDE(⟨G, k⟩)
 1  Calculate planar embedding
 2  Calculate layer decomposition (L_i)_{1≤i≤r}
 3  S ← BALANCED SEPARATION((L_i)_{1≤i≤r}, √(dk), β, w)
 4  if S = NIL
 5     then return no
 6  Let S = (S_i)_{1≤i≤q} and set S_{q+1} ← ∅
 7  Calculate the fragments (G_i)_{0≤i≤q}
 8  Construct the forests (T_{G_i})_{0≤i≤q}
 9  Let T be an empty table with signature S_1
10  for all ψ : S_1 → C
11  do T(ψ) ← TABLE(root (T_{G_0}))
12  for i ← 2 to q + 1
13  do let T_0 be an empty table with signature S_{i−1} ∪ S_i
14        for all ψ : (S_{i−1} ∪ S_i) → C
15        do T_0(ψ) ← TABLE(root (T_{G_{i−1}}))
16        T ← MERGE(T, T_0, S_{i−1}, S_i)
17  if T > k
18     then return no
19     else  return yes
```

2.5 The analysis

2.5.1 Correctness

Following the previous chapters' work the correctness proof is quite simple. We start with the key ingredient, the correctness proof for MERGE. Therefore let Π be a dissectable graph optimization problem.

Lemma 2.4. *Let $G = (V, E)$ be a graph, ψ a coloring and (V_1, S, V_2) a separation with graph fragments G_1 and G_2 as in definition 2.9. Let $U_i \subseteq V(G_i)$ for $i = 1, 2$ and let $U_3 \subseteq V$ with $U_3 \cup S = U_1 \cup U_2$. Assume we have tables T_i with signature U_i and the property that $T_i(\chi) = \mathrm{opt}(G_i, \psi_{\restriction G_i} \cdot \chi)$ for every coloring $\chi : U_i \to C$ with $i = 1, 2$.*

Then a call of MERGE(T_1, T_2, S, U_3) *returns a table T with signature U_3 and*

$$T(\phi) = \mathrm{opt}(G, \psi \cdot \phi)$$

for every $\phi : U_3 \to C$.

Proof. From the pseudocode we see that

$$T(\phi) = \operatorname*{opt}_{\chi : S \to C} (T_1(\phi_\restriction \cdot \chi) + T_2(\phi_\restriction \cdot \Phi\chi) - c(\chi))$$

holds. By hypothesis this is transformed into

$$T(\phi) = \operatorname*{opt}_{\chi : S \to C} (\mathrm{opt}(G_1, \psi_{\restriction G_1} \cdot (\phi_{\restriction G_1} \cdot \chi)) + \mathrm{opt}(G_2, \psi_{\restriction G_2} \cdot (\phi_{\restriction G_2} \cdot \Phi\chi)) - c(\chi))$$

Using associativity of $\cdot$ and lemma 2.3 this yields

$$T(\phi) = \mathrm{opt}(G, \psi_{\restriction G_1} \cdot \phi_{\restriction G_1} \cdot \psi_{\restriction G_2} \cdot \phi_{\restriction G_2})$$

from which we finally obtain

$$T(\phi) = \mathrm{opt}(G, \psi \cdot \phi).$$

using commutativity of $\cdot$ and equation (2.2) from page 54. □

The next lemma is credited to Baker and shows that the algorithm terminates.

Lemma 2.5. *Calling* TABLE *on* root $(\mathcal{T}_G)$ *leads to exactly one recursive call of* TABLE *on every other vertex of* $\mathcal{T}_G$.

For a proof, see [5].

The upcoming statement is a modification of another lemma by Baker and states the correctness of the method TABLE on a graph with given coloring ψ.

Lemma 2.6. *For every coloring* $\psi : V \to C$, *a call of* TABLE *on* root (G) *returns the value* $\mathrm{opt}(G, \psi)$.

Proof. The only additional assumption we make is a correct implementation of the method CREATE. We prove by induction on the depth of recursive calls of TABLE (observe lemma 2.5) that a call of TABLE on every vertex $v \neq \mathrm{root}\,(G)$ returns a table T with signature $\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v)$ and that, for every $\chi : \partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v) \to C$ the equation $T(\chi) = \mathrm{opt}(G_v, \chi \cdot \psi_{\restriction G_v})$. holds.

One starts the induction with level 1 leaves and level $i > 1$ leaves with $\mathcal{N}(v) = \emptyset$. The pseudocode for this case is contained in lines 1 to 8 of part (I) of TABLE. In the first case, the assumption follows from the correctness of CREATE, in the latter one a simple induction on i together with the correctness of CREATE and MERGE does the job.

The correctness in the cases of face vertices with and without enclosed components, as well as the case of level i leaves with non-empty set $\mathcal{N}(v)$ is a direct result of the considerations in section 2.2.3 together with the correctness of MERGE in lemma 2.4, for the composition in the pseudocode precisely follows the mode of decomposition in section 2.2.3.
Furthermore observe that, due to proposition 7, in every case the signature of the returned table is precisely $\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v)$,

The case of $v = \mathrm{root}\,(G)$ is slightly different. In line 16 of part (II) of TABLE, MERGE is called with the empty constraint set B, it thus returns the single value $\mathrm{opt}_\Pi(G, \psi)$ by definition and induction hypothesis. This finishes our proof. □

Theorem 2.7. *Let* Π *be the parameterized version of an dissectable optimization problem on planar graphs and let* C *be the corresponding color set. Furthermore assume that* Π *has the LSP of width* w *and size factor* d. *Then, for every* $\langle G, k\rangle$, *the algorithm* DECIDE *correctly decides whether it is a* yes-*instance.*

Proof. Π has the Layerwise Separation Property. By theorem 2.1 and the correctness of the algorithm BALANCED SEPARATION we see that, in the case that $\langle G, k\rangle$ is a *yes*-instance, we obtain a $\sqrt{dk} - \beta-$balanced separation. Therefore, if $\mathcal{S} = NIL$, we reject $\langle G, k\rangle$ with good reason. If $\mathcal{S} \neq NIL$, an induction on the variable i in the loop in line 12 shows that T has signature S_i and that the equation $T(\chi) = \mathrm{opt}(\bigcup_{j=0}^{i} G_j, \chi)$ holds for every $\chi : S_i \to C$. This, together with $S_{q+1} = \emptyset$, shows that T contains $\mathrm{opt}_\Pi(G)$ at the end. If, in the case of a minimization problem, this value exceeds k, we can reject $\langle G, k\rangle$. If not, we accept it. Thus, DECIDE correctly performs its assigned task. $\square$

2.5.2 Running time analysis

When doing a straightforward analysis of the method TABLE stated as above, one yields a slightly worse running time than obtained in [5]. In order to get the best possible worst case complexity, we adapt some technical details from [5]. It will become clear that there must be different table structures and different implementations of the method MERGE, depending on whether it is used from DECIDE or TABLE. However, it will remain clear that the theoretical analysis in the previous chapter still holds.

2.5.2.1 The method Table

Let G be an $r-$outerplanar graph and $\mathcal{T}_G$ its structure tree. The key to avoid the multiplication of colorings in calls to MERGE and to ease the task of identifying common nodes in the signatures of tables T_1 and T_2 can be solved as follows: we use 2-dimensional tables for the slice of every vertex v, every entry indexed by *two* colorings $\chi_l : \partial_{\mathrm{l}}(v) \to C$ and $\chi_r : \partial_{\mathrm{r}}(v) \to C$. Thus the table entry $T(\chi_l, \chi_r)$ will contain the value formerly saved in the entry $T(\chi_l \cdot \chi_r)$. In calls to MERGE from TABLE this approach has two advantages:

- if v is a level i vertex, by item *(iii)* of proposition 7 it is clear that every table entry can be labeled by two elements of C^i - encoded in natural numbers. Moreover, using the fact that $\partial_{\mathrm{l}}(v)$ and $\partial_{\mathrm{r}}(v)$ contain exactly one node from each of the layers $1, \ldots, i$, the correlation of a node and its value under a coloring can easily be made.

- using a table for the constants $c(\chi)$ and - if necessary - for the mapping Φ we can avoid multiplying mappings when merging slices of two adjacent vertices v_i and v_{i+1} as in lines 8 and 11 in part (II) of TABLE. This can be done by replacing V_3 with the pair $(\partial_{\mathrm{l}}(v_i), \partial_{\mathrm{r}}(v_{i+1}))$, and mappings $\chi : V_3 \to C$ with pairs of mappings $\chi_l : \partial_{\mathrm{l}}(v_i) \to C$ and $\chi_r : \partial_{\mathrm{r}}(v_{i+1}) \to C$. Furthermore, substitute $\partial_{\mathrm{r}}(v_i)$ for S and $\min\{m, T_1(\chi_l, \chi) + T_2(\Phi\chi, \chi_r) - c(\chi)\}$ for the right hand side of the assignment in line 6 of MERGE.

This results in a $\mathcal{O}(1)$ time bound for line 6 of MERGE and, with $c = |C|$, yields $\mathcal{O}(c^{3r})$ as an upper bound for such calls of MERGE on vertices of $\mathcal{T}_G$. Similar considerations hold for the calls of MERGE in lines 15 and 16 of part (II) of TABLE. This upper bound dominates the calls to MERGE in line 16 and 18 and from the method EXTEND. For the call in line 15 observe that

$$(\partial_{\mathrm{l}}(u_1) \cap \partial_{\mathrm{r}}(u_t)) \cup (\partial_{\mathrm{l}}(v) \cup \partial_{\mathrm{r}}(v)) = \partial_{\mathrm{l}}(u_1) \cup \partial_{\mathrm{r}}(u_t)$$

holds due to *(i)* and *(iv)* of proposition 7 on page 41. We consequently obtain the same upper bound of $\mathcal{O}(c^{3r})$ for this call of MERGE.

The case of degenerated slices in lines 3-8 of part (I) can be handled in the same time bound. All calls to MERGE in the loop in line 7 consume $\mathcal{O}(\sum_{j=3}^{i+1} c^j) = \mathcal{O}(c^r)$ time, as well as the ones in lines 17, 19, 22 and 23.

These calls dominate the overall running time. According to lemma 2.5, TABLE is called once for every tree vertex. Together with the fact that, due to planarity, $|E| \in \mathcal{O}(|V|)$ holds, we get a total asymptotic running time of $c^{3r}|V|$ for a call of TABLE on root (G). This result is in agreement with the original work in [5]

2.5.2.2 The method Decide

Calculating a planar embedding can be done in time $\mathcal{O}(|V|)$ according to [8]. Following proposition 4 in [4], the corresponding layer decomposition can be obtained in time $\mathcal{O}(|V|)$ as well. As was pointed out in section 2.1.4.6, the balanced separation can be obtained in time $\mathcal{O}(w\beta\sqrt{dk}|V|)$. Finally,the sequence of graph fragments $(G_i)_{0\leq i\leq q}$, together with the tree decompositions can be constructed in $\mathcal{O}(|V|)$ (see [4, 5]). Let $n = |V|$ and $n_i = |V(G_i)|$ for $i = 0, \ldots q$.

The running time of the remaining operations is dominated by the loop in line 12. The sub-loop in line 14 makes $c^{|S_{i-1} \cup S_i|}$ calls to TABLE. Each of the subsequent calls to MERGE can be done with $c^{|S_{i-1}|+|S_i|}n_{i-1}$, this time *including* the multiplication of colorings in at most n_{i-1} steps. Note that we do not to use 2-dimensional tables, because the structure of the separators is not as well-defined as in the method TABLE.

This yields an upper bound of

$$\sum_{i=1}^{q+1} \left(c^{|S_{i-1}|+|S_i|+3out(G_{i-1})} n_{i-1} + c^{|S_{i-1}|+|S_i|} n_{i-1} \right)$$

for the steps for the outer loop together with lines 10 and 11. Using the bounds on the size of the separators $|S_i|$ and the outerplanarity of the fragments G_i this term can be upper-bounded by

$$2(c^{2\sqrt{dk}\beta+3(\sqrt{dk}/\beta+2w)}) \sum_{i=1}^{q+1} n_{i-1} \leq 4nc^{2\sqrt{dk}\beta+3\sqrt{dk}/\beta+6w}$$

from which we obtain an asymptotic running time of

$$\mathcal{O}(nc^{\sqrt{dk}(2\beta+3/\beta)})$$

The $\mathcal{O}(w\beta\sqrt{dk}n)$ running time of the preprocessing steps in lines 1-8 is dominated by this running time. The exponent is minimal for $\beta = \frac{\sqrt{6}}{2}$, yielding an overall time complexity of

$$\mathcal{O}(nc^{2\sqrt{6dk}}) \tag{2.5}$$

Chapter 3

Strength of the presented concepts

In the last chapters, some theoretical concepts were presented and developed. Among them are the *Layerwise Separation Property* of Alber et al. [4] and the notion of *Dissectability*, inspired by the the concept of *weak glueability* from [4] and *glueability* from [3].

In this last chapter we will be concerned with investigating the strength of the presented concepts. The touchstone for the machinery developed by now is the number of problems that can be attacked by it. In order to gain some first impressions, we pick a few problems and examine whether they fulfil one or both of the above properties.

3.1 Planar Independent Set

In [4], Planar Independent Set is proven to have the Layerwise Separation Property of width 1 and size factor 4. Furthermore, a $\mathcal{O}(n2^{4\sqrt{3k}})$ algorithm is given. We have already seen that Planar Independent Set is dissectable with 2 colors. By equation (2.5) on page 73 we thus obtain the same running time using our algorithm.

3.2 Planar Dominating Set

Previous results: As stated in [13], DOMINATING SET is $\mathcal{NP}-$complete, even when restricted to planar graphs. In [1], PLANAR DOMINATING SET is proven to fulfil the Layerwise Separation Property with $w = 3$ and $d = 51$. In [4], it was pointed out that there is no known way to apply the algorithm of [5] to the obtained *constrained* sub-problems. Alternatively, a tree-decomposition-based algorithm is used, yielding running times of $\mathcal{O}(n3^{6\sqrt{34k}})$.

To apply the above algorithm we have to examine whether there is an dissectable version of DOMINATING SET

Dissectability: We state the constrained version using three node predicates - W_1 for "in the set", W_2 and W_3 for "not in the set". The distinction between W_2 and W_3 is made to specify the graph fragment a node is dominated from. Let $G = (V, E)$ be a graph.

$$\begin{aligned} V' \subseteq V \text{ is a CDS} :\Leftrightarrow & \forall v \in V \setminus W_3 : (v \in V' \text{ or } \exists w \in V' : (v, w) \in E) \\ & \text{and } W_1 \subseteq V' \text{ and } (W_2 \cup W_3) \cap V' = \emptyset \end{aligned}$$

Together with V''s cardinality as the objective function, this clearly yields a constrained version of DOMINATING SET.

The constraints are resolvable: any vertex both in W_1 and $W_2 \cup W_3$ is assigned $\star$. Any vertex in $W_2 \cap W_3$ is assigned 3. The corresponding multiplication clearly fulfils our requirements.

For a graph $G = (V, E)$ with separation (V_1, S, V_2), the fragments are defined as $G_1 = G[V_1 \cup S]$ and $G_2 = G[V_2 \cup S]$.

Now for points *(i)* and *(ii)* of definition 2.9. We first define $\Phi = \tau_{2,3} \in S_3$, the transposition that swaps elements 2 and 3. Together with the definition of CDS the idea is clear: nodes that are dominated from one side of the separator (members of W_2 in that graph fragment), do not need to be dominated from the other side (for they are members of W_3).

Now let (G, ψ) be an instance, V' a CDS and (V_1, S, V_2) a separation. We define $\chi : S \to C$ as

$$\chi(v) = \begin{cases} 1 & \text{if } v \in V' \\ 2 & \text{if there is } w \in V' \text{ with } (v,w) \in E(G_1) \text{ or } v \in \psi^{-1}(3) \\ 3 & \text{otherwise} \end{cases}$$

Now for part *(i)* of definition 2.9. Let V' be a CDS on (G, ψ). We have to examine whether, for example, $V' \cap V_1$ fulfils this property on $(G_1, \psi_{\restriction_{G_1}} \cdot \chi)$ The condition is clearly satisfied for every vertex $v \in V_1 \setminus S$, as its neighboring vertices remain unchanged. Assume now that $v \in S$. If $v \in V'$, we have v colored by 1 in $(G_1, \psi_{\restriction_{G_1}} \cdot \chi)$. But as $v \in V' \cap V_1$, the condition is clearly fulfilled for v. If otherwise v is dominated by a vertex $w \in V'$ we have two possibilities: In the case that $w \in V_1$, v is still dominated by w in G_1, the CDS - condition is obviously fulfilled. In the case that $w \in V_2 \setminus V_1$, v is assigned color 3 by $\psi_{\restriction_{G_1}} \cdot \chi$, Finally, if v is colored with 3 by ψ, it will still be colored by 3 in G_1 by $\psi_{\restriction_{G_1}} \cdot \chi$.
We can conclude that $V' \cap V_1$ is a CDS in $(G_1, \psi_{\restriction_{G_1}} \cdot \psi)$. The argumentation in the case of $V' \cap V_2$ is similar.

For item *(ii)* take $V_1' \in \mathcal{S}_\Pi(G_1, \psi_1 \cdot \chi)$ and $V_2' \in \mathcal{S}_\Pi(G_2, \psi_2 \cdot \Phi\chi)$. Then $V_1' \cup V_2' \in \mathcal{S}_\Pi(G, \psi_1 \cdot \psi_2)$ holds. To see this, first note that $V_1' \cap S = V_2' \cap S$ due to the definition of $\cdot$ and the fact that $\Phi(1) = 1$. Secondly, observe that we only have to examine nodes $v \in S$. The case $v \in V_1' \cup V_2'$ is clear. Otherwise, by definition of $\Phi = \tau_{2,3}$, v is assigned color 2 in one of the fragments, say for example G_1. V_1' is a CDS on G_1, so v is either dominated by $w \in V_1'$, or assigned color 3 by ψ_1. In the first case, v remains dominated by w in G, in the latter case, v is assigned color 3 by $\psi_1 \cdot \psi_2$. With the definition of $c(\chi) = |\chi^{-1}(1)|$ we avoid counting of the nodes in $V' \cup S$ twice. Thus *(ii)* holds and we conclude that Constrained Dominatin Set is in fact dissectable.

Algorithmic consequences: Together with theorem 2.7 and equation (2.5), we obtain an algorithm for deciding Planar Edge Dominating Set with running times of $\mathcal{O}(n3^{6\sqrt{34k}})$, matching the tree decomposition based result in [4].

3.3 Planar Edge Dominating Set

Previous results: In [17] it was proven that EDGE DOMINATING SET is $\mathcal{NP}-$complete, even when restricted to planar graphs with a maximum node degree of 3. However, [5] states that there is an approximation scheme for the planar version.

Layerwise Separation Property: Let $G = (V, E)$ be a plane graph with layer decomposition $(L_i)_{1 \leq i \leq r}$ and $k \in \mathbb{N}$. Assume E' is an EDGE DOMINATING SET in G with $|E'| \leq k$. Define $S_i \subseteq L_i \cup L_{i+1}$ as the set of all endpoints of edges in E' for $i = 1, \ldots r$. Then S_i clearly separates layers L_{i-1} and L_{i+2}, for every edge going from layer i to layer $i-1$ has at least one endpoint in S_i. We conclude that EDS has the LSP of width 2. The size factor is $d = 4$, for each of the two endpoints of at most k edges is contained in two of separating sets.

Resulting algorithm: On page 56 it was already proven that EDGE DOMINATING SET is dissectable with 3 colors, thus according to theorem 2.7 we obtain a parameterized algorithm that correctly decides the problem. By equation (2.5) we see that its running time is bounded by $\mathcal{O}(n3^{4\sqrt{6k}})$. As far as we know, this is the best subexponential worst case bound for a parameterized algorithm for PLANAR EDGE DOMINATING SET.

3.4 Planar Minimum Maximal Matching

A *matching* on a graph G is an independent set of edges, that is, a set of edges such that no two are adjacent. A maximal matching is a matching where no edges can be added without losing this property. A minimum maximal matching is a maximal matching with minimum cardinality or, equivalently, a minimum independent edge dominating set.

As is pointed out in [17], the cardinality of a minimum maximal matching on G equals the size of a minimum edge dominating set, therefore the algorithm above also helps to decide the problem PLANAR MINIMUM MAXIMAL MATCHING.

3.5 Planar $s-t-$Partition

In [7], an algorithm for finding optimal $s-t-$partitions in time $\mathcal{O}(r^2n^3 2^{3r})$ on the class of $r-$outerplanar graphs is given. In fact, a weaker precondition than $r-$outerplanarity, namely $r-$*outerplane-separability* is already sufficient. Its ideas are based on the algorithm in [5], so the problem seems to be a candidate for an approach using the techniques presented in this work. However, as the upcoming proposition shows, $s-t-$PARTITION does not have the Layerwise Separation Property.

Proposition 9. $s-t-$PARTITION *does not have the Layerwise Separation Property for any* $s \in \mathbb{N}$.

Proof. We prove this assertion by showing that for every $s \in \mathbb{N}$, for every $w \in \mathbb{N}$ and for every $d \in \mathbb{N}$ there is a *yes*–instance $\langle G, k\rangle$ and an embedding ϕ of G, such that for every separation of layers $(S_j)_{1\leq j\leq r}$ with $S_j \subseteq \bigcup_{q=j}^{j+w-1} L_q$ for $j = 1, \ldots, r$ we have

$$\sum_{j=1}^{r} |S_j| > dk$$

Choose $s, w, d \in \mathbb{N}$ arbitrarily and consider the (square) grid graph G_p sketched in figure 3.5. Observe that (G_p, k) is a *yes*–instance for every k with $s+1 \leq k \leq p$ by splitting the node set along to the sketched line. Let $(L_i)_{1\leq i\leq r}$ be a layer decomposition of G_p. One can prove by induction on $r-i$ that the equation $|L_i| = 4+8(r-i)$ holds for $i = 1, \ldots, r$. Furthermore, it is easy to see that due to the structure of the graph, *any* separation S_i of layers L_{i-1} and L_{i+w} with $1 < i \leq r-w$ has at least size $|L_{i+w}|$. Together, this yields a lower bound for the size of any layerwise separation of width w of G_p:

$$\sum_{i=1}^{r} |S_i| \geq \sum_{i=2}^{r-w} |L_{i+w}| = \sum_{i=2}^{r-w} 4 + 8(r-i-w)$$

The right hand side is lower bounded by $r^2 - r(3+2w)$. By increasing p we increase r, and so, for fixed k, w, d and s we obtain *yes*–instances $\langle G_p, k\rangle$ in which any layerwise separation $(S_i)_{1\leq i\leq r}$ violates the size bound $\sum_{j=1}^{r} |S_j| \leq dk$, finishing this proof.

□

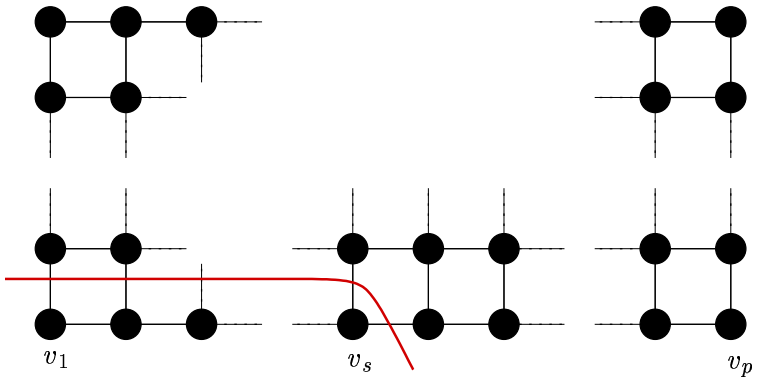

Figure 3.1: grid graphs G_p allow no small layerwise separation

Consequently, the concept of layerwise separations is not an adequate tool for extending the above mentioned result for $s-t-$PARTITION on $r-$outerplanar graphs to the class of general planar graphs.

3.6 Treewidth and fugitive search games

Another candidate for applying the presented concepts are fugitive search games on planar graphs (see e.g. [15]). General and, in retrospect, trivial considerations show that this approach must fail, at least in the case of inert fugitives with unbounded speed.

Due to the lack of connections to our subject, we avoid going into the details of search games and only use them an example to sketch a situation in which an attempt to prove the layerwise separation property is doomed to fail.

As already mentioned in section 1.3.2,the *treewidth* tw of a planar graph G is in relation with its outerplanarity by the following equation:

$$tw(G) \leq 3out(G) - 1$$

Using theorem 2.1, this result is extended in [4], yielding the following theorem:

Theorem 3.1. *Let (G, ϕ) be a plane graph that admits a layerwise separation of width w and size dk. Then, we have $tw(G) \leq 2\sqrt{6dk} + (3w - 1)$*

For a proof, see [4].

Without defininig what a *fugitive search game* or the *search number* precisely are, we state a theorem due to [15].

Theorem 3.2. *For an inert fugitive with unbounded speed, the monotone search number is equal to its treewidth plus 1.*

Assume that PLANAR SEARCH NUMBER, i.e. the decision problem that asks whether a planar graph G has search number less or equal to k, has the LSP of width w and size-factor d for some $w, d \in \mathbb{N}$. Chosing a big enough k, any graph G with $2\sqrt{6dk}+(3w-1)+1 < tw(G) < k$ implies a contradiction: the Layerwise Separation Property implies that G is a *no*– instance, for it does not allow for a layerwise separation according to theorem 3.1. On the other hand, theorem 3.2 implies that we are dealing with a *yes*–instance. Thus, PLANAR SEARCH NUMBER cannot have the Layerwise Separation Property.

It is clear that this situation occurs with all problems that can be characterized by the concept of treewidth in the above way.

Chapter 4

Conclusions and outlook

We presented a novel approach for developing fixed parameter algorithms for planar graph problems developed by Alber et. al. in [4]. In contrast to most of the existing fixed parameter results for $\mathcal{NP}-$complete problems, it yields algorithms with running times that are subexponential in the parameter's size. It relies on an algorithm that solves the concerning problem on subgraphs with bounded outerplanarity. For this task, we chose to use the well-known algorithm by Baker [5].

In adapting these algorithms to new problems, we found many similarities in both approaches. Consequently, we tried to distill the common ideas behind the underlying dynamic programming techniques. As a result, we defined the concept of *dissectability*, a special kind of optimal substructure property. We proved that it is a sufficient condition for the applicability of both algorithm's dynamic programming techniques. With its help, we easily derived subexponential fixed parameter algorithms for Planar Independent Set, Planar Dominating Set and Planar Edge Dominating Set. The result for Planar Dominating Set matches existing results, however, here they are obtained without employing the concept of tree decompositions. The result for Edge Dominating Set states, to our knowledge, a new worst case running time.

Furthermore, we have examined that the easy-to-prove concepts of Layerwise Separation Property and dissectability are quite restrictive. For two examples, $s-t-$Partition and Search Number, we proved that they do not have the LSP. The example of $s-t-$Partition together with typical exam-

ples of problems with the LSP suggest that this property might be restricted to problems in which solutions are in a way "dense" subgraphs. In fact, the LSP has to our knowledge only been proven for domination-like problems similar to the above. The same situation seems to occur for the concept of dissectability. Until now, the only known dissectable problems are the above mentioned.

However, as was pointed out in [10], there are more that 200 research papers published on solving domination-like problems on graphs, proving that we are dealing with a class that holds great interest.

Due to the restricted scope of this work, many related subjects can only be mentioned. We did, for example, not investigate the question of acutally constructing solutions. We did not address the important issue of the algorithms' space consumption. An interesting question is how the concepts of *glueability*, *weak glueability* (see [3]) and *dissectability* relate. Although dissectability is inspired by the above, it seems to be less restrictive. Further generalization of the results comes from [10]. Using tree decompositions, similar subexponential results are proven for $K_{3,3}-$minor-free or K_5-minor-free graphs. In [2], first encouraging empirical results are presented. However, the evaluated approach is based on tree decompositions and uses random graphs with low average outerplanarity.

Bibliography

[1] J. Alber, H. L. Bodlaender, H. Fernau, and R. Niedermeier. Fixed parameter algorithms for planar dominating set and related problems. In *7th Scandinavian Workshop on Algorithm Theory (SWAT)*, number 1851 in Lecture Notes in Computer Science, pages 97–110. Springer Verlag, 2000.

[2] J. Alber, F. Dorn, and R. Niedermeier. Empirical evaluation of a tree decomposition based algorithm for vertex cover on planar graphs. *Disc. Appl. Math.*, to appear.

[3] J. Alber, H. Fernau, and R. Niedermeier. Graph separators: a parameterized view. In *Proceedings of the 7th Annual International Computing and Combinatorics Conference (COCOON 2001)*, number 2108 in Lecture Notes in Computer Science, pages 318–327. Springer Verlag, 2001.

[4] J. Alber, H. Fernau, and R. Niedermeier. Parameterized complexity: Exponential speed-up for planar graph problems. Technical Report TR-01-023, Electronic Colloquium on Computational Complexity (ECCC), 2001.

[5] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *J. Assoc. Comput. Mach.*, 41(1):153–180, January 1994.

[6] H. L. Bodlaender. A partial $k-$arboretum of graphs with bounded treewith. *Th. Comp. Sci.*, 209:1–45, 1998.

[7] T. N. Bui and A. Peck. Partitioning planar graphs. *SIAM J. Comput.*, 21(2):203–215, April 1992.

[8] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using pq-trees. *J. Comput. Syst. Sci.*, 30(1):54–76, 1985.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition.* MIT Press/McGraw-Hill, 2001.

[10] E. Demaine, M. Hajiaghayi, and D. M. Thilikos. Exponential speedup of fixed parameter algorithms on k3,3-minor-free or k5-minor - free graphs. Technical Report MIT-LCS-TR-838, M.I.T, 2002.

[11] R. G. Downey and M. R. Fellows. *Parameterized Complexity.* Springer, 1999.

[12] R. G. Downey, M. R. Fellows, and U. Stege. Computational tractability: The view from mars. *Bulletin of the European Association for Theoretical Computer Science*, 69:73–97, 1999.

[13] M. R. Garey and D. S. Johnson. *Computers and Intractability – A Guide to the Theory of NP-Completeness.* Freeman, San Francisco, 1979.

[14] A. Kanevsky. Finding all minimum-size separating vertex sets in a graph. *Networks*, 23:533–541, 1993.

[15] D. M. Thilikos N. D. Dendris, L. M. Kirousis. Fugitive-search games on graphs and related parameters. *Th. Comp. Sci.*, 172:233–254, 1997.

[16] D. B. West. *Introduction to Graph Theory.* Prentice-Hall, 1996.

[17] M. Yannakakis and F. Gavril. Edge dominating sets in graphs. *SIAM J. Appl. Math*, 38(3):364–372, June 1980.