National and Kapodistrian University of Athens
School of Science, Department of Mathematics

Cyclability: Combinatorial Properties, Algorithms and Complexity

Spyridon Maniatis

PhD Thesis

Supervised by

Prof. Dimitrios M. Thilikos

June, 2018

ΠΕΡΙΛΗΨΗ

Ένα γράφημα G καλείται k-*κυκλώσιμο*, αν για κάθε k από τις κορυφές του υπάρχει ένας κύκλος στο G που τις περιέχει. Η *κυκλωσιμότητα* ενός γραφήματος G είναι ο μέγιστος ακέραιος k για τον οποίο το G είναι k-κυκλώσιμο και είναι μία παράμετρος που σχετίζεται με τη συνεκτικότητα. Σε αυτή τη διδακτορική διατριβή μελετάμε, κυρίως από τη σκοπιά της Παραμετρικής Πολυπλοκότητας, το πρόβλημα ΚΥΚΛΩΣΙΜΟΤΗΤΑ: Δεδομένου ενός γραφήματος $G = (V, E)$ και ενός μη αρνητικού ακεραίου k (η παράμετρος), να αποφασιστεί αν η κυκλωσιμότητα του G είναι ίση με k.

Το πρώτο μας αποτέλεσμα είναι αρνητικό και δείχνει ότι η ύπαρξη ενός FPT-αλγορίθμου για την επίλυση του προβλήματος ΚΥΚΛΩΣΙΜΟΤΗΤΑ είναι απίθανη (εκτός αν FPT = co-W[1], το οποίο θεωρείται απίθανο). Πιο συγκεκριμένα, αποδεικνύουμε ότι το πρόβλημα ΚΥΚΛΩΣΙΜΟΤΗΤΑ είναι co-W[1]-δύσκολο, ακόμα και αν περιορίσουμε την είσοδο στο να είναι χωριζόμενο γράφημα.

Από την άλλη, δίνουμε έναν FPT-αλγόριθμο για το ίδιο πρόβλημα περιορισμένο στην κλάση των επίπεδων γραφημάτων. Για να το πετύχουμε αυτό αποδεικνύουμε μια σειρά από συνδυαστικά αποτελέσματα σχετικά με την κυκλωσιμότητα και εφαρμόζουμε μια εκδοχή δύο βημάτων της περίφημης *τεχνικής της άσχετης κορυφής*, που εισήχθη από τους Robertson και Seymour στη σειρά εργασιών τους για Ελλάσονα Γραφήματα, ως ένα κρίσιμο συστατικό του αλγορίθμου τους για την επίλυση του προβλήματος των ΔΙΑΚΕΚΡΙΜΕΝΩΝ ΜΟΝΟΠΑΤΙΩΝ. Για να αποδείξουμε την ορθότητα του αλγορίθμου μας εισάγου-

με έννοιες, όπως αυτή των ζωτικών κυκλικών συνδέσμων, και αποδεικνύουμε αποτελέσματα με ανεξάρτητου γραφοθεωρητικού ενδιαφέροντος.

Κλείνουμε τη μελέτη μας με ένα δεύτερο αρνητικό αποτέλεσμα: Αποδεικνύουμε ότι για το πρόβλημα της ΚΥΚΛΩΣΙΜΟΤΗΤΑΣ δεν υπάρχουν πολυωνυμικοί πυρήνες, ακόμα και αν περιοριστούμε σε κυβικά επίπεδα γραφήματα, εκτός και αν δεν ισχύει μια υπόθεση της κλασσικής Θεωρίας Πολυπλοκότητας (ότι $\mathsf{NP} \subseteq \mathsf{co\text{-}NP/poly}$).

ABSTRACT

A graph G is called *k-cyclable*, if for every k of its vertices there exists a cycle in G that contains them. The *cyclability* of G is the maximum integer k for which G is k-cyclable and it is a connectivity related graph parameter. In this doctoral thesis we study, mainly from the Parameterized Complexity point of view, the Cyclability problem: Given a graph $G = (V, E)$ and an integer k (the parameter), decide whether the cyclability of G is equal to k.

Our first result is a negative one and shows that the existence of an FPT-algorithm for solving Cyclability is unlikely (unless FPT = co-W[1], which is considered unlikely). More specifically, we prove that Cyclability is co-W[1]-hard, even if we restrict the input to be a split graph.

On the other hand, we give an FPT-algorithm for the same problem when restricted to the class of planar graphs. To do this, we prove a series of combinatorial results regarding cyclability and apply a two-step version of the so called *irrelevant vertex technique*, which was introduced by Robertson and Seymour in their Graph Minors series ([83]) as a crucial ingredient for their algorithm solving the Disjoint Paths problem. To prove the correctness of our algorithm, we introduce notions, like the one of vital cyclic linkages, and give results of independent graph-theoretic interest.

We conclude our study with a negative result: We prove that Cyclability admits no polynomial kernel, even when restricted to cubic planar graphs, unless a classical complexity theoretic assumption (that NP $\subseteq$ co-NP/poly) fails.

ΠΡΟΛΟΓΟΣ

Σε αυτόν τον πρόλογο θα ήθελα να ευχαριστήσω κάποιους ανθρώπους χωρίς τη συμβολή των οποίων, η ολοκλήρωση αυτής της διαδακτορικής διατριβής θα ήταν αμφίβολη.

Πρώτα, θα ήθελα να ευχαριστήσω τον Καθηγητή Δημήτριο Μ. Θηλυκό, επιβλέ- ποντα καθηγητή των διδακτορικών σπουδών μου, για τις γνώσεις που μου μετέδωσε, την υπομονή, την κατανόηση, και τη σταθερή στήριξή του. Θεωρώ πως είναι ένας χαρισματικός δάσκαλος και, σημαντικότερα, ένας γενναιόδωρος άνθρωπος που προσφέρει πολύπλευρα στην Επιστήμη.

Ευχαριστώ ιδιαίτερα τα άλλα δύο μέλη της τριμελούς επιτροπής μου, τους Καθηγητές Σταύρο Γ. Κολλιόπουλο και Λευτέρη Μ. Κυρούση. Είχα την τύχη να γνωρίσω δύο ενδιαφέροντες ανθρώπους, με ευρείες επιστημονικές γνώσεις και έντονες κοινωνικές ευαισθησίες, όπως αρμόζει σε πανεπιστημιακούς δασκάλους.

Με τιμά ιδιαίτερα η συμμετοχή του Καθηγητή Χρήστου Α. Αθανασιάδη στην επταμελή επιτροπή της διατριβής μου. Τα μαθήματα Συνδυαστικής Θεωρίας που είχα την τύχη να παρακολουθήσω, κράτησαν αμείωτο το ενδιαφέρον μου και πιστεύω πως αποτελούν παράδειγμα της ομορφιάς και κομψότητας των Μαθηματικών. Ευχαριστώ επίσης τον Επίκουρο Καθηγητή Μιχάλη Χ. Δρακόπουλο και τον Καθηγητή Ευάγγελο Ράπτη, τόσο για τη συμμετοχή στην επταμελή επιτροπή αλλά και για την προσφορά τους στο Τμήμα Μαθηματικών, ιδιαίτερα όσον αφορά τη σύνδεση της Μαθηματικής επιστήμης με την επιστήμη των

Υπολογιστών. Τέλος, ευχαριστώ θερμά τον Αναπληρωτή Καθηγητή Ιωάννη Μούρτο, που με τιμά και αυτός με τη συμμετοχή του στην επιτροπή και, όντας μέλος του τμήματος Διοικητικής Επιστήμης και Τεχνολογίας του Οικονομικού Πανεπιστημίου Αθηνών, συμβάλλει στην επικοινωνία και συνεργασία των δύο τμημάτων.

Δεν θα μπορούσα να μην αναφερθώ στους συμφοιτητές μου, με τους οποίους, εκτός από τα πανεπιστημιακά έδρανα, μοιραστήκαμε σκέψεις, ανησυχίες και χαρές. Με αρκετούς αναπτύξαμε σχέσεις που ξεφεύγουν από τα πλαίσια των σπουδών και μοιραζόμαστε όμορφες αναμνήσεις. Ευχαριστώ ιδιαίτερα τους φίλους μου, Αρχοντία, Γιάννη και Δημήτρη γιατί τιμούν την ουσία της λέξης *φιλία*.

Ένα μεγάλο ευχαριστώ οφείλω και στους γονείς μου, Παναγιώτη και Ελευθερία, και στη σύντροφό μου Μαρία, για την αγάπη και τη στήριξη, υλική και ηθική, που πάντα προσφέρουν απλόχερα.

Θα ήθελα, τέλος, να αναφερθώ ιδιαίτερα στο Τμήμα Μαθηματικών του Ε.Κ.Π.Α., γιατί θεωρώ πως αποτελεί υπόδειγμα Δημόσιου Πανεπιστημιακού τμήματος. Παρά τις αντικειμενικές δυσκολίες, η ποιότητα των σπουδών είναι υψηλή και προσφέρεται αξιοκρατικά και χωρίς διακρίσεις. Ο χώρος του τμήματος Μαθηματικών αποτελεί εστία ελεύθερης σκέψης και δημιουργίας, που δεν περιορίζονται στη μελέτη της επιστήμης των Μαθηματικών. Με τιμά που αποτελώ κομμάτι του τμήματος και ελπίζω να συνεχίσει να προσφέρει, με τη συνεισφορά όλων των μελών του, το έργο του, με την ίδια υπευθυνότητα τόσο απέναντι στη Μαθηματική επιστήμη όσο και απέναντι στην κοινωνία.

Σκεφτόμαστε δεν σημαίνει βγαίνουμε από το σπήλαιο, ούτε ότι αντικαθιστούμε την αβεβαιότητα των σκιών με τα ευδιάκριτα περιγράμματα των ίδιων των πραγμάτων, το τρεμουλιαστό φέγγος γιας φλόγας με το φως του αληθινού ήλιου. Σημαίνει μπαίνουμε στον Λαβύρινθο, πιο συγκεκριμένα κάνουμε να φαίνεται και να είναι ένας Λαβύρινθος, ενώ θα μπορούσαμε να είχαμε μείνει «ξαπλωμένοι ανάμεσα στα λουλούδια ατενίζοντας τον ουρανό».

Κορνήλιος Καστοριάδης,
Τα σταυροδρόμια του λαβυρίνθου

CONTENTS

LIST OF FIGURES

CHAPTER 1 INTRODUCTION

1.1 Connectivity

In this first chapter, we give a brief outline of our work by discussing the main notions and problems involved and by presenting some past research. This chapter addresses the reader that is already familiar with some basic concepts of graph theory and parameterized complexity. The reader that is exposed to some of the notions for the first time is advised to quickly go through this introduction and return for a second read after having studied Chapter 2.

Undoubtedly, one of the most important pieces of information that someone would like to extract "easily" from a given graph G, is whether two specific vertices communicate, i.e., if two given are joined by a path in G. Fortunately, this problem which is widely known as

Reachability
Input: A graph $G = (V, E)$ and two distinct vertices $s, t \in V$.
Question: Is there a path in G with endpoints s and t?

can be solved "easily" (in polynomial time) using several well-known algorithms, such as BFS and DFS, which can be found in any introductory book about algorithms (for example see [24] and [61]). As a result, the problem of deciding whether any two vertices of a graph communicate, more precisely

> Connectivity
> *Input*: A graph $G = (V, E)$.
> *Question*: Is it true that for any two vertices in V, there is a path with these vertices as endpoints?

can also be solved in polynomial time by several algorithms. But why is Connectivity an important problem? Let us mention some examples:

- Suppose that $G = (V, E)$ represents a communication network N, where each node in V works both as a transmitter and as a receiver, and for any $v, u \in V$ it holds that $\{v, u\} \in E$ if and only if there is a communication channel between v and u in the network. Clearly, it is crucial to know whether a piece of information can be made available to every node, if it is initially announced to a single node and then is successively transmitted using the channels of the network. The answer to the previous question is exactly the answer for the Connectivity problem, when the given graph is the one that represents the communication network N.
- Imagine the previous example, where now the elements of E represents a tube and each node in V represents a tank. We start by pumping water into a tank. When a tank is full, it starts channeling the water that it receives to all other tanks connected with it by a tube. The flow of water stops when there is some tank that is full and all its neighbouring tanks (connected with it via a tube) are also full. Can we know if all tanks are full at the time when the flow stops? It is not hard to confirm that we can, if we know whether the underlying graph of our tank/tube system is connected.
- Suppose that we have the communication network of the first example but we additionally know that a node (can be any member of V) is corrupted, meaning that it receives messages but it does not transmit anything to its neighbours. Can we be sure that a message, initially transmitted to one of the non-corrupted nodes, will be received by all the nodes of the network? This question is equivalent to asking whether

the underlying graph of our communication networks is 2-connected (we will formally define k-connectivity in Chapter 2).

In the opening paragraph of his book *Extremal Graph Theory*, Béla Bollobás notes: "*Perhaps the most basic property a graph may posses is that of being connected. At a more refined level, there are various functions that may be said to measure the connectedness of a connected graph.*"
As we have already tried to illustrate with the previous three examples, it seems that connectivity is a really important property for a graph. This is why researchers have been studying variants of connectivity or other properties that seem relevant to it. From the viewpoint of combinatorics there are many known results but the algorithmic properties of most connectivity measures remain fairly unexplored. In this thesis we study, mainly from an algorithmic point of view, a connectivity related graph parameter, namely *cyclability*.

Cyclability. For a positive integer k, a graph G is k-*cyclable* if every k vertices of G lie on a common cycle; we assume that any graph is 1-cyclable trivially. The *cyclability* of a graph G, introduced by Chvátal in [17], is the maximum integer k for which G is k-cyclable. Clearly, a graph G is Hamiltonian if and only if its cyclability equals $|V(G)|$. Therefore, we can think of cyclability as a quantitive measure of Hamiltonicity or a tuning parameter between connectivity and Hamiltonicity. Cyclability is a well studied parameter in the graph theory literature. We give some references:

- Dirac proved that the cyclability of a k-connected graph is at least $k,$ for $k \geq 2$ [30].
- Watkins and Mesner ([92]) characterized the extremal graphs for the theorem of Dirac.
- There is a variant of cyclability restricted only to a set of vertices of a graph. Generalizing the theorem of Dirac, Flandrin et al. ([40]) proved that if a set of vertices S in a graph G is k-connected, then there is a cycle in G through any k vertices of S. (A set of vertices S is k-connected in G if a pair of vertices in S cannot be separated by removing at most $k-1$ vertices of G.)

- Another avenue of research is lower-bounds on cyclability of graphs in restricted families
  - Every k-connected and $K_{1,4}$-free graph has cyclability at least $2k$ ([41]).
  - Every 3-connected claw-free graph has cyclability at least 6 ([77]).
  - Every 3-connected planar graph has cyclability at least 23 ([4]).
  - Every 3-connected and cubic planar graph has cyclability at least 9 ([54]) and the bound is tight (consider, for example, the Petersen graph).
- A graph G is *hypohamiltonian* if it is not Hamiltonian but all graphs obtained from G by deleting one vertex are. Clearly, a graph G is hypohamiltonian if and only if its cyclability equals $|V(G)| - 1$. Hypohamiltonian graphs appear in combinatorial optimization and are used to define facets of the traveling salesman polytope [52]. Curiously, the computational complexity of deciding whether a graph is hypohamiltonian seems to be open.

Although cyclability has been extensively studied as a graph parameter, it has not been studied (to our knowledge) algorithmically so far: There are no known (non-trivial) algorithms for computing the cyclability of a given graph and there are no results about the computational complexity of Cyclability. In this thesis we initiate this study. For this, we consider the following problem.

Cyclability
Input: A graph G and a non-negative integer k.
Question: Is every k-vertex set S in G cyclable, i.e., is there a cycle C in G such that $S \subseteq V(C)$?

We postpone the formal description of our results until subsection 1.3 and proceed with a brief discussion of the ideas and results of the Graph Minors series, focusing on the techniques that are of great importance for our study.

1.2 Graph Minors

In this subsection we discuss some parts of, probably, the most influential bodies of work in modern Combinatorics, the *Graph Minor series* of Neil Robertson and Paul Seymour. In their work, which is comprised of 23 papers published between 1983 and 2011, they managed to prove the *Wagner*'s *conjecture* (nowadays known as the *Robertson-Seymour theorem*).

More specifically, they proved that the class of undirected graphs partially ordered by the minor relationship forms a well-quasi-ordering (for every infinite sequence of graphs there exists two such that on is a minor of the other). An immediate consequence of this, which has important algorithmic applications, is that every graph family closed under minors can be characterized by a set of forbidden minors, the *obstruction set* (a graph G belongs to the class if and only if it contains no member of the obstruction set as a minor).

Unfortunately, their proof is not constructive (and was proved later that no constructive proof exists [39]) and therefore we cannot hope for an algorithm that, given a minor-closed family of graphs produces the corresponding obstruction set. On the positive side, Robertson and Seymour also gave a polynomial time algorithm for checking whether a fixed graph H (whose size is considered as a parameter) is a minor of a given graph G, or more specifically they proved that for every fixed graph H there is an $O(n^3)$ time algorithm for solving the following problem

H-Minor Containment
Input: A graph G.
Question: Is some minor of G isomorphic to H?

Looking at the problem from the parameterized complexity point of view we can rephrase their result: *The problem of checking whether a graph H is contained as a minor in a graph G is in* FPT*, when parameterized by the size of H.*

Although the systematic study of Parameterized Complexity theory did not start until the early 90s, the results of the Graph Minors series already contained fruitful ideas, algorithms, and important results which have been revisited over and over and today are part of every book on computers and algorithms. In fact, it can be argued that the notions and ideas introduced in the series played an equally crucial role to the development of the theory as the results themselves.

The Robertson-Seymour theorem combined with the FPT-algorithm for the H-Minor Containment problem provide a "recipe" for designing FPT-algorithms for deciding minor-closed properties for graphs:

Given a minor-closed graph property π and a graph G, explicitly compute the corresponding obstruction set $\mathcal{O}(\pi)$ and check for every $H \in \mathcal{O}(\pi)$ whether it is contained as a minor in G. If all the answers are negative, then graph G satisfies property π, otherwise it does not.

Actually, the same strategy can also work for immersion-closed graph classes as a result of the following two theorems from Robertson and Seymour (in the last paper of the Graph Minors series which resolves a conjecture of Nash Williams), and Gröhe, Kawarabayasi, Marx, and Wollan. Although we will not concern ourselves with immersions in this thesis, we mention the results:

Theorem 1.2.1 (Robertson and Seymour [84]). *The class of all finite graphs is well-quasi-ordered by the immersion relation.*

Theorem 1.2.2 (Gröhe, Kawarabayasi, Marx, and Wollan [50]). *The problem of checking whether a graph H is contained as an immersion (or topological minor) in a graph G is in FPT when parameterized by the size of H.*

We will focus on a specific technique, the so called *irrelevant vertex technique*, that was introduced in the Graph Minors series as a crucial component for studying the fundamental problem of deciding whether a given graph contains vertex-disjoint paths with certain (given as an input) endpoints. More precisely, Robertson and Seymour studied the following problem

Disjoint Paths
Input: A graph G and pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices of G
Question: Do there exist paths $P_1, \ldots, P_k$ in G that are mutually vertex disjoint and such that P_i joins s_i and t_i, $i \in \{1, \ldots, k\}$?

The pairs $(s_1, t_1), \ldots, (s_k, t_k)$ of vertices that we want to link are also called *terminals*. It is known that Disjoint Paths is NP-complete, along with its edge-disjoint and directed variants, even when restricted to the class of planar graphs ([66, 69, 91]).

Robertson and Seymour gave an algorithm that solves the Disjoint Paths problem in $f(k)\cdot n^3$ steps (later the dependance on n was improved to quadratic in [57]), which means that the problem is fixed parameter tractable when parameterized by the number, k, of the requested paths.

Although they dealt with the problem in the 13th paper of the series ([80]), the proof was completed only after the 22th paper ([83]) which was published in 2012. The, crucial, missing part was the analysis of the so called *irrelevant-vertex technique*, which has be widely used ever since for studying various combinatorial problems (see for example [25], [26], [49], [56], [58], and [59]).

The Robertson-Seymour Algorithm. The algorithm of Robertson and Seymour for solving the Disjoint Paths problem, strongly relies on the irrelevant vertex technique. We demonstrate this by giving an outline of the algorithm. Clearly, any graph either has bounded (by some function of k) treewidth or it has "large" treewidth.

Case 1. The treewidth of G is bounded. In this case there are standard dynamic programming arguments for solving the problem, given a tree decomposition of bounded width.

Case 2. The treewidth of G is large.

Subcase 1. There exists a large clique minor in G. If there exist disjoint paths from the terminals to the clique minor, then we can exploit the fact that any two vertices of a clique are connected and link up the terminals in any way we want.
If this is not the case, then the clique minor is, in a sense, cut off from the terminals, and in it can be proved that some vertex v of the clique minor is *irrelevant*, meaning that there exists a solution for the problem in G if and only if there exists a solution in $G \setminus \{v\}$.

Subcase 2. There is no large clique minor in G. It can be proved that, after deleting a bounded number vertices, we end up with a large subgraph, that can be considered flat, of large treewidth. This means that the subgraph contains a large grid minor or a subgraph that spreads in two dimensions, called a *wall*. It can be proved that there exists a "central" vertex of the wall that is *irrelevant*.

Having analysed all the cases, the algorithm is simple: Iteratively delete irrelevant vertices (either due to Subcase 1 or to Subcase 2) until the reduced graph has bounded treewidth, at which point Case 1 applies. For a visualization of the operation of the Robertson-Seymour algorithm see Figure 1.1.

However, it is the last subcase which requires almost all the main results of the Graph Minors series for its analysis. The use of such "heavy" structure theorems, results to the algorithm having an immense running time (due to the constants hidden in $f(k)$), making it practically inapplicable to actual problems. Another proof that is much shorter and bypasses the involved graph structure theorem of [81] was given by Kawarabayashi and Wollan in [60], where they managed to prove an upper bound with $f(k)$ being of magnitude $2^{2^{2^{2^{\Omega(k)}}}}$. Unfortunately, the dependence on the parameter is still huge and renders the algorithm inefficient even for small values of k.

Another route is to try to obtain similar results for restricted graph classes. A decisive step to this direction was made by Adler, Kolliopoulos, Krause, Lokshtanov, Saurabh, and Thilikos in [3], where they proved that $f(k)$ can be just single exponential on k when the input is restricted to be a planar graph. Note that planar graphs already exclude large clique minors (they exclude K_n for any $n \geq 5$) and the task is to prove that any, large enough, grid minor contains an irrelevant vertex. As we have already pointed out, this is not an easy task (it is one of the most deep results in the Graph Minor series) but the structure of planar graphs proves to be helpful.

1.3 The results of this thesis

We are now ready to talk about the main contributions of this thesis. We will present and discuss our results one by one and give an overview of the underlying ideas and techniques. Some familiarity with classical and parameterized complexity is assumed (the reader can always return to this section after studying Chapter 2, where all necessary definitions are given).

From the classical complexity point of view, determining the cyclability of a graph is a computationally hard problem as it is easy to see that Cyclability with $k = |V(G)|$ is Hamiltonicity and Hamiltonicity is NP-complete even for planar cubic graphs ([46]). Hence, we have the following.

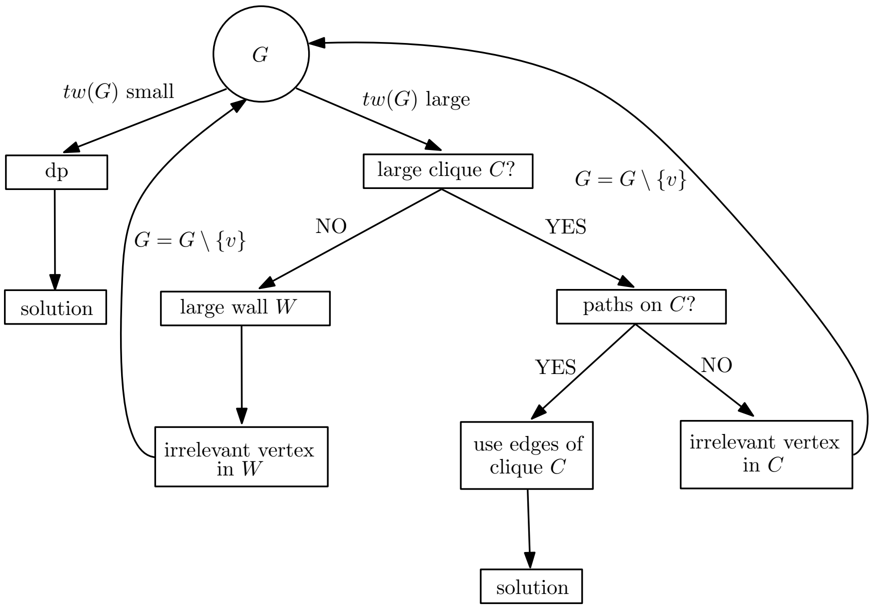


Figure 1.1: Observe that the solution can be obtained in two ways: Either by deploying a dynamic programming routine on a graph with small treewidth or by exploiting the fact that the paths intersect a big clique (described in Subcase 1). The algorithm also iterates in two ways: Either after deleting an irrelevant vertex located in a big clique (Subcase 1) or by deleting an irrelevant vertex in an "almost flat" subgraph (Subcase 2).

Proposition 1.3.1. Cyclability *is* NP*-hard for cubic planar graphs.*

As cyclability can be thought of as a tuning parameter between connectivity and hamiltonicity, it is interesting to study it from a parameterized complexity point of view. In this thesis we treat the cyclability k of the input graph as a parameter and when we mention Cyclability problem we will distinguish between the standard problem and the parameterized one, unless it is unclear from the context. Before stating our results we briefly go through some notions regarding parameterized complexity.

Parameterized complexity. A *parameterized problem* is a language $L \subseteq \Sigma^* \times \mathbb{N}$, where Σ is a finite alphabet. The instances of a parameterized problem are pairs (I, k), where $I \subseteq \Sigma^*$ is the main part and $k \in \mathbb{N}$ is the parameterized part. Parameterized Complexity settles the question of whether a parameterized problem is solvable by an algorithm (we call it FPT-*algorithm*) of time complexity $f(k) \cdot |I|^{O(1)}$ where $f(k)$ is a function that does not depend on $|I|$. If such an algorithm exists, we say that the parameterized problem belongs to the class FPT.

In a series of fundamental papers (see [33, 32, 34, 35]), Downey and Fellows defined a series of complexity classes, such as W$[1] \subseteq$ W$[2] \subseteq \cdots \subseteq$ W$[SAT] \subseteq$ W$[P] \subseteq$ XP and proposed special types of reductions such that hardness for some of the above classes makes it rather impossible that a problem belongs to FPT (we stress that FPT $\subseteq$ W$[1]$). We mention that XP is the class of parameterized problems such that there is an algorithm that solves them in time $O(|I|^{f(k)})$, for some function f (that does not depend on $|I|$) and every k.

More notions of Parameterized Complexity are discussed in Section 2.2 of Chapter 2. We also refer the reader to [22] (see also [31], [42], and [73]).

Algorithm for Cyclability**.** In this thesis we deal with the parameterized complexity of Cyclability when parameterized by k. It is easy to see that Cyclability is in XP:

For a graph G, we can check all possible $\binom{n}{k}$ subsets X of $V(G)$ of size k. For each subset X, we consider $k!$ orderings of its vertices, and for each sequence of k vertices $x_1, \ldots, x_k$ of X, we use the main algorithmic result of Robertson and Seymour in [80], to check whether there are k disjoint paths that join x_{i-1} and x_i for $i \in \{1, \ldots, k\}$ assuming that $x_0 = x_k$. We return a yes-answer if and only if we can obtain the required disjoint paths for each set X, for some ordering.

Thus, the running time of the previous algorithm is $\binom{n}{k} \cdot k! \cdot t(n)$, where $t(n)$ is the running time for the Robertson-Seymour algorithm on a n-vertex graph. This gives an $O(f(k) \cdot n^{k+3})$ algorithm for solving Cyclability. This algorithm is clearly inefficient: Not only the value of the parameter appears as an exponent of n, but also the dependance of f on k is huge.

The first attempt to brute force towards a solution is not very successful.

Can we try something more sophisticated? Is it possible that Cyclability is FPT when parameterized by k?

In this thesis we investigate the parameterized complexity of the Cyclability problem. In the remaining of this Chapter we present our results along with some brief discussion for each of them.

1.3.1 Hardness

The first thing one usually tries is either to design an FPT algorithm for the problem of interest or demonstrate that the existence of such an algorithm is unlikely. Our first result is that an FPT-algorithm for Cyclability is rather unlikely as the problem is co-W[1]-hard even when restricted to split graphs, where a split graph is any graph G whose vertex set can be partitioned into two sets A and B such that $G[A]$ is a complete graph and $G[B]$ is an edgeless graph. Specifically the following theorem is proved in Chapter 6:

Theorem 1.3.1. *It is* W[1]*-hard to decide for a split graph G and a positive integer k, whether G has k vertices such that there is no cycle in G that contains these k vertices, when the problem is parameterized by k.*

This theorem states that the complementary problem of Cyclability, i.e., the problem of deciding if a given graph G contains k vertices such that no cycle in G contains them all, is hard (unless FPT = W[1]) when parameterized by k. In fact it is hard even for a class of graphs with relatively simple structure, the class of split graphs (graphs that can be partitioned into a clique and an independent set). In other words, the problem of refuting that a graph is k-cyclable is hard when parameterized by k. This of course implies that the problem of interest, Cyclability, is co-W[1]-hard even for split graphs, when parameterized by k.

The proof (which is presented in detail in Chapter 6) is a (parameterized) reduction of the natural parameterization of the Clique problem (given a graph G and a positive integer k decide whether G contains a clique of size k) to the complement of Cyclability.

This result suggests that (efficiently) solving the Cyclability problem in its full generality is unlikely. Naturally, we focused our attention on a graph class that is important and yet is "away" from the class of split graphs: The class of *planar graphs*, i.e., graphs that can be drawn in the plane in such a way that any two edges do not intersect except on a common endpoint.

It is well-known that any planar graph excludes K_t as a minor for every $t \geq 4$, thus if a split graph is planar then its complete part should have at most 4 vertices. Moreover, any planar graph excludes $K_{t,t}$ as a minor for every $t \geq 3$ and thus a split graph that is planar can only be large if its edgeless part is large. These arguments should be enough to justify that the class of planar graphs and the class of split graphs do not share many important properties.

Additionally, planar graphs is one of the most studied graph classes in Graph Theory, both from the combinatorial and the algorithmic point of view. Reflecting their importance, researchers have created a very rich toolbox along the years, which can be employed when trying to tackle problems on planar graphs. Almost in every book on Graph Theory, such as [29, 12] and [11], there is at least one chapter devoted on planar graphs. There are even entire books devoted on algorithms on planar graphs (see [74]) or studying the more general subject of graphs embeddable on surfaces (see for example [51, 88, 48]).

1.3.2 FPT for planar graphs

The positive result of this thesis is proving that the Cyclability problem is fixed-parameter tractable when the input is restricted to be a planar graph.

Theorem 1.3.2. *The* Cyclability *problem, when parameterized by k, is in* FPT *when its input graphs are restricted to be planar. Moreover, the corresponding* FPT*-algorithm runs in $2^{2^{O(k^2 \log k)}} \cdot n^2$ steps.*

Actually, our algorithm solves as slightly more general problem, where the input comes with a subset R of annotated vertices and the question is whether every k-vertex subset of R is cyclable. More specifically, we give an FPT-algorithm for the following problem:

Planar Annotated Cyclability
Input: A graph planar G, a set $R \subseteq V(G)$, and a non-negative integer k.
Question: Does there exist, for every set S of k vertices in R, a cycle C of G such that $S \subseteq V(C)$?

Of course, by setting R to be equal with $V(G)$ we get an instance of the Cyclability problem with the input graph being planar.

Outline of the algorithm. The two key ingredients in the proof of Theorem 1.3.2 are a new, two-step, version of the *irrelevant vertex technique* and a new combinatorial concept of *cyclic linkages* along with a strong notion of "cyclical" *vitality* on them (vital linkages played an important role in the Graph Minors series, in [82] and [83]). The proof of Theorem 1.3.2 is presented in Chapter 4. Next, we give a rough sketch of our method.

We work with a variant of Cyclability in which some vertices (initially all) are coloured. We only require that every k coloured vertices lie on a common cycle. If the treewidth of the input graph G is "small" (bounded by an appropriate function of k), we employ a dynamic programming routine (presented in detail in Chapter 4) to solve the problem.

Otherwise, there exists a cycle in a plane embedding of G such that the graph H in the interior of that cycle is "bidimensional" (contains a large subdivided wall) but is still of bounded treewidth. This structure permits to distinguish in H a sequence $\mathcal{C}$ of, sufficiently many, concentric cycles that are all traversed by some, sufficiently many, paths of H.

Our first goal is to check whether the distribution of the coloured vertices in these cycles yields some "big uncoloured area" of H. In this case we declare some "central" vertex of this area *problem-irrelevant* in the sense that its removal creates an equivalent instance of the problem.

If such an area does not exists, then R is "uniformly" distributed inside the cycle sequence $\mathcal{C}$. Our next step is to set up a sequence of instances of the problem, each corresponding to the graph "cropped" by the interior of the cycles of $\mathcal{C}$, where all vertices of a sufficiently big "annulus" in it are now uncoloured.

As the graphs of these instances are subgraphs of H and therefore have bounded treewidth, we can get an answer for all of them by performing a sequence of dynamic programming (using the algorithm we present in Chapter 4) calls, each taking a linear number of steps. At this point, we prove that if one of these instances is a no-instance then initial instance is a no-instance, so we just report it and stop.

Otherwise, we pick a coloured vertex inside the most "central" cycle of $\mathcal{C}$ and prove that this vertex is *colour-irrelevant*, i.e., an equivalent instance is created when this vertex is not any more coloured.

In any case, the algorithm produces either a solution or some "simpler" equivalent instance that either contains a vertex less or a coloured vertex less.

This permits a linear number of recursive calls of the same procedure. For a visualization of the described procedure see Figure 1.2

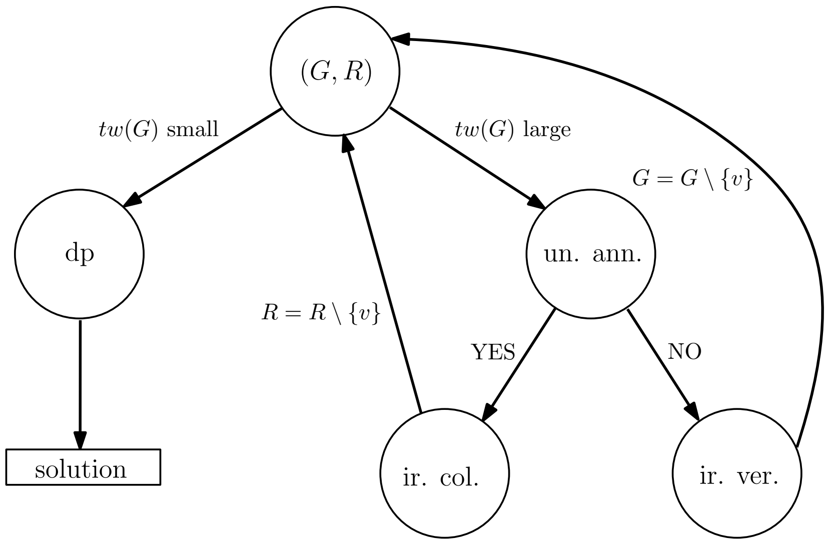


Figure 1.2: Given (G, R) as the input (upper circle), if the **tw**(G) is small then we solve by using dynamic programming (mid-left circle). If **tw**(G) is large and the annotated vertices are distributed uniformly (mid-right circle), then we find an irrelevant vertex v (bottom-right), delete it, update the input to $(G \setminus \{v\}, R)$ and iterate. Else (if the annotation is not uniform), find a vertex v whose colour is irrelevant, uncolour it, update the input to $(G, R \setminus \{v\})$ and iterate.

1.3.3 Combinatorial results

To prove the existence of irrelevant vertices when the treewidth of the input graph is large, we have to introduce several combinatorial tools. One of them is the notion of *strongly vital* linkages, a variant of the notion of vital linkages introduced in [82], which we apply to terminals traversed by cycles instead of terminals linked by paths, as it has been done in [82]. This notion of "cyclical" vitality permits a significant restriction of the expansion of cycles which certify that sets of k vertices are cyclable and is able to justify both critical steps of our algorithm. The proofs of the combinatorial results that support our algorithm are presented in Chapter 3 and we believe that they are of independent combinatorial importance. To give a brief overview we introduce the notion of *graph linkages* (formal definitions are given in 3).

Graph Linkages. A *graph linkage* of a graph G is a pair $\mathcal{L} = (H, T)$ such that H is a subgraph of G and T is a subset of the vertices of H, called *terminals* of $\mathcal{L}$, such that every vertex of H with degree different than 2 is contained in T. The set $\mathcal{P}(\mathcal{L})$, the *path set of* the graph linkage $\mathcal{L}$, contains all paths of H whose endpoints are in T and do not have any other vertex in T.

The *pattern* of L is the graph

$$\big(T, \big\{\{s,t\} \mid \mathcal{P}(\mathcal{L}) \text{ contains a path from } s \text{ to } t \text{ in } H\big\}\big).$$

Two graph linkages of G are *equivalent* if they have the same pattern and are *isomorphic* if their patterns are isomorphic.

A graph linkage $\mathcal{L} = (H, T)$ is called *strongly vital* in G if $V(H) = V(G)$ and there is no isomorphic graph linkage in G that is different from $\mathcal{L}$. We call a graph linkage $\mathcal{L} = (H, T)$ *linkage* (resp. *cyclic linkage*) if its pattern is a collection of paths (resp. a single cycle). We sometimes denote such a linkage just by writing $\mathcal{L}$.

We say that a linkage $\mathcal{L}$ in a graph G is *unique* if for all linkages $\mathcal{L}'$ in G equivalent to $\mathcal{L}$, we have that $V(\mathcal{L}) = V(\mathcal{L}')$.

The main result of Graph Minors XXI [82] is the following structural theorem:

Theorem 1.3.3 (The Unique Linkage Theorem [82]). *For all $k \geq 1$, there exists a value $w(k)$ such that the following holds. Let $\mathcal{L}$ be a linkage of G with $|\mathcal{P}(\mathcal{L})| = k$ and $V(G) = V(\mathcal{L})$. If $\mathcal{L}$ is unique, then the treewidth of G is at most $w(k)$.*

As we have already mentioned, the dependance of the value $w(k)$ on k is immense, as the proof in [82] needs the full power of the graph minor structure theorem. It was substantially improved by Kawarabayashi and Wollan in [60] (the dependance on k becomes triple exponential) and their proof is also much shorter.

Adler, Kolliopoulos, Krause, Lokshtanov, Saurabh, and Thilikos gave an improvement for the class of planar graphs [3]. In their work $w(k) = O(k^{3/2} \cdot 2^k)$ which is radically better than the bounds known for general graphs.

Our contribution towards this direction, concerning *cyclic linkages*, is the following

Theorem 1.3.4. *If a planar graph G contains a strongly vital cyclic linkage $\mathcal{L} = (C, T)$, then* $\mathbf{tw}(G) = O(|T|^{3/2})$.

By thinking of the strong vitality property of a graph linkage as uniqueness (for more details see chapter 3), we can observe something interesting: The dependence on the number of the terminals (that corresponds to k) becomes polynomial (almost linear) when the pattern of the graph linkage is a cycle. This can lead to improved running times for algorithms on problems where cyclic linkages can be used.

Unfortunately, this is not the case for Cyclability but the reason is not this bound. The dynamic programming routine (which we present in chapter 4) causes the double exponential dependence on k which, we do not believe that can be substantially improved (see the discussion in the conclusion, Chapter 8).

1.3.4 No polynomial kernels

The last result is another negative one. It states that it is unlikely, even for the case of cubic planar graphs, that Cyclability admits any polynomial kernel. Before presenting the result we have to talk about *kernelization* (for a much more detailed introduction we refer the reader to Section 2.2 of Chapter 2).

Kernelization. The notion of *kernelization*, which has been proposed as a formalization of the idea of preprocessing, has recently grown to be a separate research area in the field of Parameterized Complexity. The main idea is that, before trying to solve a problem on a given input, we can try to reduce the input to a smaller one by taking rid of parts that are not relevant for the problem. This idea, in the framework of Parameterized Complexity, has evolved to the following definition:

Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A *kernelization* (or *kernel*) for a parameterized problem L is an algorithm that given an instance (I, k) outputs, in time polynomial in $|I|$ and k, an instance (I', k') such that

- $(I, k) \in L$ if and only if $(I', k') \in L$,
- $|I'|$ is bounded by a computable function f in k and k' is bounded by a computable function g in k.

The output (I', k') of the kernelization is called a *kernel* and the function f is the size of the kernel. We say that a kernel is *polynomial* if f is a polynomial function.

A somewhat surprising result, is that a parameterized problem is in FPT if and only if it admits a kernelization algorithm (see Section 2.2 of Chapter 2). However, when we actually want to implement a preprocessing algorithm we usually need the polynomial, or even better linear, kernels.

In Chapter 7, we prove that this is unlikely for Cyclability, even for cubic planar graphs (a graph G is called cubic if $\deg_G(v) = 3$ for every $v \in V(G)$), unless some widely believed complexity theoretic assumption fails:

Theorem 1.3.5. Cyclability*, parameterized by* k*, admits no polynomial kernel unless* $\mathsf{NP} \subseteq \mathsf{co\text{-}NP/poly}$*, even when restricted to cubic planar graphs.*

The above result indicates that the Cyclability does not follow the kernelization behaviour of many other problems (see, e.g., [9]) for which surface embeddability enables the construction of polynomial kernels. For the proof we use the cross-composition technique introduced by Bodlaender, Jansen, and Kratsch in [10], and specifically we show that the NP-hard Hamiltonicity with a Given Edge problem AND-cross-composes to Cyclability.

CHAPTER 2
BASIC DEFINITIONS AND PRELIMINARY RESULTS

In this section we give some basic definitions regarding Complexity Theory, some basic notions about Graph Theory as well as some preliminary results to be used in latter sections. Any definition or lemma that is only used at a particular section is presented in the corresponding section.

Let $n \in \mathbb{N}$ and let X be a set. We denote by $\mathcal{P}(X)$ the powerset of X, i.e. $\mathcal{P}(X)$ is the set that contains all the subsets of X and for every $k \leq n$ we say that $Y \subseteq X$ is a k-*subset* of X if $|Y| = k$. We denote by $X^{\leq k}$ all the i-subsets of X, for every $i \in \{1, 2, \ldots, k\}$.

2.1 Graphs

2.1.1 Basic notions about graphs

Graphs. An undirected graph, usually denoted by G, is an ordered pair composed by a finite set $V(G)$ and a set of 2-subsets, $E(G)$, of $V(G)$. The elements of $V(G)$ are called the *vertices* of the graph and the elements of $E(G)$ its *edges*. Sometimes, when it is clear to which graph we refer to, we will write V and E without denoting the corresponding graph. For an edge $e = \{u, v\} \in E(G)$, we say that the vertices u and v are the *endpoints* of e. Given a graph G

we will also assume that it is undirected, unless stated otherwise. We denote by $\mathcal{G}$ the class of all undirected graphs.

We say that two graphs, G and H, are *isomorphic* if there exists a bijection $f : V(G) \to V(H)$ such that $\{u, v\} \in E(G)$ if and only if $f(u), f(v) \in E(H)$. We call such a function an *isomorphism* between G and H and we think of these two graphs as the same graph.

The usual way to visualise a graph G is to depict each of its vertices as a dot in the plane and connect two dots with a line if the corresponding vertices are the endpoints of an edge in $E(G)$. Such a visualisation is called a *drawing* of graph G and, obviously, it is not unique $\big($for an example see the two drawings of graph $H = \big(\{x, y, z, u, v\}, \big\{\{x, y\}, \{x, z\}, \{x, u\}, \{y, z\}, \{y, u\}, \{y, v\}, \{z, v\}, \{u, v\}\big\}\big)$ in Figure 2.1 $\big)$.

For every vertex $v \in V(G)$, *the neighbourhood* of v in G, denoted by $N_G(v)$, is the subset of vertices that are adjacent to v (v is not included), i.e. $N_G(v) = \{u \in V(G) \mid \{v, u\} \in E(G)\}$, and its size is called the *degree* of v in G, denoted by $\text{deg}_G(v)$. The *maximum* (respectively *minimum*) *degree* $\Delta(G)$ (respectively $\delta(G)$) of a graph G is the maximum (respectively minimum) value taken by $deg_G(v)$ over $v \in V(G)$.

For any set $U \subseteq V(G)$ we define $N_G(U) = \bigcup_{u \in U} N_G(u) \setminus U$. For every integer n, we denote by K_n the k*-clique* which is the graph on n vertices which contains all possible $\binom{k}{2}$ edges. Let $S \subseteq V(G)$ be a subset of the vertices of graph G. We define the *subgraph of* G *induced by* S as $G[S] = \big(S, \big\{\{u, v\} \in E(G) \mid u, v \in S\big\}\big)$ and we define the *boundary* of S, denoted by ∂S, to be the set of all vertices $v \in S$ such that there exists some edge $\{v, u\}$ with $u \in G \setminus S$.

For a subset $I \subseteq V(G)$ we say that I forms an *independent set* in G if the induced subgraph $G[I]$ is edgeless, i.e. $E\big(G[I]\big) = \emptyset$.

We also denote by $E_G(v)$ all the edges of G that have v as an endpoint (or are *adjacent* to v), i.e. $E_G(v) = \{e \in E(G) \mid e \cap v \neq \emptyset\}$.

Paths and cycles. A *path* $P = (V, E)$ is a non-empty graph where, for some $k \in \{1, 2, 3 \ldots\}$

$$V = \{v_1, v_2, \ldots, v_k\} \text{ and } E = \big\{\{v_1, v_2\}, \{v_2, v_3\}, \ldots, \{v_{k-1}, v_k\}\big\}$$

and v_i are all distinct. The vertices v_1 and v_k are called the *endpoints* of P and we say that P *links* or *connects* v_1 and v_k. The vertices of P that are not

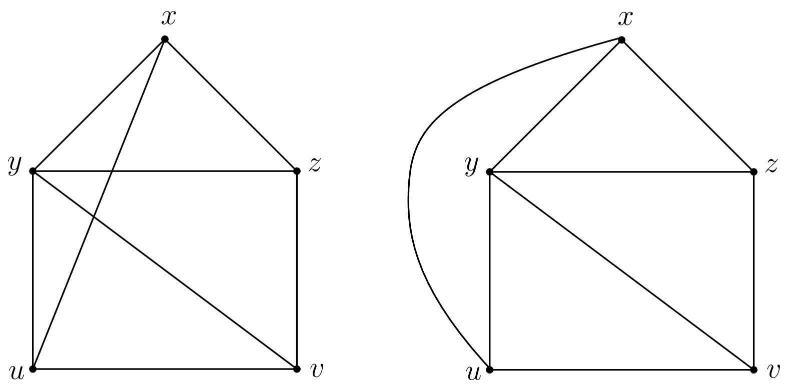


Figure 2.1: Two different drawings of graph H. Observe that at the right one, the intersection of two edges is always a vertex.

endpoints are the *inner* vertices of the path. The *length* of a path equal to the number of its edges (we allow paths of length 0 which are just a single vertex). The *distance* of vertices u and v in G, denoted by $\text{dist}_G(u, v)$, is the minimum length of a path in G which links v and u.

By $v_1 \dots v_p$ we denote the path comprised of the vertices $v_1, \dots, v_p$ and the edges $\{v_1, v_2\}, \dots, \{v_{p-1}, v_p\}$. For a path $P = v_1 \dots v_p$ and a vertex u, uP (Pu resp.) is the path $uv_1 \dots u_p$ ($u_1 \dots u_p v$ resp.). If $P_1 = u_1 \dots u_p$ and $P_2 = v_1 \dots v_q$ are paths such that $V(P_1) \cap V(P_2) = \{u_p\} = \{v_1\}$, then $P_1 + P_2$ is the *concatenation* of P_1 an P_2, i.e., the path $u_1 \dots u_{p-1} v_1 \dots v_q$.

Let $\mathcal{P} = \{P_1, P_2, \dots, P_l\}$ be a set of paths, for some $l \geq 2$. We say that the paths in $\mathcal{P}$ are *vertex-disjoint* (resp. *internally vertex-disjoint*) if no two of them have any common vertices (resp. if two of them have common vertices then these vertices are their endpoints).

If $P = u_1 \dots u_k$ is a path then $C = (V(P), E(P) \cup \{u_k, u_1\})$ is a cycle, i.e. a cycle is obtained from a path by adding an edge between its endpoints. We use $u_1 \dots u_k u_1$ to denote the cycle with the vertices $u_1, \dots, u_k$ and the edges $\{u_1, x_u\}, \dots, \{u_{p-1}, u_p\}, \{u_p, u_1\}$. The *length* of a cycle is the number of its edges (which is equal to the number of its vertices). Let C be a cycle in a graph G. An edge $e \in E(G) \setminus E(C)$ which joins two vertices of C is called a *chord* of C. A cycle in G that has no chords is called an *induced cycle* in G.

Graph connectivity. We say that a graph G is *connected* if for any two vertices $u, v \in V(G)$, there exists at least one path in G with u and v as its endpoints. If G is not connected, we define the *connected components* of G to be the induced subgraphs $G[V_i]$, where $V_1, \ldots, V_k$ are the maximal (under the subset relation) vertex sets that induce connected subgraphs of G.

Let k be a positive integer. We say that a graph G is *k-connected* if for every pair u, v of vertices in G, there exist at least k internally vertex disjoint paths in G that connect u and v. Actually, the above definition is a result from Karl Menger [68] (known as *Menger's theorem*) and gives an equivalent characterisation of k-connected graphs. The original definition says that a (connected) graph G is k-connected, if the removal of any k vertices of G results to a connected graph or in other words in order to undermine the connectivity of G one has to remove at least $k + 1$ of its vertices.

Trees and forests. Let G be a graph. We say that G is a *forest* if it does not contain any cycle as a subgraph. If G is a forest and is also connected, then we say that G is a *tree*. It is now easy to observe that a graph is a forest if and only if all its connected components are trees, which also justifies the names of these graph classes.

Let T be a tree. We say that a vertex $t \in V(T)$ is a *leaf* of T if $\text{deg}_T(t) \leq 1$. A vertex of T that is not a leaf is called a *non-leaf vertex*. It is easy to prove that any two vertices u and v in a tree T are linked via a unique path of T, which we denote by uTv.

Due to their "simple" structure, trees enjoy some nice algorithmic properties. This is one of the reasons why researchers tried to expand the class of trees into wider graph classes which, in a way, possess a tree-like structure and thus share the nice algorithmic properties of trees. We will give more details about this vague statement later, when we will talk about *treewidth* and *dynamic programming on graphs of bounded treewidth*.

The class of *planar* graphs is of special importance for us, as one of the main results of this work is an algorithm solving Cyclability in FPT-time for this particular class of graphs. Next we define the class of planar graphs and mention some of the most important notions related to them

Planar, plane, and outerplanar graphs. A graph G is called *planar* if it can be embedded in the plane $\mathbb{R}^2$ (or equivalently in the sphere $\mathbb{S}^2 = \{(x, y, z) \in \mathbb{R}^3 : x + y + z = 1\}$) in such a way that there are no two edges of it whose embeddings intersect (they can meet only at their endpoints). Such an embedding is called a *planar embedding of* G and we say that such an embedding is a *plane graph* (observe that a planar graph can have more than one planar embeddings that can also be different from a topological point of view). Given a plane graph G we denote its faces by $F(G)$, i.e. $F(G)$ is the set of the connected components of $\mathbb{R}^2 \setminus G$ (in the operation $\mathbb{R}^2 \setminus G$ we treat G as the set of points of $\mathbb{R}^2$ corresponding to its vertices and its edges).

The *dual*, G^*, of a plane (resp. planar) graph G is also a plane (resp. planar) graph and has one vertex for each face of G. There is an edge between two vertices of G^* if and only if the boundaries of their corresponding faces share an edge (observe that if a plane graph is not connected it can have, two or more, different (from a topological point of view) dual graphs). For an example of a plane graph and its corresponding dual graph see Figure 2.1.1.

An *outerplanar* graph is a plane graph whose vertices are all incident to the infinite face. If an edge of an outerplanar graph is incident to its infinite face then we call it *external*, otherwise we call it *internal*. The *weak dual* of an outerplanar graph G is the graph obtained from the dual of G after removing the vertex corresponding to the infinite face of the embedding.

Grids. Let $m, n \geq 1$. The $(m \times n)$-*grid* is the Cartesian product of a path of length $m - 1$ and a path of length $n - 1$. In the case of a *square grid* where $m = n$, we say that n is the *size* of the grid. Given that $n, m \geq 2$, the *corners* of an $(m \times n)$-*grid* are its vertices of degree 2. When we refer to a $(m \times n)$-*grid* we will always assume an orthogonal orientation of it that classifies its corners to the *upper left*, *upper right*, *down right*, and *down left* corner of it.

Given that Γ is an $(m \times n)$-*grid*, we say that a vertex of G is one of its *centers* if its distance from the set of its corners is the maximum possible. Observe that a square grid of even size has exactly 4 centers. We also consider an $(m \times n)$-grid embedded in the plane so that, if it has more than 2 faces then the infinite one is not a square. The *outer cycle* of an embedding of a $(m \times n)$-grid is the one that is the boundary of its infinite face. We also refer to the *horizontal* and the *vertical lines* of a $(m \times n)$-*grid* as its paths between vertices

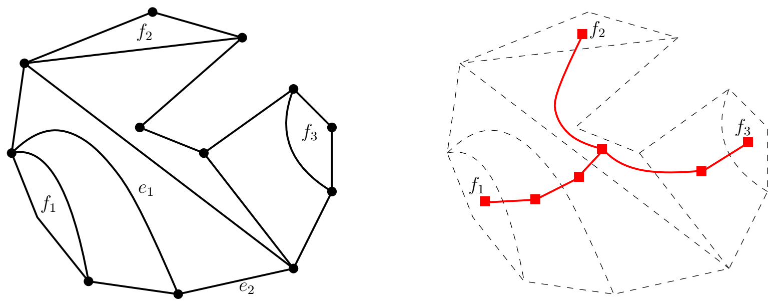


Figure 2.2: An outerplanar graph at the left and its weak dual at the right. Its simplicial faces are f_1, f_2 and f_3, e_1 is an internal edge, and e_2 is an external edge.

of degree smaller than 4 that are traversing it either "horizontally" or "vertically" respectively. We make the convention that an $(m \times n)$-grid contains m vertical lines and n horizontal lines. The *lower horizontal line* and the *higher horizontal line* of Γ are defined in the obvious way (for an example see Figure 2.1.1).

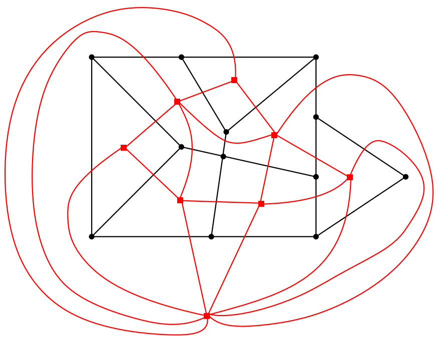


Figure 2.3: A plane graph (black) embedded in the plane along with its dual graph (red). There is one dual vertex (red square) for every face of the plane graph. Any edge of the black graph is on the boundary of exactly two of its faces which are connected by an edge in the dual (red) graph.

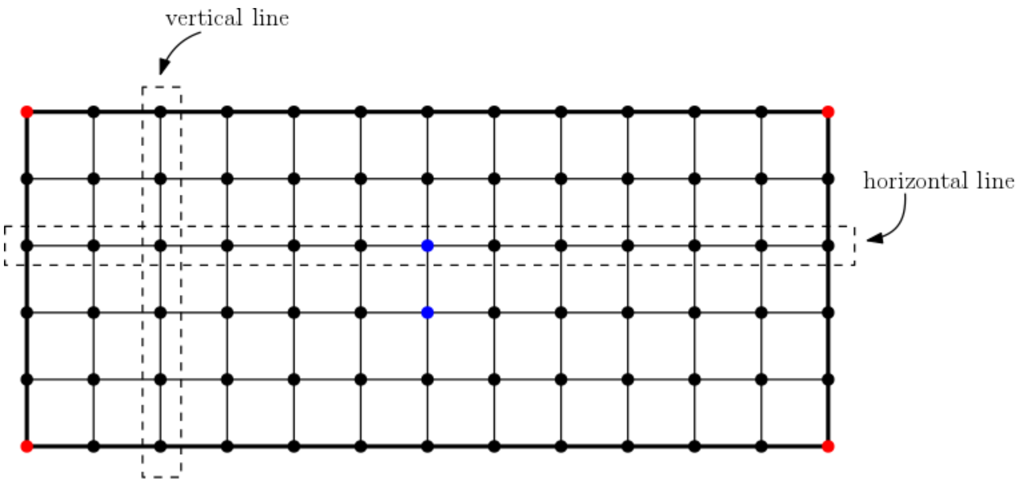


Figure 2.4: A (13×6)-grid is depicted. Its corners are the red vertices and its centers are the two blue vertices. The outer cycle is the bold rectangle that contains the corners of the grid.

2.1.2 Graph operations and relations between graphs

Operations between graphs. Let G and H be two graphs. We define the *union* of G and H as the graph

$$G \cup H = \big(V(G) \cup V(H), E(G) \cup E(H)\big),$$

their *intersection* as the graph

$$G \cap H = \big(V(G) \cap V(H), E(G) \cap E(H)\big),$$

and, finally, we define the *product* of G and H as the graph

$$\begin{aligned} G \times H \quad = \quad & \Big(V(G) \times V(H), \{\{(u_1, v_1), (u_2, v_2)\} \mid \\ & (\{u_1, u_2\} \in E(G) \wedge v_1 = v_2) \vee (\{v_1, v_2\} \in E(H) \wedge u_1 = u_2)\}\Big) \end{aligned}$$

Next we define some basic *operations* (or *transformations*) on graphs. Let

G be a graph and let $v \in V(G)$ and $e \in E(G)$:

- *Vertex removal* (*deletion*): We denote by $G - v$ (or by $G \setminus v$) the graph obtained from G after *removing* (*deleting*) vertex v, i.e. $V(G-v) = V(G) \setminus \{v\}$ and $E(G - v) = \{e \in E(G) \mid v \notin e\}$. For $S \subset V(G)$, we denote by $G \setminus R$ the graph obtained from G after deleting from it all vertices in R.

- *Edge removal* (*deletion*): We denote by $G - e$ (or by $G \setminus e$) the graph obtained from G after *removing* (*deleting*) edge e, i.e. $V(G - e) = V(G)$ and $E(G - v) = E(G) \setminus \{e\}$. For $F \subseteq E(G)$, we denote by $G \setminus E$ the graph obtained from G after deleting from it all edges in F.

- *Vertex dissolution*: Suppose that $deg_G(v) = 2$ and $\{u, v\}, \{v, w\} \in E(G)$. We denote by G/v the graph obtained from G after deleting vertex v and adding the edge $\{u, w\}$ (if this edge does not already exist in G), i.e. $V(G/v) = V(G) \setminus \{v\}$ and $E(G/v) = E(G) \setminus \big\{\{u, v\}, \{v, w\}\big\} \cup \big\{\{u, w\}\big\}$.

- *Edge subdivision*: The operation of removing an edge $e = \{u, v\} \in E(G)$ from G and adding a path of length 2 whose endpoints are u and v is called a *subdivision* of edge e in G.

- *Edge contraction*: Let $e = \{u, v\} \in E(G)$ and let $v^* \notin V(G)$. We denote by G/e the graph obtained from G after removing vertices u and v and adding the (new) vertex v^* and an edge between v^* and every vertex in $N_G(\{u, w\})$, i.e., $V(G/e) = V(G) \setminus \{u, w\} \cup \{v^*\}$ and $E(G/e) = E(G) \setminus E_G(u) \setminus E_G(w) \cup \big\{\{v^*, x\} \mid x \in N_G(\{u, w\})\big\}$.

Let G and H be two graphs. Using the, previously defined, graph operations we define some relations on the class of graphs:

- *Subgraph*: If H can be obtained by applying vertex and edge deletions on G, we say that H is a *subgraph* of G and write $H \subseteq G$.

- *Induced subgraph*: If H can be obtained by applying vertex deletions on G, we say that H is an *induced subgraph* of G and write $H \subseteq_{in} G$.

- *Spanning subgraph*: If H can be obtained by applying edge deletions on G, we say that H is a *spanning subgraph* of G and write $H \subseteq_{sp} G$.

- *Subdivision*: If H can be obtained by applying edge subdivisions on G, we say that H is a *subdivision* of G and write $H \subseteq_{es} G$.
- *Contraction:* If H can be obtained by applying edge contractions on G, we say that H is a *contraction* of G and write $H \leq_c G$.
- *Topological minor*: If H can be obtained by applying vertex deletions, edge deletions and vertex dissolutions on G, we say that H is a *topological minor* of G and write $H \leq_{tm} G$.
- *Minor*: If H can be obtained by applying vertex deletions, edge deletions and edge contractions on G, we say that H is a *minor* of G and write $H \leq_m G$.

The most well-studied from the previous relations are $\subseteq$, $\subseteq_{in}$, $\subseteq_{sp}$, which are known as the main *subgraph relations* and $\leq_m$, $\leq_{tm}$, which are known as the main *topological relations*. As it will become obvious later, the topological relations play a special role in our work, that is why we give an alternative definition for the minor relation, which is probably more intuitive (for an example see Figure 2.5).

Alternative definition of minor relation. Let G and H be two graphs. H is a minor of G if there exists a function $f : V(G) \to V(H)$ such that:

- For every $v \in V(H)$, the reverse image of v through f, i.e. $f^{-1}(v)$, is a connected set in G, and for any $u \in V(H) \setminus \{v\}$ it holds that $f^{-1}(v) \cap f^{-1}(u) = \emptyset$.
- For every edge $\{v, u\} \in E(H)$, there exists at least one edge with endpoints in $f^{-1}(v)$ and $f^{-1}(u)$ in G.

If such a function exists we also say that G *contains* H *as a minor*.

Definition 2.1.1. *Let $\mathcal{G}$ be a graph class and let $\sqsubseteq \in \{\subseteq, \subseteq_{in}, \subseteq_s, \subseteq_{es}, \leq_c, \leq_{tm}, \leq_m\}$. We say that $\mathcal{G}$ is closed with respect to $\sqsubseteq$ if for every graph G*

$$G \in \mathcal{G} \text{ and } G' \sqsubseteq G \implies G' \in \mathcal{G}$$

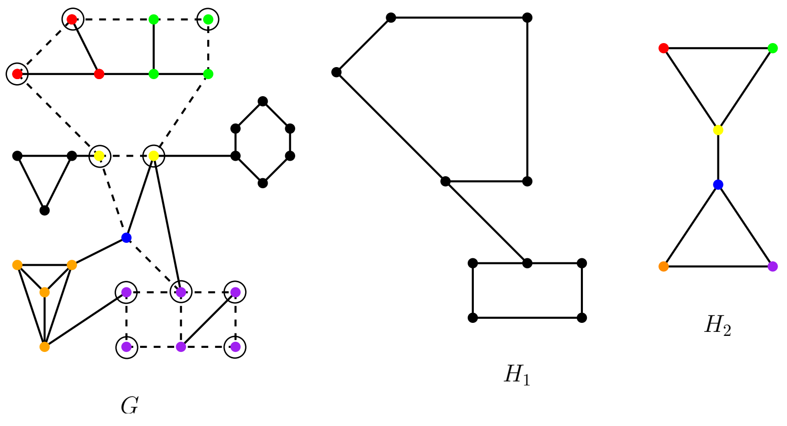


Figure 2.5: The graph H_1 is a topological minor of the graph G (certified by the circled vertices of G and the dashed edges of G) and the graph H_2 is a minor of G (consider the function $\phi : V(H) \to 2^{V(G)}$ that sends a vertex of H_2 to the subset of vertices of G of the same colour and observe that each "colour-class" in G induces a connected subgraph.

2.1.3 Graph parameters

In this subsection we talk about graph parameters in general and focus on a specific well known width-parameter, namely *treewidth*, which is relevant to our work.

Graph parameters. A graph parameter is a (partial) function $p : \mathcal{G} \to \mathbb{N}$, i.e., a function that maps graphs to non-negative integers. We say that a parameter p is computable if there exists an algorithm that given a graph G as an input, either outputs the value $p(G)$.

Some well known graph parameters are the Δ (resp. δ) denoting the maximum (resp. minimum) degree of a graph, the maximum independent set, the minimum vertex cover, the feedback vertex set, the chromatic number, the girth etc.

Given a graph parameter p and a relation $\sqsubseteq$ on graphs, we say that p is $\sqsubseteq$-closed if for every two graphs G and H with $H \sqsubseteq G$ it holds that $p(H) \leq p(G)$.

In this thesis we will often refer to a certain parameter, which is the main representative of the class of *width parameters* and, in a way, reflects the resemblance of a graph with a tree. That is why it is called *treewidth* and one of the several existing definitions is the following:

Treewidth. A *tree decomposition* of a graph G is a pair $\mathcal{D} = (\mathcal{X}, T)$ in which T is a tree and $\mathcal{X} = \{X_i \mid i \in V(T)\}$ is a family of subsets of $V(G)$ such that:

- $\bigcup_{i \in V(T)} X_i = V(G)$
- for each edge $e = \{u, v\} \in E(G)$ there exists an $i \in V(T)$ such that both u and v belong to X_i
- for all $v \in V$, the set of nodes $\{i \in V(T) \mid v \in X_i\}$ forms a connected subtree of T.

The *width* of a tree decomposition is defined to be the number max$\{|X_i| \mid i \in V(T)\} - 1$. The *treewidth* of a graph G (denoted by **tw**(G)) is the minimum width over all possible tree decompositions of G. At Figure 2.6, there is a graph on 10 vertices and a tree decomposition of it with width 3. It is easy to confirm that any tree decomposition of this graph has width at least 3, thus the treewidth of this graph its is 2. We give some examples of the values of treewidth for some specific graph classes.

Trees and forests have, as expected, treewidth 1 and cycles have treewidth 2. The treewidth of an outerplanar graph is at most 2, but the treewidth of a planar graph can be arbitrarily large (for example the $(n \times n)$-grid has treewidth n) but still it is sublinear to the number of vertices, more precisely $O(\sqrt{n})$. On the other hand, every graph G that excludes a planar graph H as a minor has treewidth at most c_H, where c_H is a constant that depends on graph H.

The concept of treewidth was originally introduced by Umberto Bertelé and Francesco Brioschi (1972) under the name of *dimension*. It was later rediscovered by Rudolf Halin (1976), and it was rediscovered for a third time by Neil Robertson and Paul Seymour in [78], and played a crucial role to the developments of the Graph Minors series papers.

Branchwidth. A *branch decomposition* of a graph G is a pair (T, τ), where T is a tree with vertices of degree one or three and τ is a bijection from $E(G)$ to the set of leaves of T. The *order function* $\omega : E(T) \to 2^{V(G)}$ of a branch decomposition maps every edge e of T to a subset of vertices $\omega(e) \subseteq V(G)$ as follows. The set $\omega(e)$ consists of all vertices $v \in V(G)$ such that there exist edges $f_1, f_2 \in E(G)$ with $v \in f_1 \cap f_2$, and such that the leaves $\tau(f_1)$, $\tau(f_2)$ are in different components of $T - \{e\}$.

The *width* of a branch decomposition (T, τ) is equal to $\max_{e \in E(T)} |\omega(e)|$ and the *branchwidth* of G, denoted by $\mathbf{bw}(G)$, is the minimum width over all branch decompositions of G.Branchwidth was introduced by Robertson and Seymour in [79]. For any graph G, $\mathbf{bw}(G)$ and $\mathbf{tw}(G)$ are within a constant factor of each other, however, unlike treewidth, branchwidth is computable in polynomial time on planar graphs.

The following two results combined, imply a relation between the treewidth of a planar graph and the minimum size of the largest grid-minor that it contains.

Result 1. ([53]) If G is a planar graph and $\mathbf{bw}(G) \geq 3k + 1$, then G contains a $(k \times k)$-grid as a minor.

Result 2.([79]) If G is a graph, then $\mathbf{bw}(G) \leq \mathbf{tw}(G) + 1 \leq \frac{3}{2} \cdot \mathbf{bw}(G)$.

Proposition 2.1.1. *If G is a planar graph and* $\mathbf{tw}(G) \geq 4.5 \cdot k + 1$, *then* G *contains a* $(k \times k)$*-grid as a minor.*

2.2 Parameterized Complexity

2.2.1 Basic definitions

Let Σ be an alphabet (we usually think of Σ as the set $\{0, 1\}$) and let Σ^* (the *Kleene star* of Σ) be the set of all finite sequences with elements from Σ. An element of Σ^* is called a *word* on the alphabet Σ.

Parameterized languages and problems. We will call every subset L of $\Sigma^* \times \mathbb{N}$ a *parameterized language*, and for every element $\langle x, k \rangle \in L \subseteq \Sigma^* \times \mathbb{N}$ we will say that k is the *parameter* and x the *main input*. For every $k \in \mathbb{N}$, we call $L_k = \{\langle x, k \rangle : \langle x, k \rangle \in L\}$ the kth *slice* of L.

A decision problem Π is called a *parameterized problem* if any instance of is encoded as a pair $\langle x, k \rangle \subseteq \Sigma^* \times \mathbb{N}$. We will say that $\langle x, k \rangle$ is a yes-instance

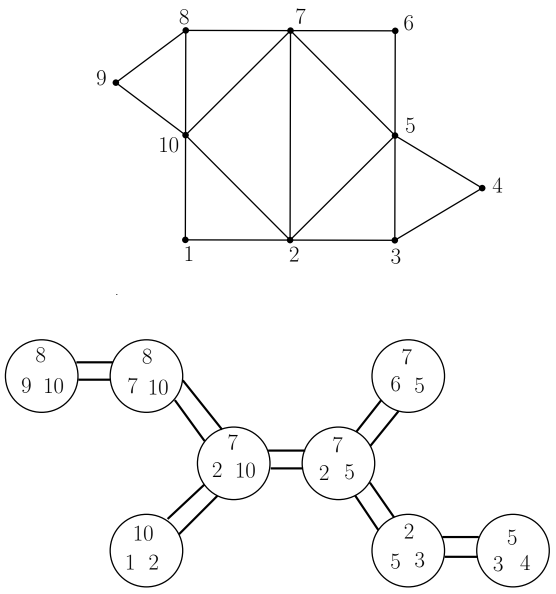


Figure 2.6: An example of a graph of treewith 2 along with a tree-decomposition of minimum width.

for Π if $\langle x, k\rangle$ encodes an instance for which the question imposed in problem Π is answered positively, and will write $\langle x, k\rangle \in \Pi$. Otherwise we will say that $\langle x, k\rangle$ is a no-instance for Π and will write $\langle x, k\rangle \notin \Pi$. If Π is a parameterized problem then it naturally defines the parameterized language

$$\begin{aligned} L_\Pi &= \{\langle x, k\rangle \in \Sigma^* \times \mathbb{N} \mid \langle x, k\rangle \text{ is a yes-instance for } \Pi\} = \\ &= \big\{\langle x, k\rangle \in \Sigma^* \times \mathbb{N} \mid \langle x, k\rangle \in \Pi\big\} \end{aligned}$$

and conversely, a parameterized language L can be associated with the problem Π_L for which $\langle x, k\rangle$ is a yes-instance if $\langle x, k\rangle \in L$ and a no-instance otherwise. In what follows we will not distinguish between a parameterized language and its corresponding parameterized problem unless it is necessary.

Fixed-parameter tractability (the class FPT**).** We say that a parameterized problem Π is *fixed-parameter tractable* if there exists an algorithm (or more formally a deterministic Turing Machine) $\mathcal{A}$, a constant c, and a computable [1] function f such that, for all $\langle x, k\rangle \in \Sigma^* \times \mathbb{N}$, $\mathcal{A}\big(\langle x, k\rangle\big)$ runs for at most $f(k) \cdot |x|^c$ steps (where $|x|$ is the length of the encoding of x) and

$$\langle x, k\rangle \in L_\Pi \iff \mathcal{A}\big(\langle x, k\rangle\big) = 1$$

where we suppose that the output of algorithm $\mathcal{A}$ is 1 if it accepts its input and 0 otherwise. The class of all fixed-parameter tractable problems is called FPT and is considered to be the class of efficiently solvable problems in the framework of Parameterized Complexity (class FPT can also be thought of as the analog of P in terms of classical complexity).

For a problem in FPT, we sometimes say that it can be solved in FPT-*time* or that there exists an FPT-*time algorithm* for solving it, meaning that there exists an algorithm that solves it in $f(k) \cdot |x|^c$ time (where k is the parameter, $|x|$ is the length of the encoding of the input, f is a computable function, and c is a constant).

2.2.2 Why Parameterized Complexity?

The idea of finding a solution to a problem, which can be described as a series of concrete and "easy" to perform steps, is very old and can be tracked far back in the history of Mathematics. We are referring, of course, to the idea that is nowadays widely known as an *algorithm*. One of the oldest and most well known algorithms (although at that point this notion did not yet exist) is the *euclidean algorithm* (given by the "father of Geometry", Euclid) for finding the greatest common divisor of two given integers.

The systematic study of algorithms and problems that admit algorithmic solutions started less than 100 years ago, in the 1930s, as a branch of Mathematics named *Computability Theory*. One of the main objects of this field is to study whether there exists an algorithmic solution for a given problem or any solution is provably *non-constructive*. But what does "algorithmic" even mean? Is it possible to formally define such a notion?

[1]The weak demand for f to be just computable can be quite alerting. After all, parameterized complexity is supposed to be related to practical computation. Of course the same issue arises also with the definition of P and additionally it turns out that, most of the times, constant the c is small and function the f is tolerable, e.g. is an exponential function of k.

A widely accepted answer to the previous, nearly philosophical, question is given from the *Church-Turing thesis* which, roughly, states the following:

There exists an algorithmic solution for a problem if and only if there exists a Turing machine that solves it.

A Turing machine is a computation model introduced by Alan Turing in 1936 [89] and is the predecessor of todays personal computer. As the topic of computational models is out of the scope of this thesis, for an extensive introduction see [67] and [86]. Of course the Church-Turing thesis is not a conjecture that can be proved or disproved in some axiomatic system. It is more like a meta-conjecture whose "credibility" has been tested throughout the years.

Having established a framework for arguing about computability and motivated by the rapid improvement of "real" computational models (not just theoretical constructions as the Turing machines) who could execute complicated tasks increasingly fast, researchers took a step forward and started to explore the notion of *efficiency*. Efficiency refers to the *resources* needed for solving a problem algorithmically. The two main such resources, which are usually considered as measures of efficiency of an algorithm, are *space* and *time*. In this thesis we will focus on the latter.

Given these new parameters, computational problems can be classified further in *complexity classes* based on the efficiency of algorithms that solve them. Undoubtedly, the most well known complexity classes are P and NP, where P contains the problems that can be solved efficiently (there exist deterministic algorithms that solve them in time that is bounded by a fixed polynomial on the size of the input) and NP contains the problems which require nondeterminism in order to be solved in polynomial time. It is widely believed that P $\neq$ NP, which can be roughly translated to the fact that there exist computational problems for which any algorithm solving them needs exponential time. Consequently, most computer scientists face at some point the following question when studying a computational problem:

Is there an efficient algorithm solving the problem of interest? If not, is it possible to provide some evidence that it cannot be solved efficiently?

The natural approach to address this type of questions is either trying to come up with a polynomial time algorithm that solves the problem (which places it in P) or proving that, assuming P $\neq$ NP, it is in NP but not in P. This can be done by *reducing* an NP-*complete* problem to the problem of interest. Roughly speaking, NP-*complete* problems are the hardest in the class NP and the reduction of such a problem to another problem suggests that the latter is at least as hard, thus characterised as intractable. For more information about the theory of NP-completeness we refer the reader to the monumental work of Garey and Johnson in [45] and to all introductory Complexity Theory books such as [75] and [5].

The construction of a polynomial time algorithm is usually the best outcome one can hope for (although nowadays this claim becomes more and more inaccurate as we need to solve problems where the input is huge; the running time of a polynomial n^4 algorithm when the input is the web network does not seem appealing at all! In many cases even a linear algorithm can be practically useless and this means that the desired algorithm will not even have the opportunity to access all of its input. For some more information on the subject we refer the reader to [85]). But what happens if we prove that our problem is NP-complete? Is this the end of the story? Fortunately, the answer is no and we briefly present the main side roads one can choose from:

- **Approximation:** A very important class of problems that attracts much attention (mainly due to applications in Operational Research) is the one of *optimization problems* where the task is to find the *best solution* from all feasible solutions. Unfortunately, many optimization problems have proved to be NP-complete. When the need for an exact solution is not imperative, a way to overcome this difficulty is trying to design efficient algorithms that find a solution which is guaranteed to be "close" to the optimal.

  Of course an analogue of intractability arises in this setting too and much work has been done in the direction of obtaining *innaproximability* and *lower bounds* results. Approximation algorithms have been developed rapidly in the last decades and proved to be a very fruitful area. For an extensive introduction we refer the reader to [93] and [90].

- **Use of randomness:** Another tool that can be used to cope with an NP-complete problem is *randomness* and the study of randomized algorithms was spurred by the discovery of a randomized primality test [87]. The main idea of this approach is roughly the following: In order to "prune" some of the branches of computation, which seem to be unavoidably exponential (under worst-case analysis) when trying to solve an NP-complete problem, the randomized algorithm makes some random choices and based on them, and probably other deterministic computation, produces an answer.

One has to distinguish between algorithms that use randomness in order to reduce the expected running time and always terminate in bounded time producing the right answer (called *Las Vegas algorithms*) and algorithms that terminate in polynomial time but there is a chance that they produce a wrong answer or no answer at all (called *Monte Carlo algorithms*).

Having designed a randomized algorithm for a problem, it is sometimes possible to produce a deterministic algorithm for solving the same problem. This procedure is known as *derandomization* and has attracted much attention recently. More information about randomized algorithms can be found in [71] and [70] and for some information about the complexity classes that arise from randomized algorithms see [75] and [5].

- **Parameterization:** When a problem is NP-complete, any exact deterministic algorithm that solves it needs (in the worst case) exponential (or at least superpolynomial) to n time, where n is the length of the input. The parameterized complexity point of view examines whether this exponential explosion on the running time unavoidably "spreads" to a large part of the input (meaning a part whose length depends on n) or there are some particular *parameters* of the problem that cause the increase on the running time. For some NP-complete problems that are of great importance in other areas, such as biology, there were algorithms that, although being exponential in the worst case, worked efficiently in practice. Then a natural question arose:

  Are there some parameters in these particular problems which happen to be bounded and this way "soften" the intractability? Can theory formalize this phenomenon and study it methodically?

  Research has shown that such parameterizations exist for many, previously classified as intractable, problems and when restricted to the case where they are bounded, there exist algorithms that justify their placement into the sphere of tractability. The related area, which has grown to be an entire field in Computer Science, is called *Parameterized Complexity* and the algorithms designed in this setting are called *parameterized* (or *multivariate*) *algorithms*. The main introductory texts for Parameterized Complexity are [31], [73], [42], [36], and [22], as they have appeared chronologically.

All the previously mentioned methods have been studied extensively in the last decades and each one of them constitutes a wide research area in the frame of Theoretical Computer Science. Of course, ideas and techniques from any of these areas "flow" between them and researches are always interested in combining notions from some of or all the fields, as, for example, indicated (already in the title) by [72]. In this thesis, we focus on the last suggestion for "NP-completeness treatment" and we start by explaining, via concrete examples, why this approach is promising.

Why Parameterized Complexity? For an example consider the following parameterized problem

p-Vertex Cover
Input: A graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Is there a set $S \subseteq V$ such that $|S| \leq k$ and $G \setminus S$ has no edges?

This problem is a parameterized version of the classical Vertex Cover problem (which is one of the first 21 problems proven to be NP-complete by R. Karp in [45]), with the *natural parameterization* in the sense that the parameter is chosen to be the size of the desired solution. The problem obtained when using the natural parameterization of problem Π, will be denoted by p-Π and when it is clear from the context, we will sometimes omit p.

It is not hard to prove that p-Vertex Cover can be solved in time $2^k \cdot n$ (using *bounded search trees*, see [22]), which places it in FPT. Great effort has been made in order to improve on the parametric dependance for this important problem and the state of the art algorithm (due to Chen, Kanj, and Xia [15]) runs in time $O(1.2738^k + k|G|)$ and uses a series of techniques reflecting the development of parameterized algorithm design throughout the years. This means that, even though Vertex Cover is known to be "hard" from a classic complexity viewpoint, it becomes tractable even when the solution we are looking for is big (as its natural parameterization admits an algorithm whose dependence on the size of the graph is linear and the dependence on the size of the solution, the parameter k, is *low exponential*).

But can we hope that the natural parameterization of any hard, say NP-complete problem, is enough to place it in the sphere of tractability? Consider the parameterized version of the well-known Clique problem

p-Clique
Input: A graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Is there a set $S \subseteq V$ such that $|S| \geq k$ and $G[S]$ has no edges?

The Clique problem is also one of the first important problems proven to be NP-complete in [55]. An obvious algorithm for solving p-Clique is to check all possible subsets of $V(G)$ of size k (also called *brute force*), which results to an $O(|G|^k)$-time algorithm $\big($as $\binom{n}{k} = O(n^k)\big)$. The observant reader can spot the main difference between the running time of this algorithm and even the simplest one for p-Vertex Cover. The difference is that the parameter k appears at the exponent of $|G|$ (size of the input) for the case of p-Clique, which makes the algorithm less attractive. As we will shortly see, the p-Clique problem is W$[1]$-complete and thus considered intractable, even from the parameterized complexity point of view.

As we will see in the following section, class W$[1]$ cannot encapsulate all fixed-parameter intractable problems. The parameterized versions of two well-known NP-complete problems that are not believed to be in W$[1]$ are the following

p-Dominating Set
Input: A graph $G = (V, E)$ and an integer k.
Parameter: k
Question: Is there a set $D \subseteq V$ such that every vertex in $V \backslash D$ is adjacent to some vertex in D?

p-Set Cover
Input: A finite set $\mathcal{U}$, a set $\mathcal{F} \subseteq \mathcal{P}(\mathcal{U})$, and an integer k.
Parameter: k
Question: Is there a $\mathcal{C} \subseteq \mathcal{F}$ such that $|\mathcal{C}| = k$ and $\bigcup_{C \in \mathcal{C}} C = \mathcal{U}$?

To sum up, even when we only consider natural parameterizations, it becomes obvious that the classical complexity landscape is getting refined. Some of the intractable problems become tractable and some others are characterised as fixed-parameter intractable and are classified in different hardness classes. Things get even more interesting when more parameters, such as the *maximum degree*, *girth*, *treewidth*, *cutwidth*, *cliquewidth*, *chromatic number* etc. come in to play. It is often the case that the parameterized complexity of a problem changes when we focus on different parameters.

The rapid increase of computational power, the exponential growth of information and the wide use of computers into pretty much every aspect of human activity, made the classical complexity theory seem outdated (from the viewpoint of actual implementation of algorithms). Researchers realised that in order to design efficient algorithms in a highly structured world, one has to take into account the structural characteristics of the available data. This imperative need for exploitation of structure lead to the development *fine-grained analysis* and parameterized complexity, which has become a rapidly growing field of Computer Science, with many theoretical and practical accomplishments to show.

2.2.3 Fixed-parameter intractability and the W-hierarchy.

In order to introduce the theory of fixed-parameter intractability, we will first present an overview of some basic concepts of classical intractability theory. We will keep the formality simple, for now, as our main goal is develop some intuition and not to present strict formalism. This chapter will work either as a brief introduction or as a reminder of some fundamental notions. In any case, the intention is to create a natural transition to the parameterized complexity setting, which is essential for some parts of this thesis.

We will define the notion of *polynomial reductions* and the class of NP-complete problems.

Reductions. A *polynomial-time many-one reduction* (or *Karp reduction*) from problem B to problem C is a polynomial time algorithm $\mathcal{A}$ which, given an instance x of B, outputs an instance $\mathcal{A}(x)$ of C such that

$$x \text{ is a yes-instance of } B \iff \mathcal{A}(x) \text{ is a yes-instance of } C$$

This kind of reduction "transfers" computational intractability: If there is no polynomial-time algorithm for solving problem B, then the same holds for problem C. On the other hand, if C can be solved in polynomial time then B is also polynomial-time solvable. Obviously, in order to start benefiting from this method of reducing a problem to another, one has to identify a "hard" problem or a problem which is widely believed to be hard. After establishing such a problem, any other problem reduced to our "hard" problem inherits the "pessimism". In the late 1960s and early 1970s, Stephen Cook [18] and Leonid Levin [63] worked (independedly) towards this direction and gave birth to the theory of computational intractability.

Classical intractability theory. The first, and most widely studied, notion of computational intractability is undoubtedly the *theory of NP-completeness*. We give one of the several existing definitions of the class NP: a language L belongs in the class NP if and only if there exists a polynomial p and a polynomial relation R (meaning that R can be decided in polynomial time for any (x, y) pair) such that

$$x \in L \iff \exists y\big(|y| \leq p\big(|x|\big) \wedge R(x,y) = 1\big)$$

A language L is *NP-complete* if and only if $L \in$ NP and for any language $L' \in$ NP, L' is polynomially reducible to L, i.e. $L' \leq^p_m A$. The definition of a computational problem being NP-complete is completely analogous to the one for languages. The most obvious, although somewhat "artificial", NP-complete problem is the following

Turing Machine Acceptance
Input: A nondeterministic Turing machine M, a string x, and a natural number n.
Question: Is there a computation path of M accepting x in at most n steps?

Clearly, Turing Machine Acceptance is NP-complete, as a nondeterministic Turing machine can guess y and then check if $R(x,y) = 1$, which roughly describes a reduction of a problem in NP to Turing Machine Acceptance.

The first natural NP-complete problem is SAT ([18] and [63]), the problem of deciding, given a boolean formula ϕ, if there exists an assignment to its variables that make ϕ true. Nowadays, several hundreds of interesting computational problems are known to be NP-complete but the decisive step that boosted research in this direction was the work of S. Karp [55] who proved the NP-completeness of 21 important combinatorial problems, initiating a huge list of results in this direction. Many of these problems have natural structure which can be translated as hope for the existence of efficient (polynomial) algorithms which are based on a clever exploitation of this structure.

Unfortunately, the NP-completeness of Turing Machine Acceptance, a such generic and opaque problem, makes the existence of any such algorithm seem really unreasonable, as this would imply that we could be able to decide in polynomial time whether a given Turing machine on a given input has some accepting path. That is why the Cook-Levin theorem is considered to be strong evidence for the P $\neq$ NP hypothesis. Today, there is much more evidence to support this hypothesis but this subject is out of the scope of this thesis.

Next, we define a notion that allows the transfer of "pessimism" in the Parameterized Complexity framework, or, in other words, it allows the transfer of fixed-parameter intractability from a parameterized language (or problem) to another.

Definition 2.2.1 (Parameterized reductions)**.** *Let $L, L' \subseteq \Sigma^* \times \mathbb{N}$ be two parameterized languages. We say that L* reduces to L' by a standard parameterized m-reduction *if there exist, a constant c and functions $f : \Sigma^* \times \mathbb{N} \to \Sigma^*$, $g : \mathbb{N} \to \mathbb{N}$ and $h : \mathbb{N} \to \mathbb{N}$, such that*

- *$f(\langle x, k\rangle) = x'$ can be computed in time $h(k) \cdot |x|^c$ and*
- *$\langle x, k\rangle \in L$ if and only if $\big\langle x', g(k)\big\rangle \in L'$.*

Fixed-paramerer intractability. Following the method for establishing classical intractability results, researchers tried to prove that a number of combinatorial problems are of the same fixed-parameter complexity as a problem about nondeterministic Turing machines. The problem of Turing Machine Acceptance can be naturally generalized by the following (somewhat artificial) problem:

Short Turing Machine Acceptance
Input: A nondeterministic Turing machine M and a string x.
Parameter: A positive integer k.
Question: Is there a computation path of M accepting x in at most k steps?

It seems reasonable to support that, if someone accepts that the Turing Machine Acceptance problem is intractable (in the classical way), then the Short Turing Machine Acceptance problem is fixed-parameter intractable. Indeed there are many problems that can be proved (via parameterized reductions) to have the same parameterized complexity as Short Turing Machine Acceptance and this strengthens the hypothesis of Short Turing Machine Acceptance not being fixed-parameter tractable.

In order to define the main hardness classes for Parameterized Complexity, namely the W-*hierarchy*, we need to introduce some definitions about circuits, and more specifically, *decision circuits*.

Boolean circuits. A *boolean circuit* is a directed acyclic graph whose nodes are labeled in the following way:

- every node of indegree 0 is an *input node*
- every node of indegree 1 is a *negation node* (with $\neg$ as a symbol)
- every node of indegree at least 2 is either an and-*node* (with $\wedge$ as a symbol) or an or-*node* (with $\vee$ as a symbol)

Exactly one of the nodes with outdegree 0 is labeled as the *output node*. The *depth* of a boolean circuit is the maximum length of a path from an input node to the output node. Assigning boolean values, i.e., values in $\{0, 1\}$, to the input nodes determines the value of every node in the obvious way: a negation node turns 0 to 1 and vice versa, an end-node outputs 1 if and only if all its inputs are 1, and an or-node outputs 1 if and only if it receives at least one 1. If the value of the output node is 1 for an assignment α on the input variables, then we say that assignment α *satisfies* the circuit. It can be easily checked, in polynomial time, if a specific assignment satisfies a given circuit. We say that a circuit C is *satisfiable* if there exists an assignment on its input variables that satisfies C, and the corresponding problem is the following:

Circuit Satisfiability
Input: A boolean circuit C.
Question: Is circuit C satisfiable?

It is not hard to prove that Circuit Satisfiability is NP-complete, as 3-SAT is polynomially reducible to it (we do not describe the reduction in detail but we give an example in Figure 2.7). By defining the *weight* of an assignment to be the number of input nodes receiving value 1 from the assignment, we can define a parameterized version of Circuit Satisfiability:

Weighted Circuit Satisfiability (WCS)
Input: A boolean circuit C.
Parameter: A positive integer k.
Question: Is there an assignment of weight k that satisfies C?

The WCS problem can be solved in polynomial time for every fixed k using brute force, i.e., by trying all the $O(n^k)$ assignments of weight k and checking for each of them whether it satisfies the given circuit. The problem, though, does not seem to be fixed-parameter tractable as many "hard" parameterized, such as Clique, Independent Set and Dominating Set (all with the natural parameterization) can be reduced to it.

For a concrete example, we give a reduction from Independent Set to WCS: Let $G = (V, E)$ be a graph, k be a positive and let (G, k) be an input for the Independent Set problem. We construct one input node and one negation node for every vertex of G. We also add an or-node with indegree 2 for every edge in G. Finally, we add an and-node with indegree $|E|$. We connect every input node with its corresponding negation node and every or-node with the two negation nodes that correspond to the two vertices of G which form the edge related to this particular or-node. Finally, we connect all or-nodes to the and-node and we add an output node that gives the value of the and-node as the output of the circuit. (see also Figure 2.9).

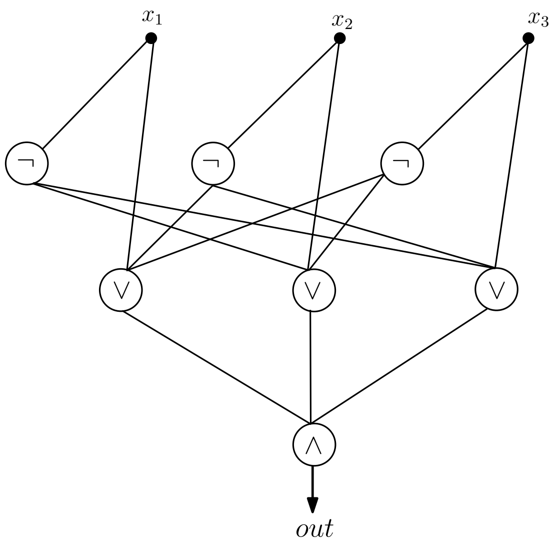


Figure 2.7: The circuit C that corresponds to the 3-CNF formula $\phi = (x_1 \vee \neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_2 \vee \neg x_3) \wedge (\neg x_1 \vee \neg x_2 \vee x_3)$. It is easy to see that, circuit C is satisfiable if and only if formula ϕ is satisfiable.

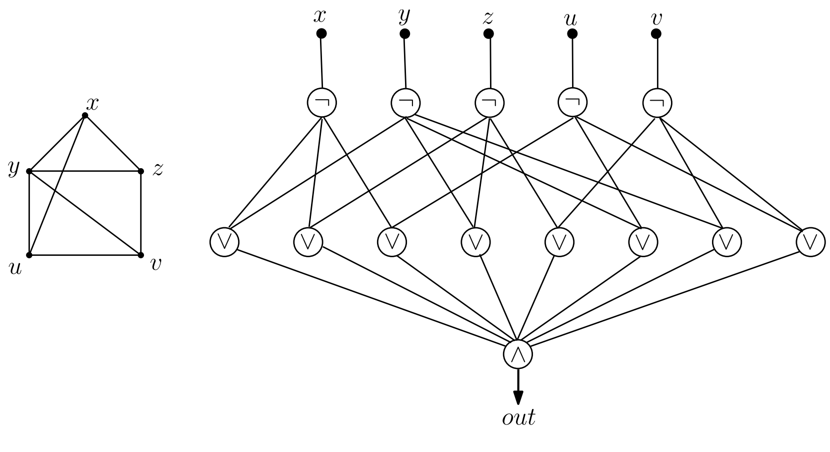


Figure 2.8: At the right: A graph with G with five vertices and eight edges. At the left: A decision circuit C (constructed from G as described by the reduction) satisfied by the independent sets of G. More specifically an independent set of size k in G corresponds to a satisfying assignment of weight k for C.

The W-hierarchy. We can define the levels of the W-hierarchy by restricting the WCS problem to different classes of decision circuits. For this, we distinguish between the *small nodes* and the *large nodes* of a circuit, where a node is characterised as small when its indegree is at most 2 and large otherwise. The *weft* of a circuit is the maximum number of large nodes on a path from an input node to the output node.

For a class of circuits $\mathcal{C}$, we denote by WCS$[\mathcal{C}]$ the restriction of the problem WCS where the input circuit is a member of $\mathcal{C}$. We denote by $\mathcal{C}_{t,d}$ the class of all circuits with weft at most t and depth at most d. We are now ready to define the levels of the W-hierarchy:

Definition 2.2.2. *For a positive integer t, we say that a parameterized problem Π belongs to the class* W$[t]$ *if there is a parameterized reduction from Π to* WCS$[\mathcal{C}_{t,d}]$ *for some positive integer d.*

Corollary 2.2.1. Independent Set*, parameterized by the size of the solution, is in* W$[1]$*.*

Proof. The corollary follows easily from Definition 2.2.2 and the reduction from Independent Set to WCS$[\mathcal{C}_{1,3}]$ (as it was described previously and depicted in Figure 2.9). □

It is also possible to prove that any problem in W$[1]$ can be reduced to Independent Set but the proof (given in [35]) is nontrivial and out of the scope of this thesis. Many other well-known problems can be proven to be complete for some level of the W-hierarchy. We give some of the most important in the next theorem. For a more extensive introduction to the W-hierarchy and fixed-parameter intractable problems, we refer the reader to [42], [36], and [22]. We give some complete problems for the first two levels of the W-hierarchy.

Theorem 2.2.1 ([33], [32], [35])**.** Dominating Set*,* Set Cover*, and* Hitting Set *are* W$[2]$*-complete.* Independent Set *and* Clique *are* W$[1]$*-complete.*

2.2.4 Kernelization

Another fundamental concept in the theory of Parameterized Complexity is *kernelization*. The idea of *preprocessing*, or *data reduction*, was known several years before the development of the theory of Parameterized Complexity. This idea can be roughly described as follows: Efficiently solve the "easy" parts of the instance (or even get rid of irrelevant parts) and reduce it to its hard "core" structure (which is hopefully much smaller than the initial instance) and then employ a slower (even exponential) exact algorithm to solve the reduced instance and obtain an answer.

But how to measure the effectiveness of such preprocessing routines? Suppose that we define such an algorithm as one that, given an instance, produces in polynomial time an equivalent instance that is at least one bit smaller than the initial one. Then, the existence of such an algorithm for an NP-hard problem would imply that P = NP, making the existence of such preprocessing routines for any NP-hard problem very unlikely.

In the framework of Parameterized Complexity a robust definition of preprocessing was given, by demanding that instances that are large compared to their parameter should be "shrunk", while instances that are small compared to the size of their parameter do not need any further preprocessing. Next we provide some formal definitions and we start by giving, once again, the formal definition of kernelization.

Definition 2.2.3. *Let $L \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. A* kernelization *(or* kernel*) for the parameterized problem L is an algorithm that given an instance (I, k) outputs, in time polynomial in $|I|$ and k, an instance (I', k') such that*

i) $(I, k) \in L$ *if and only if* $(I', k') \in L$*,*

ii) $|I'|$ *is bounded by a computable function f in k, and*

iii) k' *is bounded by a computable function g in k.*

The output (I', k') of the kernelization is called a kernel *and the function f is the size of the kernel. We say that a kernel is* polynomial *(resp.* linear*) if f is a polynomial (resp. linear) function.*

Next, we present the proof of a, somewhat surprising result, which states that a parameterized problem is in FPT if and only if it admits a kernel. This means that kernelization is an equivalent way of defining fixed-parameter tractability.

Theorem 2.2.2. *A parameterized problem Q is in* FPT *if and only if it admits a kernelization algorithm.*

Proof. Suppose that Q admits a kernelization algorithm, say $\mathcal{A}_1$, and let $\mathcal{B}_1$ be an algorithm for solving Q that runs in time $h(n)$, where n is the size of the input. We define algorithm $\mathcal{B}_2$, which on input (I, k) operates as follows: It runs algorithm $\mathcal{A}_1$ as a subroutine and obtains an equivalent instance (I', k'), where $|I'| \leq f(k)$ and $k' \leq g(k)$ for some polynomial functions f and g. Then it runs algorithm $\mathcal{B}_1$ with (I', k') as an input and outputs the its answer. It is easy to see that the running time of $\mathcal{B}_1$ is $O\big(h(f(k) + g(k)\big)$, which classifies problem Q in FPT.

Suppose now that Q is in FPT. Then, there exists an algorithm $\mathcal{A}_2$ deciding whether $(I, k) \in Q$ in time $f'(k) \cdot |I|^c$, for some computable function f' and a constant c. A kernelization algorithm for Q operates as follows: It runs algorithm $\mathcal{A}_2$ on (I, k) for at most $|I|^{c+1}$ steps. If $\mathcal{A}_2$ terminates it returns the obtained answer (YES or NO) as the output. If $\mathcal{A}_2$ does not terminate in $|I|^{c+1}$ steps, then it returns (I, k) itself as an output. Observe that, $\mathcal{A}_2$ not terminating in $|I|^{c+1}$ steps means that $f'(k) \cdot |I|^c > |I|^{c+1}$, which gives that $|I| < f'(k)$. Thus, we have that $|I| + k \leq f'(k) + k$ and the output of the kernelization algorithm on this case is a kernel of size at most $f'(k) + k$. $\square$

In order to make the notion of a kernelization more clear, we give a simple kernel for the natural parameterization of the Vertex Cover problem.

Example 1 (Vertex Cover)**.** *Let G be a graph and $S \subseteq V(G)$. We say that S is a* vertex cover *of G if every edge of G has at least one endpoint in S and we say that its* size *is $|S|$. Remember that the p-*Vertex Cover *problem asks, given a graph G and a positive integer k as input, to check whether there exists a vertex cover of size k in G.*

*We will describe a kernelization algorithm for p-*Vertex Cover*. We start by giving two* reduction rules *that will be used as steps of the algorithm:*

- **R1.** *If G contains an isolated vertex v (a vertex v with $deg_G(v) = 0$), delete v from G. The new instance is $(G \setminus v, k)$.*

- **R2.** *If there is a vertex v in G such that $deg_G(v) \geq k + 1$, delete v from G and decrement the value of the parameter by one. The new instance is $(G \setminus v, k - 1)$.*

The application of the reduction rules **R1** *and* **R2** *on an input* (G, k) *results in equivalent instances for the* p-Vertex Cover *problem: Obviously, an isolated vertex covers no edges of* G *so it will never be in a vertex cover and thus it can be removed. Additionally, if* G *contains a vertex* v *of degree more than* k*, then* v *should be in every vertex cover of size at most* k *(otherwise all the, at least* $k + 1$*, neighbours of* v *are needed to cover all the edges adjacent to* v*).*

The exhaustive application of rules **R1** *and* **R2** *completely removes the vertices of degree* 0 *and degree at least* $k + 1$*. Based on the observation that, a set of* k *vertices can cover at most* $k \cdot d$ *edges in a graph of maximum degree* d*, we prove the following lemma:*

Lemma 2.2.1. *If* (G, k) *is a* yes-*instance for* p-Vertex Cover *and reduction rules* **R1** *and* **R2** *are not applicable to* G*, then* $|V(G)| \leq k^2 + k$ *and* $|E(G)| \leq k^2$*.*

Proof. As rule **R1** is not applicable, G contains no isolated vertices, thus for any vertex cover S of G, every vertex in $G \setminus S$ is adjacent to some vertex of S. As rule **R2** is not applicable, every vertex of G has degree at most k. It follows that $|V(G \setminus S)| \leq (k+1)|S| \leq k^2 + k$, where the last inequality holds as the hypothesis of (G, k) being a yes-instance implies the existence of a vertex cover S of size at most k in G. Finally, any vertex of a vertex cover can cover at most k edges of G, thus $|E(G)| \leq k \cdot k = k^2$. □

We are now in the position to give our last reduction rule:

- **R3.** *Let* (G, k) *be an instance such that rules* **R1** *and* **R2** *are not applicable. If* $k > 0$ *and* $|V(G)| > k^2 + k$ *or* $|E(G)| > k^2$*, then* (G, k) *is a* no-*instance for* p-Vertex Cover*.*

We have built up to the following

Theorem 2.2.3. *Problem* p-Vertex Cover *admits a kernel with* $O(k^2)$ *vertices and* $O(k^2)$ *edges.*

Proof. We describe the steps of the kernelization algorithm: Apply reductions rules **R1** and **R2** until they cannot be applied anymore. Apply rule **R3** on the reduced instance, say (G', k'), and output a trivial no-instance if $|V(G')| > k^2 + k$ or $|E(G')| > k^2$ and (G', k') otherwise. The correctness of the algorithm is obvious from the previous analysis. □

The kernel for p-Vertex Cover that we just described is *quadratic* (as its size is quadratic with respect to the parameter k). Of course, we can always wonder if we can do better: Is it possible to construct a linear kernel for the same problem? A kernel with $2k$ vertices and $O(k^2)$ edges has been constructed (for more on kernelization algorithms for Vertex Cover see [1], [16], [14]) but there is complexity-theoretic evidence that these sizes cannot be improved any further. One of the most recent achievements of Parameterized Complexity, which will be briefly discussed in the subsequent section, is the construction of a theoretical framework for proving kernelization lower bounds.

2.2.5 Kernelization lower bounds

As we have proved in the previous section (Theorem 2.2.2), the existence of any kernelization algorithm is equivalent to the existence of a fixed-parameter algorithm for a problem. But is every kernelization algorithm *good*? It would be ideal if a linear (or even polynomial) sized kernel was guaranteed for every fixed-parameter tractable problem. All kind of techniques (even brute force) could then be applied to the shrunk instance, leading to an efficient solution. Unfortunately, there are many important parameterized problems which, under some plausible complexity theoretic assumptions, admit no polynomial kernels. We will gradually build to some of the main theorems starting with a concrete example, the natural parameterization of the Longest Path problem

p-Longest Path
Input: A graph G.
Parameter: A non-negative integer k.
Question: Is there a path of length k in G?

Assume that this problem admits a kernel with at most k^3 vertices, i.e., there is an algorithm $\mathcal{A}$ that, given an instance (G, k) for p-Longest Path, outputs an equivalent instance (G', k') such that $|V(G')|, k' \leq k^3$.

It is easy to observe that, the Longest Path problem has the following property regarding connectivity: If our input graph G is not connected then we have a positive answer if and only if we have a positive answer for at least one of the connected components of G.

Suppose we are given k^7 instances with the same parameter k, denoted by $(G_1, k), (G_2, k), \dots, (G_{k^7}, k)$, and let (H, k) be a new instance where H is the disjoint union of $G_1, G_2, \dots, G_{k^7}$. From our previous observation, it is clear that the answer to (H, k) is equal to the logical OR of the answers to the instances $(G_1, k), (G_2, k), \dots, (G_{k^7}, k)$.

By applying the kernelization algorithm $\mathcal{A}$ to (H, k) we get an equivalent instance (H', k') with $|V(H')|, k' \leq k^3$. We can encode H' in $\binom{k^3}{2}$ bits and k' in $3 \log k$ bits, giving a total of roughly $k^6/2 + 3 \log k$ bits for the encoding of our kernel (H', k'). But this number is even less than the number of the k^7 instances we started from, meaning that there exists at least one instance that we discarded and that during the kernelization process we "forgot" information about most of the instances. But is it possible, to evaluate and safely discard in polynomial time, instances of an NP-hard problem such as the Longest Path?

Next we provide some evidence that the behaviour of such a kernelization algorithm would indeed be suspicious, as we will link our skepticism to some well-established complexity-theoretic assumptions. We proceed with some definitions.

Definition 2.2.4 (co-NP/poly). *We say that a language L belongs to the complexity class* co-NP/poly *if there exists a Turing machine M and a sequence of strings $(a_n)_{n=0,1,2,\dots}$ (known as the* advice*) such that:*

- *Machine M, when given x with $|x| = n$ as an input, has access at a_n and has to decide whether $x \in L$. Machine M works in* co-deterministic polynomial time*, meaning that $x \in L$ if and only if the algorithm derives this conclusion for every possible run.*

- *The size of the advice is polynomially bounded by the size of the input, i.e. $|a_n| \leq p(n)$ for some polynomial $p(\cdot)$.*

Note that the advice strings a_n depend only on the size of the input or, in other words, inputs of the same size come with the same advice string.

Definition 2.2.5. *Let $L, R \subseteq \Sigma^*$ be two languages. An* OR-distillation *of L into R is an algorithm that, given a sequence of strings $x_1, x_2, \dots, x_t \in \Sigma^*$, runs in time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs one string $y \in \Sigma^*$ such that*

1. *$|y| \leq p(\max_{i=1}^{t} |x_i|)$ for some polynomial $p(\cdot)$.*

2. $y \in R$ if and only if there exists at least one index i such that $x_i \in L$.

The second condition asserts that the answer to the output, y, instance of R is equivalent to the logical OR of the answers to the input instances of L.

We are now ready to state and prove the crucial result of this section:

Theorem 2.2.4. *Let $L, R \subseteq \Sigma^*$ be two languages. If there exists an OR-distillation of L into R, then $L \in$ co-NP/poly.*

Proof. We can assume, without loss of generality, that $\Sigma = \{0, 1\}$. Let $\mathcal{A}$ be an OR-distillation of L into R and $p(\cdot)$ the polynomial (without loss of generality, we assume that it is nondecreasing) that bounds the length of the output of $\mathcal{A}$.
Let $K = p(n)$ so that algorithm $\mathcal{A}$ when running on a sequence of strings each of length at most n, outputs a string of length at most K. Let also $t = K + 1$. Thus, algorithm $\mathcal{A}$ maps the set $D = (\Sigma^{\leq n})^t$ of t-tuples of input strings into the set $\Sigma^{\leq K}$.

Let $A = L \cap \Sigma^{\leq n}$ (the yes-instances of L of length at most n) and $\overline{A} = \Sigma^{\leq n} \setminus L$ (the no-instances of L of length at most n). Similarly, let $B = R \cap \Sigma^{\leq K}$ and $\overline{B} = \Sigma^{\leq K} \setminus R$. From the second condition in 2.2.5 we have that $\mathcal{A}$ maps $(\overline{A})^t$ into $\overline{B}$ and $D \setminus (\overline{A})^t$ into B.

We will say that a string $x \in \Sigma^{\leq n}$ is *covered* by a string $y \in \Sigma^{\leq K}$ if there exists a t-tuple $(x_1, x_2, \ldots, x_t) \in D$ such that $x = x_i$ for some $i \in \{1, \ldots, t\}$, and $\mathcal{A}(x_1, x_2, \ldots, x_t) = y$. We will say that a set $X \subseteq \Sigma^{\leq n}$ is covered by a set $Y \subseteq \Sigma^{\leq K}$ if every element of X is covered by at least one element of Y. The following claim, which is the main argument of the proof, states that there exists a "small" subset of $\overline{B}$ that covers all the elements of $\overline{A}$.

Claim 2.2.1. *There is a set $Y \subseteq \overline{B}$ such that $|Y| \leq n + 1$ and Y covers $\overline{A}$.*

Proof of claim. We will consecutively choose strings $y_1, y_2, y_3, \ldots \in \overline{B}$ until (after at most n steps) the set $Y_i = \{y_1, y_2, \ldots, y_i\}$ covers $\overline{A}$. Let $S_i \subseteq \overline{A}$ be the strings that are not covered by Y_i. We initially have that $S_0 = \overline{A}$. Our construction will guarantee that $|S_i| \leq \frac{|\overline{A}|}{2^i}$, for every $i = 0, 1, \ldots$. Obviously, $|S_0| = |\overline{A}| \leq \frac{|\overline{A}|}{2^0}$. Since $\overline{A} \subseteq \Sigma^{\leq n}$ and therefore $|\overline{A}| \leq 2^{n+1}$, the construction will terminate after at most $n + 1$ steps.

We now describe how we choose a string y_i based on the knowledge of Y_{i-1}. As algorithm $\mathcal{A}$ maps $(S_{i-1})^t \subseteq (\overline{A})^t$ into $\overline{B}$ and $|\overline{B}| \leq |\Sigma^{\leq K}| < 2^{K+1}$, there exists (by the pigeonhole principle) some $y \in \overline{B}$ such that

$$|\mathcal{A}^{-1}(y) \cap (S_{i-1})^t| \geq \frac{|(S_{i-1})^t|}{2^{K+1}} = \left(\frac{|S_{i-1}|}{2}\right)^t.$$

If we set $y_i = y$, then every string from every tuple from the set $\mathcal{A}^{-1}(y) \cap (S_{i-1})^t$ is contained in $S_{i-1} \setminus S_i$ since it gets covered by y. Therefore, $\mathcal{A}^{-1}(y) \cap (S_{i-1})^t \subseteq (S_{i-1} \setminus S_i)^t$ and

$$\left(\frac{|S_{i-1}|}{2}\right)^t \leq |\mathcal{A}^{-1}(y) \cap (S_{i-1})^t| \leq |(S_{i-1} \setminus S_i)^t| = |S_{i-1} \setminus S_i|^t.$$

Hence, we get that $|S_{i-1} \setminus S_i| \geq \frac{|S_{i-1}|}{2}$, which gives $|S_i| \leq |S_{i-1}|/2$ and $|S_i| \leq \frac{|\overline{A}|}{2^i}$ (which is what we want) follows by induction. As the numbered of uncovered elements gets halved after every step, the construction will terminate after at most $n + 1$ steps and Y will be the current set Y_i. □

It remains to show how Claim 2.2.1 implies Theorem 2.2.4. As we want to prove that $L \in \text{co-NP/poly}$, we need to construct (a) an algorithm deciding membership in L in co-nondeterministic polynomial time and (b) a sequence of advice strings a_n for $n = 0, 1, 2, \ldots$, that will be given to the algorithm along with an input of size n.

Advice a_n will be an encoding of the covering set Y which is of polynomial size as Y contains at most $n+1$ strings, each of length at most $K = p(n)$. The algorithm works as follows: Given an input x with $|x| = n$, it tries to prove that $X \notin L$. If this is the case, then there exists a tuple $(x_1, \ldots, x_t) \in D$ such that $x = x_i$ for some $i \in \{1, \ldots, t\}$ and $\mathcal{A}(x_1, \ldots, x_t) = y \in Y$. The algorithm co-nondeterministically guesses this tuple, computes $\mathcal{A}(x_1, \ldots, x_t)$ and checks whether the result is contained in the advice string a_n. If $x \notin L$, then there exists at least one guess for which a string in Y will be computed. This string is a certificate that $x \notin L$ since $Y \subseteq \overline{B}$. If $x \in L$, then by the second condition of the definition of an OR-distillation, every tuple containing x is mapped to a string contained in B, so outside of Y whose encoding is the advice string a_n.
□

Now it is time to study the consequences of the theorem that we just proved:

Corollary 2.2.2. *If an* NP*-hard language $L \subseteq \Sigma^*$ admits an OR-distillation into some language $R \subseteq \Sigma^*$, then* NP $\subseteq$ co-NP/poly.

Proof. Let L' be a language in NP. We can check if $x \in L'$ in the following way: We apply the NP-hardness reduction from L' to L and obtain a string, say $f(x)$. We can now decide whether $f(x) \in L$ in co-NP/poly as implied by Theorem 2.2.4. As $x \in L'$ iff $f(x) \in L$, we have constructed an algorithm resolving membership in L' in co-NP/poly and therefore NP $\subseteq$ co-NP/poly. □

But why is NP $\subseteq$ co-NP/poly considered unlikely? From a complexity-theoretic point of view, the assumption that NP $\not\subseteq$ co-NP/poly may be viewed as a stronger variant of the NP $\neq$ co-NP hypothesis. It is known that NP $\subseteq$ co-NP/poly implies that $\Sigma_3^P =$ PH, i.e., the polynomial hierarchy collapses to its third level. Even though this collapse is not as dramatic as P $=$ NP, it is widely considered implausible.

But we have not yet stated anything about parameterized languages. How can these results be translated to kernelization lower bounds for parameterized problems?

Definition 2.2.6. *Let Σ be a finite alphabet. An equivalence relation $\mathcal{R}$ on the set of strings Σ^* is called a* polynomial equivalence relation *if the following two conditions hold:*

1. *There is an algorithm that given two strings $x, y \in \Sigma^*$ decides whether x and y belong to the same equivalence class in time polynomial in $|x|+|y|$.*
2. *Relation $\mathcal{R}$ restricted in $\Sigma^{\leq n}$ has at most $p(n)$ equivalence classes, for some polynomial $p(\cdot)$.*

Definition 2.2.7. *Let $L \subseteq \Sigma^*$ be a language and $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized language. We say that L* cross-composes *into Q, if there exists a polynomial equivalence relation $\mathcal{R}$ and an algorithm $\mathcal{A}$ (the* cross composition*), satisfying the following conditions: Algorithm $\mathcal{A}$, when given as input a sequence $x_1, x_2, \ldots, x_t$ of $\mathcal{R}$-equivalent strings, runs in time polynomial in $\sum_{i=1}^t |x_i|$, and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:*

1. *$k \leq p(\max_{i=1}^t |x_i| + \log t)$, for some polynomial $p(\cdot)$ and*

2. *$(y, k) \in Q$ if and only if there exists at least one index i such that $x_i \in L$.*

Note that this definition is similar to the one of OR-distillation, but here it is only the output parameter that has to be "small", while the string y can even have the size of the concatenation of the input instances.

We need one last definition before stating the main theorem of this section.

Definition 2.2.8. *A* polynomial compression *of a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$ into a language $R \subseteq \Sigma^*$, is an algorithm that takes as input an instance $(x, k) \in \Sigma^* \times \mathbb{N}$, works in time polynomial in $|x| + k$, and outputs a string y such that:*

1. *$|y| \leq p(k)$, for some polynomial $p(\cdot)$ and*
2. *$y \in R$ if and only if $(x, k) \in Q$.*

This definition is similar to the one of a kernel, and indeed a polynomial kernel is also a polynomial compression by treating the output kernel as an instance of the unparameterized version of Q. The main difference between the two is that a polynomial compression is allowed to output an instance of any language R, even an undecidable one.

In what follows, we prove that given a cross composition of a language $L \in \Sigma^*$ into a parameterized language $Q \subseteq \Sigma^* \times \mathbb{N}$ and a polynomial compression of Q into some language R, we can construct an OR-distillation of L into R. The proof, despite being a bit lengthy, does not contain any important ideas and is just a careful application of the definitions.

Theorem 2.2.5. *If an* NP*-hard language L cross-composes into a parameterized language Q, then Q does not admit a polynomial compression unless* NP $\subseteq$ co-NP/poly*.*

Proof. Let $\mathcal{A}$ be a cross-composition of L into Q, $p_0(\cdot)$ be the polynomial bounding the parameter of the output instance of $\mathcal{A}$ and let $\mathcal{R}$ be the polynomial equivalence relation used in the cross-composition.

Assume also that Q admits a polynomial compression, $\mathcal{C}$, into some language R and let OR(R) be the language consisting of strings of the form s_1#s_2# …#s_q, such that $s_i \in R$ for at least one index $i \in \{1, \ldots, q\}$ (# is a special character that is added to Σ). We will conclude that $\mathsf{NP} \subseteq$ co-NP/poly by constructing an OR-distillation of L into OR(R) and using Corollary 2.2.2.

Let $x_1, x_2, \ldots, x_t$ be the sequence of input strings and let $n = \max_{i=1}^{t} |x_i|$. We first apply some (polynomial) preprocessing: find and remove all duplicates in the sequence $x_1, x_2, \ldots, x_t$. The number of the remaining strings is at most the number of different string over Σ of length at most n. Therefore,

$$t = \sum_{i=0}^{n} |\Sigma|^i \leq |\Sigma|^{n+1}.$$

Hence, after the removal of duplicates, we have that $\log t = O(n)$.

Partition the sequence $x_1, x_2, \ldots, x_t$ into equivalent classes, say $\mathcal{C}_1, \mathcal{C}_2, \ldots, \mathcal{C}_q$, with respect to the relation $\mathcal{R}$. By the Definition 2.2.6 of a polynomial equivalence relation, this can be done in polynomial time and $q \leq p_1(n)$, for some polynomial $p_1(\cdot)$.

For $j = 1, 2, \ldots, q$, apply the cross-composition $\mathcal{A}$ to the strings that comprise the equivalence class Q_j, obtaining an instance (c_j, k_j) such that $k_j \leq p_0(\max_{x \in C_j} |x| + \log |C_j|)$, polynomially bounded in n and $\log t$, and $(c_j, k_j) \in Q$ if and only if there exists some $x \in C_j$ such that $x \in L$. As $t = O(n)$, there exists some polynomial $p_2(\cdot)$ such that $k_j \leq p_2(n)$, for every $j \in \{1, 2, \ldots, q\}$.

Now apply the compression algorithm $\mathcal{C}$ to each instance (c_j, k_j), thus obtaining a string s_j such that $s_j \in R$ if and only if $(c_j, k_j) \in Q$. As $k_j \leq p_2(n)$ and $|s_j| \leq p_3(k_j)$ for some polynomial $p_3(\cdot)$, we infer that $|s_j \leq p_4(n)$ for some polynomial $p_4(\cdot)$.

We conclude the construction by merging all strings s_j into one instance $s = s_1 \# s_2 \# \ldots \# s_q$. It is clear that, $s \in$OR(R) if and only if there exists some $i \in \{1, 2 \ldots, t\}$ such that $x_i \in L$. This means that the second condition in the Definition 2.2.5 of OR-distillation is satisfied. For the first condition, we have that $q \leq p_1(n)$ and $|s_j| \leq p_4(n)$ and therefore, $|s| \leq p_1(n) \cdot (p_4(n) + 1) - 1$ (for a visualisation of the proof, see Figure **??**

□

As we have already discussed, polynomial compression can be replaced by polynomial kernel in the previous theorem, making it a useful tool for proving kernelization lower bounds for NP-hard problems (under the assumption NP $\not\subseteq$ co-NP/poly).

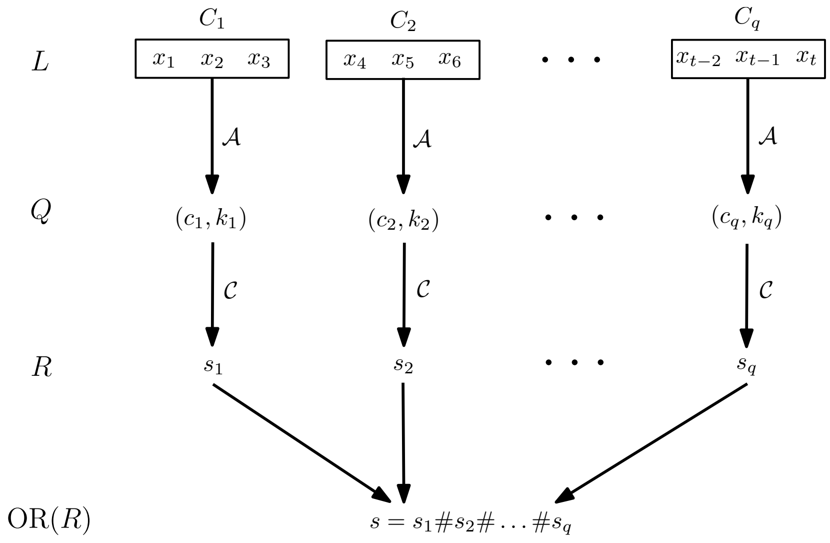


Figure 2.9: This figure gives an intuition about how we can create an OR-distillation for language L, given a cross-composition $\mathcal{A}$ of L into (the parameterized language) Q and a polynomial compression $\mathcal{C}$ of Q into the language R.

We conclude this section by applying Theorem 2.2.5 for (formally now) proving that it is unlikely that p-Longest Path admits a polynomial kernel.

Example 2. *We will describe a cross composition of the* NP*-hard problem* Hamilton Path *(which, given a graph G, asks whether there exists a path in G that meets all of its vertices) to the parameterized problem* p-Longest Path*:*
For the relation $\mathcal{R}$, we put into one equivalence class, say $\mathcal{C}_0$ all the strings of Σ^* that do not encode any graph and we partition the remaining instances with respect to the number of vertices of the graph, denoting by $\mathcal{C}_n$ the class that includes the n-vertex graph, for every $n \geq 1$.

The cross-composition algorithm when given a sequence of graphs in $\mathcal{C}_0$, returns a trivial no*-instance for the* p-Longest Path*. Given a sequence of n-vertex graphs, it returns the encoding of their disjoint union together with the parameter $k = n$.*

It is not hard to construct an encoding such that condition 1 of Definition

2.2.7 holds. Condition 2 is satisfied as well, as the disjoint union of any number of n-vertex graphs has an path on n vertices if and only if at least one of the input graphs contains a Hamiltonian path.

It turns out that we can define AND-distillation by replacing condition 2 in Definition 2.2.5 with the requirement that $y \in R$ if and only if for all i we have $x_i \in L$. Similarly, AND-cross-composition can be defined by replacing condition 2 in Definition 2.2.7 by requiring that $(y, k) \in Q$ if and only if if for all i we have $x_i \in L$. It is not hard to prove an analogue of Theorem 2.2.5 where OR-distillations are replaced by AND-distillations. However, the arguments in the proof of Theorem 2.2.4 break apart when trying to translate them to the AND-setting. Fortunately, a, much more difficult, proof for this appears in [37]. We will give the formal definitions in Section 7 where we prove that it is unlikely that Cyclability, parameterized by k, admits a polynomial kernel when restricted to the class of planar graphs.

2.3 Monadic second-order logic

One way to classify a computational problem in a complexity class (either a classical or a parameterized one) is to place it in the framework of Logic and, more precisely, to express the problem in the formalism of some specific language in the framework of Logic. One of the most well-known results, which illustrates this parallelism between logic and complexity (the main subject of *descriptive complexity theory*), is the following theorem which was proved by Ronald Fagin in his doctoral thesis in 1973 (also appears in [75]).

Theorem 2.3.1 (Fagin's Theorem)**.** *The class of all graph-theoretic properties expressible in existential second-order logic is precisely* NP*.*

We do not go into any further details for explaining Fagin's theorem as it is not directly related to this thesis. We mention it because we think that is can work as an introduction to the concept of representing a problem as a formula in the frame of some logic language.

We will instead present a result, namely *Courcelle's theorem*, that is strongly related to Parameterized Complexity Theory and is similar to Fagin's theorem, in the sense that it exposes the correlation between a variant of second-order logic and the class FPT (a significant difference is that Courcelle's theorem is *algorithmic* and is considered to be the archetype of algorithmic metathorems [20]). For doing so, we next present a brief description of Monadic Second-Order Logic (shorter MSO_2) for graphs.

Monadic Second-Order Logic. The syntax of *Monadic Second Order Logic* (MSO_2) requires an infinite number of individual variables for vertices and edges (we usually use letters $x, y, z. \ldots$) and an infinite number of set variables for sets of vertices and sets of edges (we usually use capital letters $X, Y, Z, \ldots$). It also includes the logical connectives $\wedge$ (*conjunction*), $\vee$ (*disjunction*), $\neg$ (*negation*) (we can also include $\rightarrow$ (*implication*) and $\leftrightarrow$ (*bi-implication*) to make the formalism simpler when writing formulas) and the quantifiers $\exists$ (*existential quantifier*) and $\forall$ (*universal quantifier*) that can be applied to the variables. MSO_2 additionally includes the following five binary relations:

1. $u \in U$, where u is a vertex variable, U is a vertex-set variable and the interpretation is the obvious.

2. $d \in D$, where u is a edge variable, D is a edge-set variable and the interpretation is the obvious.

3. $\texttt{inc}(d, u)$, where d is an edge variable, u is a vertex variable and the interpretation is that the edge d is incident on the vertex u.

4. $\texttt{adj}(v, u)$, where v and u are vertex variables and the interpretation is that v and u are adjacent vertices.

5. $x = y$ (resp. $X = Y$) and the interpretation is equality of variables (resp. set variables).

The semantics of MSO_2 are defined in the obvious way according to the interpretations of the previous binary relations.

Let π be a graph property, G be a graph and $\mathcal{G}$ be the class of all graphs. If G has property π we write $\pi(G)$. We say that π is *expressible in* MSO_2 if there exists some MSO_2 formula ϕ_π such that

$$(\forall G \in \mathcal{G})\,[\,G \models \phi_\pi \;\Leftrightarrow\; \pi(G)\,]$$

To make this definition more clear we give an example of an MSO_2 expressible graph property.

Example 3 (3-colourability). *A graph $G = (V, E)$ is called* 3-colourable *if its vertex set V can be partitioned into three subsets X_1, X_2, X_3 such that there exists no edge $e = \{u, v\} \in E$ such that $u, v \in X_i$ for some $i \in \{1, 2, 3\}$, i.e. if we consider the vertices of X_1, X_2, X_3 to have three distinct colours there exists no bichromatic edge in G.*

To express this property in MSO_2*, it is sufficient to quantify the existence of three subsets X_1, X_2, X_3 which form a partition of V and each of them is an independent set.*

$$\textbf{3colourability} = \exists_{X_1,X_2,X_3\subseteq V}\ \textbf{partition}(X_1, X_2, X_3) \\ \wedge\ \textbf{indp}(X_1) \wedge \textbf{indp}(X_2) \wedge \textbf{indp}(X_3)$$

In order for the **3colourability** *formula to be short and not too involved, we used two auxiliary subformulas, namely* **partition** *(verifying that (X_1, X_2, X_3) is a partition of the vertex set) and* **indp** *(verifying that a given vertex set is an independent set) which we define as:*

$$\begin{aligned}\textbf{partition}(X_1, X_2, X_3) = &\forall_{v\in V} \big[(v \in X_1 \wedge v \notin X_2 \wedge v \notin X_3) \\ &\vee (v \notin X_1 \wedge v \in X_2 \wedge v \notin X_3) \\ &\vee (v \notin X_1 \wedge v \notin X_2 \wedge v \in X_3)\big]\end{aligned}$$

$$\textbf{indp}(X) = \forall_{u,v\in V}\ \neg\mathtt{adj}(u, v)$$

It is easy to confirm that these two last formulas express the desired properties and **3colourablity** *formula is an expression of the property of a graph being 3-colourable, in* MSO_2.

We are now in the position to state the celebrated theorem of Courcelle (see also [7] and [13])

Proposition 2.3.1 (Courcelle's theorem [21, 20]). *Given a graph G and an* MSO_2 *formula ϕ describing a graph property π, and parameterizing by* **tw**(G) *(the treewidth of G) and by $|\phi|$ (the size of the formula ϕ), there exists a computable function f such that it can be determined in time $f(\textbf{tw}(G), |\phi|) \cdot n^{O(1)}$ whether graph G has property π.*

The intuition behind Courcelle's theorem is that, if the property of interest can be expressed in a special fragment of second-order logic, namely MSO_2, then the problem of deciding whether a graph G has this property admits an algorithmic solution where any"heavy" time requirements depend only on the treewidth of G (the size of the formula can be ignored because it is usually small) and the contribution of the size of the whole input can be restricted to be just linear. On the other hand, function f is huge (and this is unavoidable unless P$=$NP, as indicated by Gröhe and Frick in [44]) and the theorem cannot directly be used, at least in its full strength, for practical purposes.

There have been some efforts to construct practical algorithms for implementing Courcelle's theorem (see [64], [65] for more details and [62] for some evaluation data) and the future seems promising. For further details on the topic, we refer the reader to Chapter 13 of [36] and to Chapter 7 of [22].

The reason why we introduced the Monadic Second Order Logic and presented the theorem of Courcelle, is that Cyclability is MSO_2-expressible (as we prove in Chapter 4) and therefore is in FPT when parameterized by the treewidth of the input graph. We remind that being able to solve Cyclability on graphs of bounded treewidth is a crucial ingredient of our algorithm.

However, the heavy time requirements that arise from the use of Courcelle's theorem would render our algorithm completely impractical, even though it would still be an FPT-algorithm. That is why, after giving a MSO_2 formula that expresses the property of a graph being k-cyclable, we also construct (in Chapter 4) a dynamic programming routine for solving the Cyclability problem given a tree decomposition of the corresponding graph. This allows the final running time of our algorithm to be more attractive.

2.4 Cycles, walls and annuli

As we have already mentioned, when we gave an overview of our algorithm for solving Cyclability on planar graphs (Section 1.3 of Chapter 1), we deal with graphs having "large" treewidth by finding either a colour irrelevant vertex or a problem irrelevant vertex. In order to do this we have to exploit that the treewidth being large implies the existence of a large grid-minor in our graph. This in turn, implies the existence of a large bidimensional subgraph (a *wall*) and of another bidimensional structure, which we call *railed annulus*.

In this section, we give formal definitions for this structures, building towards Chapter 3, where we prove some combinatorial results about cyclic linkages which we uses later (in the analysis of the algorithm) to justify the existence of irrelevant vertices.

Concentric cycles. Let G be a graph embedded in the sphere $\mathbb{S}_0$ and let $\mathcal{D} = \{D_1, \ldots, D_r\}$, be a sequence of closed disks in $\mathbb{S}_0$. We call $\mathcal{D}$ *concentric* if $D_1 \subseteq D_2 \subseteq \cdots \subseteq D_r$ and no point belongs to the boundary of two disks in $\mathcal{D}$. We call a sequence $\mathcal{C} = \{C_1, \ldots, C_r\}$, $r \geq 2$, of cycles of G *concentric* if there exists a concentric sequence of closed disks $\mathcal{D} = \{D_1, \ldots, D_r\}$, such that C_i is the boundary of D_i, $i \in \{1, \ldots, r\}$. For $i \in \{1, \ldots, r\}$, we set $\overline{C}_i = D_i$, $\mathring{C}_i = \overline{C}_i \setminus C_i$, and $\hat{C}_i = G \cap D_i$ (notice that $\overline{C}_i$ and $\mathring{C}_i$ are sets while $\hat{C}_i$ is a subgraph of G). Given i, j with $i \leq j - 1$, we denote by $\hat{A}_{i,j}$ the graph $\hat{C}_j \setminus \mathring{C}_i$. Finally, given a $q \geq 1$, we say that a vertex set $R \subseteq V(G)$ is q-*dense* in $\mathcal{C}$ if, for every $i \in \{1, \ldots, r - q + 1\}$, $V(\hat{A}_{i,i+q-1}) \cap R \neq \emptyset$.

It is not hard to observe (we give a formal proof in Chapter 3), that the existence of a large grid-minor in a planar graph G implies that there exists an embedding of G such that a large sequence of concentric cycles is formed. Actually, these concentric cycles are crossed by paths forming what we call a *railed annulus*:

Railed annulus. Let $r \geq 2$ and $q \geq 1$ be two integers and let G be a graph embedded on the sphere $\mathbb{S}_0$. A (r, q)-*railed annulus* in G is a pair $(\mathcal{C}, \mathcal{W})$ such that $\mathcal{C} = \{C_1, C_2, \ldots, C_r\}$ is a sequence of r concentric cycles that are all intersected by a sequence $\mathcal{W}$ of q paths $W_1, W_2, \ldots, W_q$ (called *rails*) in such a way that $\bigcup \mathcal{W} \subseteq \hat{A}_{1,r}$ and the intersection of a cycle and a rail is always connected, that is, it is a (possibly trivial) path (see Figure 2.10 for an example).

As it is more convenient to work with subgraphs rather than minors, we translate the existence of a large grid-minor in a planar graph G to the existence of a large subgraph, which we call a *subdivided wall*, in G.

Walls and subdivided walls. Let h be a integer and $h \geq 1$. A *wall of height* h is the graph obtained from a $((h + 1) \times (2 \cdot h + 2))$-grid with vertices (x, y), $x \in \{1, \ldots, 2 \cdot h + 4\}$, $y \in \{1, \ldots, h + 1\}$, after the removal of the "vertical"

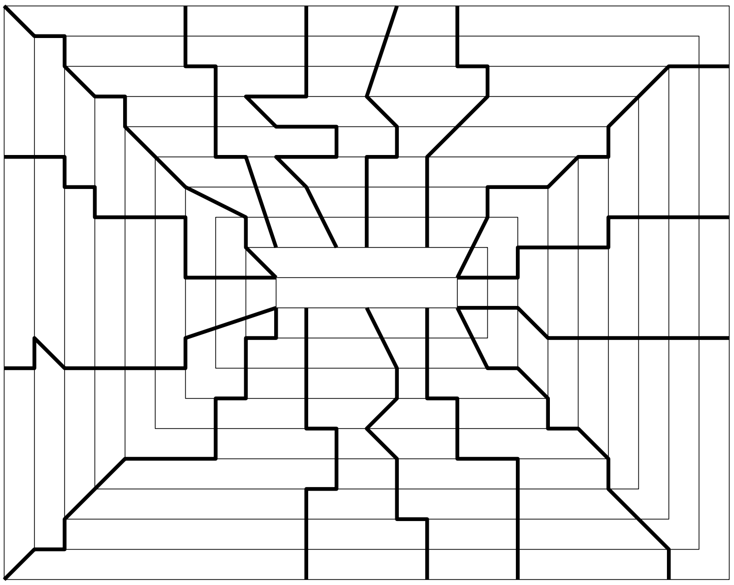


Figure 2.10: A (10,15)-railed annulus.

edges $\{(x, y), (x, y + 1)\}$ for odd $x + y$, and then the removal of all vertices of degree 1. We denote such a wall by W_h. A *subdivided wall of height h* is a wall obtained from W_h after replacing some of its edges by paths without common internal vertices (see Fig. 2.12 for an example). The *perimeter* P_W of a subdivided wall W is the cycle defined by its boundary. Let $C_2 = P_W$ and let C_1 be any cycle of W that has no common vertices with P_W. Notice that $\mathcal{C} = \{C_1, C_2\}$ is a sequence of concentric cycles in G. We define the *compass* K_W *of* W *in* G as the graph $\hat{C}_2$.

Layers of a wall. Let W be a subdivided wall of height $h \geq 2$. The *layers* of W are recursively defined as follows. The first layer, J_1, of W is its perimeter. For $i \in \{2, \dots, \lfloor \frac{h}{2} \rfloor\}$, the i-th layer, J_i, of W is the perimeter of the subwall W' obtained from W by removing its perimeter and repetitively removing occurring vertices of degree 1. We denote the *layer set* of W by $\mathcal{J}_W = \{J_1, \dots, J_{\lfloor \frac{h}{2} \rfloor}\}$

Our last definition is a rather technical one that makes the treatment of concentric cycles easier. The *tightness* of a sequence of concentric cycles in a plane graph implies that it cannot be extended into a bigger one by adding another cycle of the graph, which lies in the interior of the outer cycle.

Tight concentric cycles. Let G be a graph embedded in the sphere $\mathbb{S}_0$. A sequence $\mathcal{C} = \{C_1, \ldots, C_r\}$ of concentric cycles of G is *tight* in G, if

- C_1 is *surface minimal*, i.e., there is no closed disk D of $\mathbb{S}_0$ that is properly contained in $\overline{C_1}$ and whose boundary is a cycle of G;
- for every $i \in \{1, \ldots, r-1\}$, there is no closed disk D such that $\overline{C}_i \subset D \subset \overline{C}_{i+1}$ and such that the boundary of D is a cycle of G.

See Figure 2.11 for a an example of the tightness definition.

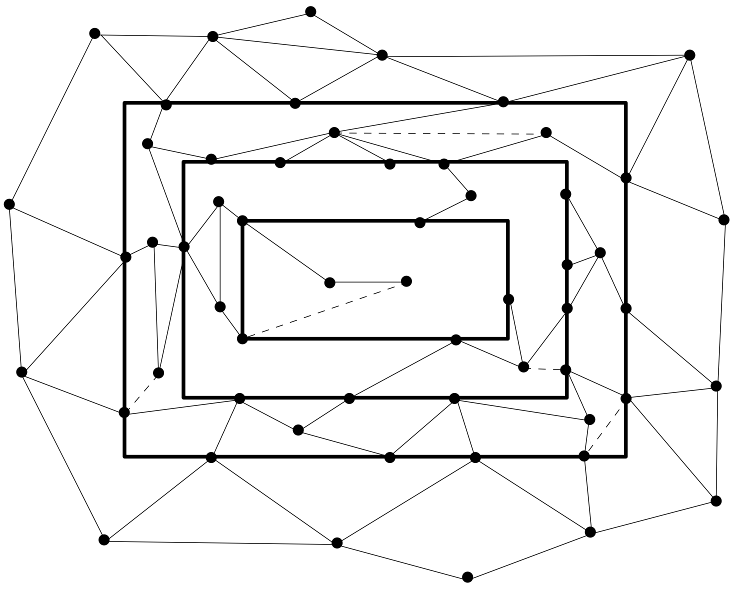


Figure 2.11: A sequence of three tight concentric cycles. The addition of any of the dashed edges makes the sequence non-tight.

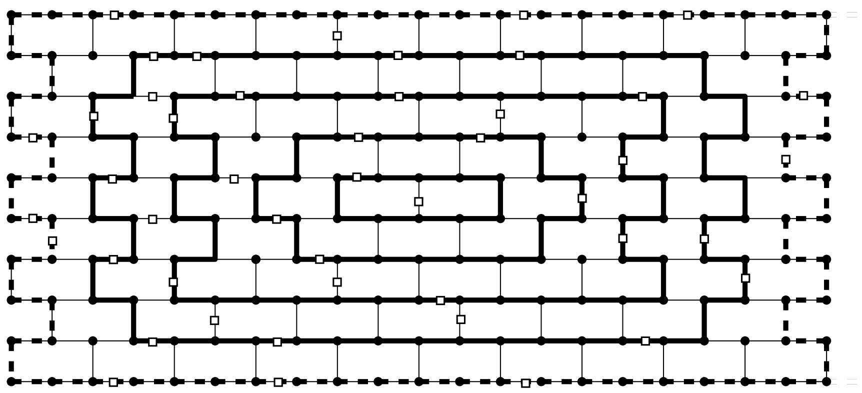


Figure 2.12: A subdivided wall of height 9. The white squares represent the subdivision vertices. The bold curves are its layers and the bold-dashed curve is its perimeter.

Given a graph G we denote by $\mathbf{gw}(G)$ the maximum integer h for which G contains a subdivided wall of height h as a subgraph. The next lemma follows easily by combining results in [43],[53], and [79] and relates the treewidth of a planar graph with the maximum height of a subdivided wall in it.

Lemma 2.4.1. *If G is a planar graph, then $\mathbf{tw}(G) \leq 9 \cdot \mathbf{gw}(G) + 1$.*

We are now ready to proceed to the next chapter where we prove some combinatorial results about cyclic linkages.

CHAPTER 3

COMBINATORICS OF CYCLIC LINKAGES

In this section, we generalize the notion of a linkage that was introduced by Robertson and Seymour in [82]. Specifically, we introduce the notion of *graph linkages* and study some combinatorial properties of a specific kind of graph linkage, namely a *cyclic linkage*. Cyclic linkages are important for us because they represent the kind of structure we are looking for in a graph when trying to estimate its cyclability.

Our goal is to find a relation between the existence of a unique (in a way) cyclic linkage in a graph and the treewidth of this graph, proving an analogue of the Unique linkage theorem (in [82]) of Robertson and Seymour, but for cyclic linkages). This will enable us to justify the existence of irrelevant vertices for the Cyclability problem on planar graphs, when the treewidth of the input graph is sufficiently large.

3.1 Graph linkages and cheap graph linkages

Graph Linkages. Let G be a graph. A *graph linkage* in G is a pair $\mathcal{L} = (H, T)$ such that H is a subgraph of G without isolated vertices and T is a subset of the vertices of $H,$ called *terminals* of $\mathcal{L},$ such that every vertex of H with degree different than 2 is contained in T. The set $\mathcal{P}(\mathcal{L}),$ which we call *path set of* the graph linkage $\mathcal{L}$, contains all paths of H whose endpoints are in T and do not have any other vertex in T. The *pattern* of L is the graph

$$(T, \{\{s,t\} \mid \mathcal{P}(\mathcal{L}) \text{ contains a path from } s \text{ to } t \text{ in } H\}).$$

Two graph linkages of G are *equivalent* if they have the same pattern and are *isomorphic* if their patterns are isomorphic. A graph linkage $\mathcal{L} = (H,T)$ is called *weakly vital* (reps. *strongly vital*) in G if $V(H) = V(G)$ and there is no other equivalent (resp. isomorphic) graph linkage that is different from $\mathcal{L}$. Clearly, if a graph linkage $\mathcal{L}$ is strongly vital then it is also weakly vital. We call a graph linkage $\mathcal{L}$ *linkage* if its pattern has maximum degree 1 (i.e., it consists of a collection of paths), in which case we omit H and refer to the linkage just by using $\mathcal{L}$. We also call a graph linkage $\mathcal{L}$ *cyclic linkage* if its pattern is a cycle. For an example of distinct types of cyclic linkages, see Figure 3.1.

Notice that there is a critical difference between equivalence and isomorphism of linkages. To see this, suppose that $\mathcal{L} = (C,T)$ is a cyclic linkage of a graph G and let A_G be the set of all cyclic linkages that are isomorphic to $\mathcal{L}$, while B_G is the set of all cyclic linkages that are equivalent to $\mathcal{L}$. Notice that the cycles in the cyclic linkages of A_G should meet the terminals in the same cyclic order. On the contrary, the cycles of the cyclic linkages of B_G may meet the terminals in any possible cyclic ordering. Consequently $A_G \subseteq B_G$. For example, if $\mathcal{L} = (C,T)$ is the cyclic linkage of the graphs in Figure 3.1, then $|A_{G_1}| = 1$, $|B_{G_1}| = 1$, $|A_{G_2}| = 1$, $|B_{G_2}| = 12$, $|A_{G_2}| = 4$, $|B_{G_3}| = 28$.

CGL-configurations. Let G be a graph embedded on the sphere $\mathbb{S}_0$. Then, we say that a pair $\mathcal{Q} = (\mathcal{C}, \mathcal{L})$ is a *CGL-configuration* of *depth* r if $\mathcal{C} = \{C_1, \dots, C_r\}$ is a sequence of concentric cycles in G, $\mathcal{L} = (H,T)$ is a graph linkage in G, and $T \cap V(\hat{C}_r) = \emptyset$, i.e., all vertices in the terminals of $\mathcal{L}$ are outside $\overline{C_r}$. The *penetration* of $\mathcal{L}$ in $\mathcal{C}$, $p_{\mathcal{C}}(\mathcal{L})$, is the number of cycles of $\mathcal{C}$ that are intersected by the paths of $\mathcal{L}$ (when $\mathcal{L} = (C,S)$ is cyclic we will sometimes refer to the penetration of $\mathcal{L}$ as the penetration of cycle C). We say that $\mathcal{Q}$ is *touch-free* if for every path $P \in \mathcal{L}$, the number of connected components of $P \cap C_r$ is not 1. See figure 3.2 for an example of a CGL-configuration.

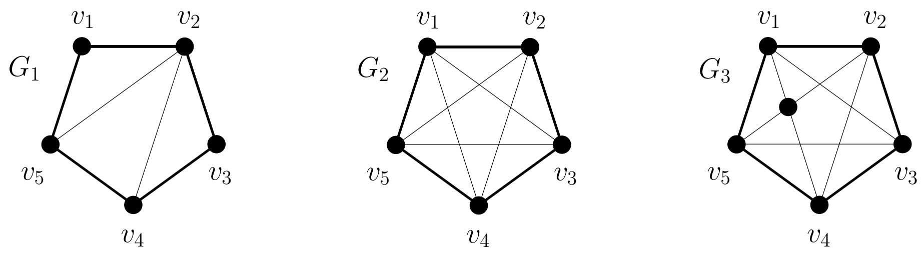


Figure 3.1: Three graphs G_1, G_2, and G_3. In each graph the bold edges define the cycle $C = (\{v_1, \ldots, v_5\}, \{\{v_1, v_2\}, \ldots, \{v_4, v_5\}, \{v_5, v_6\}\})$ where $T = V(C)$. Consider the cyclic linkage $\mathcal{L} = \{C, T\}$ where $T = V(C)$. $\mathcal{L}$ is a weakly vital linkage in G_1 and G_2 while it is not a weakly vital linkage in G_3. Moreover, $\mathcal{L}$ is a strongly vital linkage in G_1 while it is not a strongly vital linkage neither in G_2 nor in G_3.

Cheap graph linkages. Let G be a graph embedded on the sphere $\mathbb{S}_0$, let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be a sequence of cycles in G, and let $\mathcal{L} = (H, T)$ be a graph linkage where $T \subseteq V(G \setminus \hat{C}_r)$ (notice that $(\mathcal{C}, \mathcal{L})$ is a CGL-configuration). We define function c which maps graph linkages of G to non-negative integers such that

$$c(\mathcal{L}) = |E(\mathcal{L}) \setminus \bigcup_{i \in \{1,\ldots,r\}} E(C_i)|.$$

A graph linkage $\mathcal{L}$ of G is *$\mathcal{C}$-strongly cheap* (resp. *$\mathcal{C}$-weakly cheap*), if $T(\mathcal{L}) \cap \hat{C}_r = \emptyset$ and there is no other isomorphic (resp. equivalent) graph linkage $\mathcal{L}'$ such that $c(\mathcal{L}) > c(\mathcal{L}')$. Obviously, if $\mathcal{L}$ is $\mathcal{C}$-strongly cheap then it is also $\mathcal{C}$-weakly cheap.

Tilted grids. Let G be a graph. A *tilted grid* of G is a pair $\mathcal{U} = (\mathcal{X}, \mathcal{Z})$ where $\mathcal{X} = \{X_1, \ldots, X_r\}$ and $\mathcal{Z} = \{Z_1, \ldots, Z_r\}$ are both sequences of $r \geq 2$ vertex-disjoint paths of G such that

- for each $i, j \in \{1, \ldots, r\}$ $I_{i,j} = X_i \cap Z_j$ is a (possibly edgeless) path of G,
- for $i \in \{1, \ldots, r\}$ the subpaths $I_{i,1}, I_{i,2}, \ldots, I_{i,r}$ appear in this order in X_i,

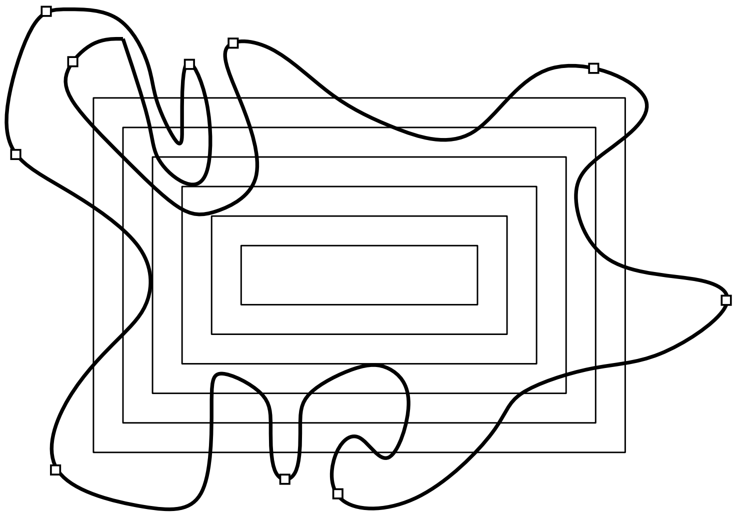


Figure 3.2: A CLG-configuration $\mathcal{Q} = (\mathcal{C}, \mathcal{L})$ with $\mathcal{L} = (H, T)$. Here, $\mathcal{C}$ is a sequence of six concentric cycles, H (the **bold** curve) is a cycle (thus $\mathcal{L}$ is a cyclic linkage) and T is represented by the set of squares. The penetration of $\mathcal{L}$ in $\mathcal{C}$ is 4 and $\mathcal{Q}$ is touch-free.

- for $j \in \{1, \ldots, r\}$ the subpaths $I_{1,j}, I_{2,j}, \ldots, I_{r,j}$ appear in this order in Z_j,
- $E(I_{1,1}) = E(I_{1,r}) = E(I_{r,1}) = E(I_{r,r}) = \emptyset$,
- the graph $G^*_{\mathcal{U}}$ taken from the graph $G_{\mathcal{U}} = (\bigcup_{i \in \{1,\ldots,r\}} X_i) \cup (\bigcup_{i \in \{1,\ldots,r\}} Z_i)$ after contracting all edges in $\bigcup_{(i,j) \in \{1,\ldots,r\}^2} I_{i,j}$ is isomorphic to the $(r \times r)$-grid.

We refer to the cardinality r of $\mathcal{X}$ (or $\mathcal{Z}$) as the *capacity* of $\mathcal{U}$.

Tidy tilted grids. Given a plane graph G and a graph linkage $\mathcal{L} = (H, T)$ of G we say that a tilted grid $\mathcal{U} = (\mathcal{X}, \mathcal{Z})$ of G is an *$\mathcal{L}$-tidy tilted grid* of G if $T \cap D_{\mathcal{U}} = \emptyset$ and $D_{\mathcal{U}} \cap \mathcal{L} = \bigcup \mathcal{Z}$ where $D_{\mathcal{U}}$ is the closed interior of the perimeter of $G_{\mathcal{U}}$ (for an example see Figure 6).

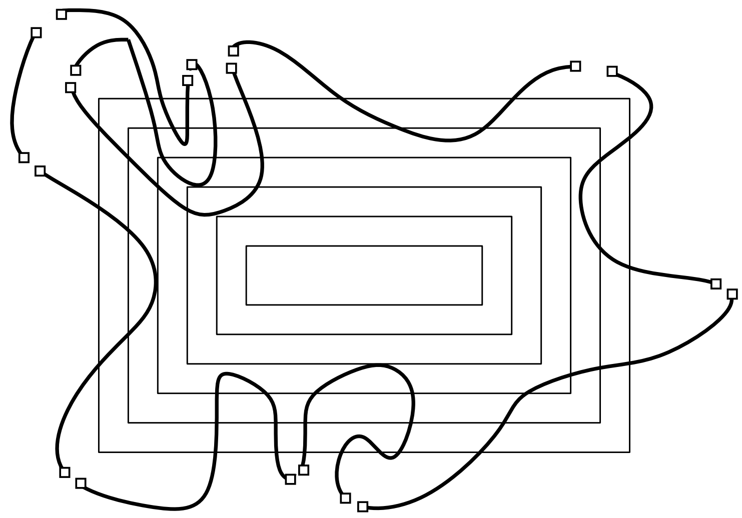


Figure 3.3: The linkage that corresponds to the cyclic linkage depicted in Figure 3.2.

In order to be able to use some of the known results for linkages for our purposes, we associate any cyclic linkage with a linkage in the following way.

From graph linkages to linkages. Let G be a graph and let $\mathcal{L} = (H, T)$ be a graph linkage of G. We denote by $G_{\mathcal{L}}$ the graph obtained by subdividing all edges of G incident to terminals and then removing the terminals. Similarly, we define $\mathcal{L}^* = (H^*, T^*)$ so that H^* is the graph obtained by subdividing all edges incident to terminals, removing the terminals, and considering as terminals the subdivision vertices. Notice that $\mathcal{L}^*$ is a linkage of $G_{\mathcal{L}}$. Notice that if $\mathcal{L}$ is strongly vital then $\mathcal{L}^*$ is not necessarily strongly vital. However, if $\mathcal{L}$ is weakly vital, then so is $\mathcal{L}^*$ (see Figure 3.3 for an example).

Vertex dissolving. Let G be a graph and $v \in V(G)$ with $N_G(v) = \{u, w\}$. The operation of *dissolving* v in G is the following: Delete v from G and add edge $\{u, w\}$ to $E(G)$, allowing the existence of multiple edges.

3.2 Cyclic linkages in plane graphs

In this section we prove that the existence of a unique cyclic linkage in planar graph G forces the treewidth of G to be small.

The following proposition follows from the combination of Lemma 5, Lemma 6, and Observation 3 of [3].

Proposition 3.2.1. *Let G be a graph embedded on the sphere $\mathbb{S}_0$ and let $\mathcal{Q} = (\mathcal{C}, \mathcal{L})$ be a touch-free CGL-configuration of G, where $\mathcal{C}$ is tight in G and $\mathcal{L}$ is a $\mathcal{C}$-weakly cheap linkage whose penetration in $\mathcal{C}$ is at least r. Then G contains some $\mathcal{L}$-tidy tilted grid in G of capacity at least $r/(4 \cdot |\mathcal{P}(\mathcal{L})|)$.*

The following result is somehow surprising as the strongly vitality of a cyclic linkage $\mathcal{L}$ in a plane graph excludes $\mathcal{L}$-tidy tilted grids of any size greater than 3. The interesting thing is that a similar theorem also holds for (see [3]) for unique linkages but the size of the excluded tidy tilted grids for this case depends on the number of the paths k of the linkage (and more specifically its exponential on k).

Lemma 3.2.1. *Let G be a graph embedded on the sphere $\mathbb{S}_0$. If G contains a strongly vital cyclic linkage $\mathcal{L} = (C, T)$, then G does not contain an $\mathcal{L}$-tidy tilted grid of capacity 4.*

Proof. Assume that $\mathcal{L} = (C, T)$ is a strongly vital cyclic linkage in G and that Γ is an $\mathcal{L}$-tidy tilted grid of capacity 4 in G. Let also Γ_4 be the (4×4)-grid. Observe that Γ_4 is the graph that we get after contracting all edges of Γ with at least one endpoint of degree 2. We contract Γ to Γ_4 in G and let G' be the resulting graph.

Let $V(\Gamma_4) = \{v_{ij} \mid i, j \in \{1, \ldots 4\}\}$ and

$$E(\Gamma_4) = \{\{v_{ij}, v_{i'j'}\} \mid |i - i'| + |j - j'| = 1\}.$$

Observe that Γ_4 is also an $\mathcal{L}$-tidy tilted grid of capacity 4 in G' and that $\mathcal{L}$ is also strongly vital in G' (if not, then it was not strongly vital in G). Let $H = \Gamma \cup C$ and H' be the contraction of H that we get after contracting all edges of H whose ends have both degree 2.

Let also

$$H^* = \Gamma_4 \cup P_1 \cup P_2 \cup P_3 \cup P_4,$$

where for every $i \in \{1, 2, 3, 4\}$, each P_i is a path of length 2 such that P_1 connects v_{11} with v_{12}, P_2 connects v_{13} with v_{14}, P_3 connects v_{41} with v_{44} and P_4 connects v_{42} with v_{43} (i.e. for every cyclic linkage $\mathcal{L} = (C, T)$ if we contract all edges of $H = \Gamma \cup C$ whose ends have degree 2, we get a graph isomorphic to H^* which is a (4×4)-grid in addition to some paths that are subgraphs of C).

It is not hard to confirm that for every possible H, its corresponding contraction, H', is isomorphic to H^*. It remains to show that there exists a cyclic linkage $\mathcal{L}' = (C', T)$ in G', where C' is different from C. As H^* is a unique graph (up to isomorphism), a way of rerouting C (in order to obtain a different cyclic linkage) is given in Figure 3.4. $\square$

Lemma 3.2.2. *Let G be a graph embedded on the sphere $\mathbb{S}_0$ that is the union of $r \geq 2$ concentric cycles $\mathcal{C} = \{C_1, \ldots, C_r\}$ and one more cycle C of G. Assume that $\mathcal{C}$ is tight in G, $T \cap V(\hat{C}_r) = \emptyset$, the cyclic linkage $\mathcal{L} = (C, T)$ is strongly vital in G, and its penetration in $\mathcal{C}$ is r. Then $r \leq 16 \cdot |T| - 1$.*

Proof. Let $\sigma : \mathcal{P}(\mathcal{L}) \to T$ be such that σ is a bijection that maps each path of $\mathcal{P}(\mathcal{L})$ to one of its endpoints. For every $i \in \{1, \ldots, r\}$, we define $\mathcal{Q}^{(i)} = (\mathcal{C}^{(i)}, \mathcal{L}^{(i)})$ where $\mathcal{C}^{(i)} = \{C_1, \ldots, C_i\}$ and $\mathcal{L}^{(i)} = (C, T^{(i)})$ where

$$T^{(i)} = T \setminus \{\sigma(P) \mid P \in P(\mathcal{L}) \text{ and } P \cap \hat{C}_i = \emptyset\}.$$

Notice that if some $\mathcal{Q}^{(i+1)}$ is not touch-free, then $T^{(i)} \leq T^{(i+1)} - 1$ (as, by the definition of touch-free configurations, there exists at least one path P in $\mathcal{P}(\mathcal{L})$ such that $P \cap \hat{C}_{i+1} \neq \emptyset$ but $P \cap \hat{C}_i = \emptyset$). In the trivial case where every $\mathcal{Q}^{(i)}$ is not touch-free we derive easily that $r \leq |T|$ and we are done.

Otherwise, let $\mathcal{Q}' = (\mathcal{C}', \mathcal{L}')$ be the touch-free CGL-configuration in $\{\mathcal{Q}^{(1)}, \ldots, \mathcal{Q}^{(r)}\}$ of the highest index, say i (as we excluded the trivial case we have that $i \geq 1$). Certainly, $\mathcal{C}' = \mathcal{C}^{(i)}$ and $\mathcal{Q}'$ is tight in G. Moreover, $\mathcal{L}'$ is strongly vital in G. From Lemma 3.2.1, G does not contain an $\mathcal{L}'$-tidy tilted grid of capacity 4. Thus, $G_{\mathcal{L}}$ as well does not contain an $\mathcal{L}'^*$-tidy (remember how a linkage $\mathcal{L}^*$ is created from a graph linkage $\mathcal{L}$ after the "duplication" of the terminals of $\mathcal{L}$) tilted grid of capacity 4. Recall now that, as $\mathcal{L}'$ is strongly vital in G it is also weakly vital in G and therefore $\mathcal{L}'^*$ is weakly vital in $G_{\mathcal{L}'}$. Notice also that $\mathcal{Q}'^* = (\mathcal{C}', \mathcal{L}'^*)$ is a CGL-configuration of $G_{\mathcal{L}'}$ where $\mathcal{C}'$ is tight in $G_{\mathcal{L}'}$. As $\mathcal{L}'^*$ is weakly vital in $G_{\mathcal{L}'}$, then, by its uniqueness, $\mathcal{L}'^*$ is $\mathcal{C}'$-weakly cheap. Recall

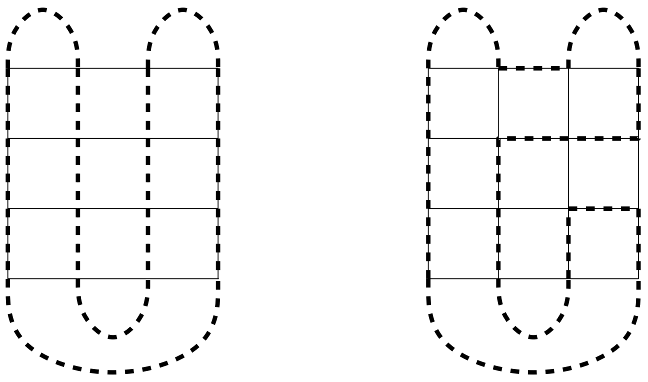


Figure 3.4: On the left, a simplified $\mathcal{L}$-tidy (4×4)-grid (corresponding to graph H^*) and on the right, a rerouting of the cycle of $\mathcal{L}$ in the grid.

that the penetration of $\mathcal{L}'$ in $\mathcal{C}'$ is $r-(i-1)$ and so is the penetration of $\mathcal{L}'^*$ in $\mathcal{C}'$. As $\mathcal{Q}'$, and therefore $\mathcal{Q}'^*$ as well, is touch-free we can apply Proposition 3.2.1 and obtain that $G_{\mathcal{L}'}$ contains some $\mathcal{L}'^*$-tidy tilted grid of capacity at least

$$(r-(i-1))/(4 \cdot |\mathcal{P}(\mathcal{L}'^*)|) \leq (r-(i-1))/(4(|\mathcal{P}(\mathcal{L})|-(i-1)).$$

We derive that

$$(r-(i-1))/(4(|\mathcal{P}(\mathcal{L})|-(i-1)) < 4,$$

therefore $r \leq 16 \cdot |\mathcal{P}(\mathcal{L})| - 15(i-1)$ which implies that $r \leq 16 \cdot |T| - 1$. $\square$

A corollary of Lemma 3.2.2 is the following.

Corollary 3.2.1. *If a plane graph G contains a strongly vital cyclic linkage $\mathcal{L} = (C, T)$, then* $\mathbf{tw}(G) = O(|T|^{3/2})$.

This corollary is a combinatorial result of independent importance and we think that it can be used to study more connectivity related problems. It states that, the uniqueness of a vital cyclic linkage in a plane graph G implies that the treewidth of G is bounded by $c \cdot t^{3/2}$, where c is a constant and t is the number of terminals of the cyclic linkage.

Notice that, according to what is claimed in [3], we cannot restate the above corollary for weakly vital linkages, unless we change the bound to be an exponential one. That way, the fact that treewidth is (unavoidably, due to [3]) exponential to the number of terminals for (weakly) vital linkages is caused by the fact that the ordering of the terminals is predetermined.

Lemma 3.2.3. *Let G be a graph embedded on the sphere $\mathbb{S}_0$ that is the union of r concentric cycles $\mathcal{C} = \{C_1, \ldots, C_r\}$ and a Hamiltonian cycle C of G. Let also $T \cap V(\hat{C}_r) = \emptyset$. If $\mathcal{L} = (C, T)$ is $\mathcal{C}$-strongly cheap then $\mathcal{L}$ is a strongly vital cyclic linkage in G.*

Proof. Assume that $\mathcal{L}$ is not strongly vital in G, i.e., there is a different, isomorphic to $\mathcal{L} = (C, T)$, cyclic linkage $\mathcal{L}' = (C', T)$ in G. As $\mathcal{L} \neq \mathcal{L}'$ we have that $C' \neq C$, therefore there exists an edge $e \in E(C') \setminus E(C)$ (this is because $V(C) = V(C')$ which follows from the strong vitality of $\mathcal{L}$ in G).

But, as

$$E(G) = E(C) \cup \bigcup_{i=1}^{r} E(C_i),$$

we derive that $e \in \bigcup_{i=1}^{r} E(C_i)$ (observe that the only way C' can be different from C is by using extra edges from the cycles of $\mathcal{C}$).

Thus, we get that

$$|E(C') \cap \bigcup_{i=1}^{r} E(C_i)| > |E(C) \cap \bigcup_{i=1}^{r} E(C_i)|$$

and, by the definition of cheap graph linkages, $c(\mathcal{L}) > c(\mathcal{L}')$, which contradicts the assumption that $\mathcal{L}$ is $\mathcal{C}$-strongly cheap. Therefore, $\mathcal{L} = (C, T)$ is a strongly vital cyclic linkage in G, as claimed. □

We are now able to prove the main combinatorial result that is used in our algorithm for solving Cyclability on planar graphs.

Lemma 3.2.4. *Let G be a plane graph containing some sequence of concentric cycles $\mathcal{C} = \{C_1, \ldots, C_r\}$. Let also $\mathcal{L} = (C, T)$ be a cyclic linkage of G where $T \cap V(\hat{C}_r) = \emptyset$. If $\mathcal{L}$ is $\mathcal{C}$-strongly cheap, then the penetration of $\mathcal{L}$ in $\mathcal{C}$ is at most $r \leq 16 \cdot |T| - 1$.*

Proof. Suppose that some path $P \in \mathcal{P}(\mathcal{L})$ intersects at least $16 \cdot |T|$ cycles of $\mathcal{C}$. Then, P intersects all cycles in $\mathcal{C}^* = \{C_{r-16\cdot|T|+1}, \ldots, C_r\}$.

Let G' be the graph obtained by $C \cup \pmb{\cup}\mathcal{C}^*$ after dissolving all the vertices of degree 2 that do not belong to T and let $\mathcal{L}' = (C', T)$ be the linkage of G' obtained from $\mathcal{L}$ if we dissolve the same vertices in the paths of $\mathcal{L}$. Similarly, by dissolving vertices of degree 2 in the cycles of $\mathcal{C}^*$ we obtain a new sequence of concentric cycles which, for notational convenience, we denote by $\mathcal{C}' = \{C_1, \ldots, C_{r'}\}$, where $r' = 16 \cdot |T|$.

The cyclic linkage $\mathcal{L}'$ is $\mathcal{C}'$-strongly cheap because $\mathcal{L}$ is $\mathcal{C}$-strongy cheap (it is easy to observe that no edge of $\bigcup_{i=1}^{r} E(C_r) \setminus E(C)$ belongs to $E(C')$). Notice that C' is a Hamiltonian cycle of G' and, from Lemma 3.2.3, $\mathcal{L}'$ is a strongly vital cyclic linkage of G'. We also assume that $\mathcal{C}'$ is tight (otherwise we can replace it by a tight one and observe that, by its uniqueness, $\mathcal{L}'$ will be cheap to this new one as well). As $\mathcal{L}'$ is $\mathcal{C}'$-strongly cheap and $\mathcal{C}'$ is tight, from Lemma 3.2.2, $r' \leq 16 \cdot |T| - 1$; a contradiction. □

This result can be intuitively described as follows: Any cyclic linkage $\mathcal{L} = (C, T)$ that penetrates a sequence of concentric cycles $\mathcal{C}$, that does not contain any of the terminals of T, deep enough (deeper than $16 \cdot |T| - 1$), can be replaced by another linkage $\mathcal{L}' = (C', T)$ (specifically a $\mathcal{C}$-strongly cheap one) whose penetration in $\mathcal{C}$ is at most $16 \cdot |T| - 1$.

Or to restate this, when trying to find a cyclic linkage that meets the terminals in T we do not have to go deep into a sequence of concentric cycles $\mathcal{C}$ that does not contain any element of T. This, of course, means that the vertices "in the central" part of $\mathcal{C}$ are *irrelevant* for our problem.

This is the basis of our application of the irrelevant vertex technique and will be used for the analysis of our algorithm in Chapter 5. In the next chapter, we talk about dynamic programming algorithms for problems on graphs of bounded treewidth and we design such an algorithm for the Cyclability problem.

CHAPTER 4 ALGORITHMS FOR GRAPHS OF BOUNDED TREEWIDTH

4.1 Using Courcelle's theorem for Cyclability

We are now ready to show how we can "efficiently" (at the end of this section, the use of brackets will become clear) solve instances of Cyclability, when the input graph has bounded treewidth. Remember that this is a crucial step of our algorithm: When the instance, after maybe some deletions of irrelevant vertices, has sufficiently small treewidth, we solve it using dynamic programming. More specifically:

We can prove that the Cyclability problem can be solved in FPT-time when we restrict the treewidth of the input graphs to bounded by some function of k, by expressing the property of a graph being k-cyclable in MSO_2 and directly employing Courcelle's theorem (as described in Proposition 2.3.1 of subsection 2.3). It suffices to prove the following lemma:

Lemma 4.1.1. *The property of a graph G being k-cyclable can be expressed in MSO_2 for every integer $k \geq 0$.*

Proof. Let k be a positive integer. We define

$$\textbf{cyclability}_\mathbf{k}(G) = \forall_{S \subseteq V} \big[\textbf{card}_\mathbf{k}(S) \to \big(\exists_{C \subseteq E}\, \textbf{connE}(C) \wedge \\ \forall_{v \in V}\, \textbf{deg0or2}(v, C) \ \wedge\ \forall_{v \in S}\, \textbf{deg2}(v, C)\big)\big],$$

where $\textbf{card}_\mathbf{k}(S)$ is an auxiliary formula that checks whether the cardinality of the set $S \subseteq V$ is k. Formula $\textbf{connE}(C)$ checks whether the graph $(V(C), C)$ (where $V(C)$ is the set of all vertices that are endpoints of some edge in C) is connected and formulas $\textbf{deg0or2}(v, C)$ (resp. $\textbf{deg2}(v, C)$) verifies that a vertex v has zero or two adjacent (resp. exactly two) edges belonging to C. It is now clear that, the formula $\textbf{cyclability}_\mathbf{k}$ is satisfied by a graph $G = (V, E)$ if and only if the following property holds:

For every subset S of V with cardinality k, there exists a cycle C in G that contains all the vertices in S.

Of course this is equivalent to G being k-cyclable. It remains to define the four auxiliary formulas.

$$\textbf{card}_\mathbf{k}(S) = \exists_{s_1 \in S}\ \exists_{s_2 \in S} \ldots \exists_{s_k \in S}\big(v \neq s_1 \wedge v \neq s_2 \wedge \ldots \\ \wedge\, v \neq s_k \to v \notin S\big)$$

$$\textbf{connE}(C) = \forall_{Y \subseteq V(C)} \big[\big(\exists_{u \in V(C)}\, u \in Y \ \wedge\ \exists_{v \in V(C)}\, v \notin V(C) \setminus Y\big) \to \\ \big(\exists_{e \in C}\, \exists_{u \in Y}\, \exists_{v \in V(C) \setminus Y}\, \mathsf{inc}(u, e) \wedge \mathsf{inc}(v, e)\big)\big]$$

$$\textbf{deg2}(v, C) = \exists_{e_1, e_2 \in C}\ \big[(e_1 \neq e_2) \ \wedge\ \mathsf{inc}(v, e_1) \ \wedge\ \mathsf{inc}(v, e_2) \wedge \\ \big(\forall_{e_3 \in C}\ \mathsf{inc}(v, e_3) \to \big(e_1 = e_3 \vee e_2 = e_3\big)\big)\big]$$

$$\textbf{deg0or2}(v, C) = \textbf{deg2}(v, C) \ \vee\ \forall_{e \in C}\, \neg\mathsf{inc}(v, e)$$

□

The following theorem is a direct consequence of Proposition 2.3.1 and Lemma 4.1.1 that we just proved

Theorem 4.1.1. Cyclability *problem, when restricted to the class of graphs of bounded treewidth and parameterized by k, belongs to the class* FPT*.*

However, as we have already discussed in subsection 2.3, the celebrated metatheorem of Courcelle, despite its theoretical power, is used almost exclusively as a classification tool due to the (unavoidably, [44]) immense running times that result from the construction of the automaton described in the proof. One way to unlock the door to efficient algorithms for a problem restricted to graphs of bounded treewidth is to exploit the specific structure of the problem and use dynamic programming on a "good" tree decomposition of the input graph. Next, we provide some intuition on this technique and demonstrate how it can be applied on the Cyclability problem.

4.2 Treewidth and dynamic programming

A very useful property of a tree decomposition, say $\mathcal{D} = (T, \mathcal{X})$, of a graph G is that the intersection of any two bags that are endpoints of an edge in T is a separator of G. When trying to solve a problem in graphs of bounded treewidth, we can sometimes exploit (for example by deploying dynamic programming) the fact that our graph can be decomposed into pieces which are connected through sets of "small" size to obtain an efficient solution. We formally state and prove the property we discussed

Lemma 4.2.1. *Let $\mathcal{D} = \left(T, \{X_t\}_{t \in V(T)}\right)$ be a tree decomposition of a graph G and let $\{a, b\}$ be an edge of T. The forest $T \setminus \{a, b\}$ obtained by deleting the edge $\{a, b\}$ from T, consists of two connected components T_a (that contains a) and T_b (that contains b). If*

$$A = \bigcup_{t \in V(T_a)} X_t \quad \textit{and} \quad B = \bigcup_{t \in V(T_b)} X_t,$$

then the set $X_a \cap X_b \subseteq V(G)$ separates A from B in G.

Proof. Both a and b belong to every path with endpoints $t_1 \in T_1$ and $t_2 \in T_2$. Therefore, $A \cap B \subseteq X_a \cap X_b$ and it now suffices to prove that G has no edge $\{u, v\}$ with $u \in A \setminus B$ and $b \in B \setminus A$. If such an edge $\{u, v\}$ exists in G then there is a $t \in T$ such that $u, v \in X_t$. By the choice of u and v we have that $t \notin T_a$ and $t \notin T_b$, a contradiction. □

In order to handle a tree decomposition more "smoothly" we would like it to have some properties, which are presented in the next definition.

Definition 4.2.1. *For a tree decomposition $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$, we distinguish a vertex $r \in V(T)$, which is called the* root *of T, and this way introducing parent-child and ancestor-descendant relations in T. We say that the rooted (on r) tree decomposition $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$ is* nice *if the following conditions hold:*

- *The root r and any non-leaf vertex have exactly two children.*
- *The bags that correspond to the root and the leaves of T are all empty, i.e. $X_r = X_l = \emptyset$ for every leaf l of T.*
- *Every non-leaf node of T has one of the following types:*
  - **Introduce node:** *A node t with exactly one child t' such that $X_t = X_{t'} \cup \{v\}$ for some $v \notin X_{t'}$. For such a node we say that v is* introduced *at t.*
  - **Forget node:** *A node t with exactly one child t' such that $X_t = X_{t'} \setminus \{v\}$ for some $v \in X_{t'}$. For such a node we say that v is* forgotten *at t.*
  - **Join node:** *A node t with two children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$.*

It is not hard to prove the following Lemma (see for example [36] and [22]):

Lemma 4.2.2. *If a graph G admits a tree decomposition of width at most k, then it also admits a nice tree decomposition of width at most k. Moreover, given a tree decomposition $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$ of G of width at most k, one can in time $O\big(k^2 \cdot \max(|V(T)|, |V(G)|)\big)$ compute a nice tree decomposition of G of width at most k that has at most $O\big(k|V(G)|\big)$ nodes.*

Of course, in order to algorithmically exploit the nice structure of tree decompositions we first need to, somehow, obtain a tree decomposition of optimum, or approximately optimum, width for the given graph.

It turns out that computing the treewidth of a given graph is an NP-hard problem ([6]) but it is FPT to check whether its treewidth is at most w, where w is considered to be the parameter. The later follows from the Robertson-Seymour theory, as treewidth is a minor-closed parameter: If H is a minor of a graph G then $\mathbf{tw}(H) \leq \mathbf{tw}(G)$.

As we have already mentioned, this technique is not constructive. Fortunately, here is an algorithm of Bodlaender ([8]), that given a n-vertex graph G and an integer k, runs in time $k^{O(k^3)} \cdot n$ and either constructs a tree decomposition of G of width at most k or concludes that $\textbf{tw}(G) \geq k$.

In this thesis we will not focus on the computation of treewidth and we will usually assume that an optimal, or nearly optimal, tree decomposition is given.

We proceed by giving some concrete examples of applying dynamic programming for problems on graphs of bounded treewidth.

Dynamic programming. One of the most well known techniques in the algorithms design theory is *dynamic programming*. The general idea behind this technique is that we can solve a complex problem by dividing it into simpler subproblems, solving each one of them and combining the solutions (which are stored and used whenever required). More details of these ideas can be found in any introductory algorithms book such as [24] and [19].

A typical application of dynamic programming is solving problems on trees. Consider for example the following problem

Weighted Independent Set on Trees
Input: A tree $T = (V, E)$ rooted at a node $r \in V$ and a function $w : V \to \mathbb{R}_+$.
Goal: Find a set $I \subseteq V$ such that $G[I]$ is edgeless and $\sum_{i \in I} w_i$ is maximum.

Let $T = (V, E)$ be tree, $r \in V$ be the root of tree T and $w : V \to \mathbb{R}_+$ be a function that assigns a weight on every node of the tree. We say that a node in $v \in V$ is a *leaf* of T if $\mathsf{deg}_T(v) = 1$. We denote by $\mathsf{dep}_T(v)$ the depth of node v in tree T and define the depth of the root r to be equal to zero and the depth of a node v to be its distance from the root r. For any $v \in V$, we define

$$N_T^1(v) = \{u \in V \mid \mathsf{dep}_T(u) = \mathsf{dep}_T(v) + 1\}$$

$$N_T^2(v) = \{u \in V \mid \mathsf{dep}_T(u) = \mathsf{dep}_T(v) + 2\}$$

We construct a dynamic programming routine for solving Weighted Independent Set on Trees. We define

$$c[v] = \begin{cases} w(v) & \text{if } v \text{ is a leaf of } T \\ \max\left\{ \sum_{u \in N_T^1} w[u], \sum_{u \in N_T^2} w[u] + w[v] \right\} & \text{otherwise} \end{cases}$$

Clearly, the value $c[r]$ computed in a leaf-to-root manner, is the solution we are looking for, i.e. $c[r]$ is the value of a maximum independent set in T. The formula for $c[v]$ is valid, as node v will either be in the solution (excluding its children in N_T^1) or it will not and thus all nodes in N_T^1 can be in the solution.

This simple example demonstrates the power of dynamic programming: The computation that we do on every step depends only on values of some (usually one or two) of the prior steps. Of course, for this strategy to work there has to be some special structure in our problem. The crucial property in our example is that every inner node (different from the root and the leaves) of a tree is a separator, i.e., the deletion of any inner node makes the graph disconnected.

It turns out that this idea can be extended to problems on graphs of bounded treewidth. If $G = (V, E)$ is a graph and $\mathcal{D} = \big(T, \{X_t\}_{t \in V}\big)$ is a tree decomposition of G of width w then, by Lemma 4.2.1, the vertices in X_t form a separator, of size at most w, in G. This property of tree decompositions is the basis for designing dynamic programming routines for problems on graphs of bounded treewidth, as we will become clear in the next two subsections.

4.2.1 Dynamic programming for Weighted Independent Set

Before presenting a (somewhat involved) dynamic programming algorithm for solving Cyclability on graphs of bounded treewidth, we start with some simpler applications. We first demonstrate how the technique can be applied on the Weighted Independent Set problem.

Weighted Independent Set
Input: A graph $G = (V, E)$ and a function $w : V \to \mathbb{R}_+$.
Goal: Find a set $I \subseteq V$ such that $G[I]$ is edgeless and $\sum_{i \in I} w_i$ is maximum.

Let $G = (V, E)$ be a graph with $|V| = n$, and let $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$ be a tree decomposition of G that has width at most k. By Lemma 4.2.2, we can assume that $\mathcal{D}$ is a nice tree decomposition of G, rooted at some node $r \in V(T)$. We define V_t to be the union of all bags that belong to the subtree of T that is rooted at t, including the bag, X_t, that corresponds to t. By applying Lemma 4.2.1 to any edge between some $t \neq r$ and its parent, we get that $\partial V_t \subseteq X_t$ (the same holds, trivially, for the root r as $V_r = V$ and $\partial V_r = \emptyset$). This fact is crucial because it demonstrates that the subgraph induced by V_t can communicate with the rest of the graph through X_t which is "small" (as it is a bag of the decomposition). This means that we can perform time consuming computations on X_t without getting inefficient and then producing a solution by, either combining the solutions of the subproblems or by extending our partial solution at every step. This, somewhat intuitive, strategy becomes clear when applying it to the Weighted Independent Set problem.

Let $G = (V, E)$ and $w : V(G) \to \mathbb{R}_+$ be an input for the problem and let $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$ be a nice tree decomposition of G with width k. An important observation is that for two given independent sets, I_1 and I_2, in G, if $I_1 \cap X_t = I_2 \cap X_t$ and $w(I_1 \cap V_t) > w(I_2 \cap V_t)$, then I_2 is definitely not the optimal solution (as we can replace $I_2 \cap V_t$ with $I_1 \cap V_t$ an independent set of bigger weight). This holds because X_t separates $V_t \setminus X_t$ from the rest of the graph and for this specific problem we can naturally extend a partial solution for V_t to the whole graph G, by making some "local" computation for each bag of the decomposition. More specifically, for every $t \in V(T)$ and every $S \subseteq X_t$ we define

$$c[t, S] = \max\big\{w(\tilde{S}) \mid S \subseteq \tilde{S} \subseteq V_t,\ \tilde{S} \cap X_t = S \text{ and } \tilde{S} \text{ is independent}\big\}$$

If no such $\tilde{S}$ exists (meaning that S is not an independent set) we set $c[t, S] = -\infty$. Observe that $c[t, S]$ can be computed efficiently (with respect to the treewidth of G) for every t, as there are at most $2^{|X_t|}$ choices for the intersection of $\tilde{S}$ with X_t and we can efficiently check if a vertex set $\tilde{S}$ is independent. Moreover, it is clear that $c[r, \emptyset]$ is the maximum weight of an independent set, as $V_r = V$ and $X_r = \emptyset$.

We now show how we can compute the values $c[t, S]$. As we will shortly see, the notion of a nice tree decomposition will prove to be very beneficial because it forces the relation between a bag and the bag of its children to be "simple". We give recursive formulas for $c[t, S]$, where leaf nodes correspond to the base case of the recurrence:

- **Leaf node:** If t is a leaf of T, then we have $c[t, \emptyset] = 0$
- **Introduce node:** If t is an introduce node with child t', then we have that $X_t = X_{t'} \cup \{v\}$, for some $v \notin X_{t'}$. Let $S \subseteq X_t$ and assume that S is independent. Then we set

  $$c[t, S] = \begin{cases} c[t', S] & \text{if } v \notin S \\ c[t', S \setminus \{v\}] + w(v) & \text{otherwise} \end{cases}$$

  If S is not independent, then we set $c[t, S] = -\infty$. It is easy to check that our recurrence formula is true, as any partial solution of $V_{t'}$ when extended to V_t will either contain the introduced node v or it will not contain it.

- **Forget node:** If t is a forget node with child t', then we have that $X_t = X_{t'} \setminus \{v\}$, for some $v \in X_{t'}$. Let $S \subseteq X_t$ and assume that S is independent. Then we set

  $$c[t, S] = \max\big\{c[t', S], c[t', S \cup \{v\}]\big\}$$

  Again, if S is not independent then we set $c[t, S] = -\infty$. If $v \in \tilde{S}$, then $c[t', S \cup \{v\}] \geq w(\tilde{S}) = c[t, S]$, and if $v \notin \tilde{S}$ then $c[t', S] \geq w(\tilde{S}) = c[t, S]$. As exactly one of this happens we get that

  $$c[t, S] \leq \max\big\{c[t', S], c[t', S \cup \{v\}]\big\}.$$

  On the other hand, each set considered in the maximization for $c[t', S]$ is also considered for the maximization of $c[t, S]$, and the same holds for $c[t', S \cup \{v\}]$. So we get that

  $$c[t, S] \geq \max\big\{c[t', S], c[t', S \cup \{v\}]\big\}.$$

  By combining the previous two inequalities we confirm that our recurrence relation is true.

- **Join node:** Suppose now that t is a join node with children t_1, t_2 such that $X_t = X_{t_1} = X_{t_2}$. Let $S \subseteq X_t$ and assume that S is independent (if not we set $c[t, S] = -\infty$ as before). We claim that

  $$c[t, S] = c[t_1, S] + c[t_2, S] - w(S).$$

  This follows from the observation that there is no edge between $V_{t_1} \setminus X_t$ and $V_{t_2} \setminus X_t$ (by Lemma 4.2.1). This means that we can "join" partial solutions for the subproblems on V_{t_1} and V_{t_2} and take care only of the common vertices in S.

It now remains to estimate the time needed in order to compute $c[r, \emptyset]$, which the solution we are looking for: the maximum weight of an independent set in G. As we are working on a tree decomposition of width k, we have, for every $t \in V(T)$, that $|X_t| \leq k + 1$ and therefore at every node t we compute $2^{|X_t|} \leq 2^{k+1}$ values of $c[t, S]$ in time $2^k \cdot k^{O(1)}$ (using a data structure that allows to perform adjacency queries in $O(k)$ time). Finally, as the number of nodes of the tree decomposition is $O(kn)$, we obtain the following

Theorem 4.2.1. *Let $G = (V, E)$ with $|V| = n$ and $w : V(G) \to \mathbb{R}_+$. Let also $\mathcal{D} = \big(T, \{X_t\}_{t \in V(T)}\big)$ be a nice tree decomposition of G of width at most k. Then, the* Weighted Independent Set *problem in G can be solved in time $2^k \cdot k^{O(1)} \cdot n$.*

4.2.2 Dynamic programming for Hamiltonian Cycle

The previous example is a typical application of dynamic programming on a tree decomposition. However, the Weighted Independent Set problem has nothing to do with our problem, Cyclability.

Before proceeding to the presentation of the dynamic programming algorithm for solving Cyclability we apply the technique on a well known problem, which is clearly strongly related to Cyclability, namely the Hamiltonian Cycle problem:

Hamiltonian Cycle
Input: A graph $G = (V, E)$.
Question: Is there a cycle in C in G such that $V(C) = V$?

Let $G = (V, E)$ be a graph of treewidth at most w and let $\mathcal{D} = \big(T, \{X_t\}_{t \in V_t}\big)$ be a nice tree decomposition of width w of G.
Let X be a node (or bag) of $\mathcal{D}$, i.e., $X = X_t$ for some $t \in V(t)$. We define

$$B_X = \{v \in V \mid v \in X\}$$

$$V_X = \{v \in V \mid v \text{ appears in the subtree of } T \text{ rooted at } X\}$$

Suppose that H is a Hamiltonian cycle in G. Then, as the vertices in V_X communicate with the rest vertices of V only through the vertices of X (follows from Lemma 4.2.1), we have that the subgraph $H[V_X]$ of H, is a set of paths with endpoints in B_X. The vertices of B_X are partitioned into the three following sets according to their degree in $H[V_X]$:

$$B_X^0 = \{v \in B_X \mid \text{deg}_{H[V_X]} = 0\}$$

$$B_X^1 = \{v \in B_X \mid \text{deg}_{H[V_X]} = 1\}$$

$$B_X^2 = \{v \in B_X \mid \text{deg}_{H[V_X]} = 2\}$$

A subproblem for node X is a quadruple (B_X^0, B_X^1, B_X^2, M), where M is a matching of the vertices in B_X^1 (two matched vertices are the endpoints of some path). Clearly, the number of different subproblems on a node X is at most $3^w \cdot w^w$, as every one of the at most w nodes belongs to exactly one part of the partition and the different matchings of B_X^1 (whose size is at most w) are at most w^w.

We call each subproblem (B_X^0, B_X^1, B_X^2, M) a *pattern* and what we want to compute on every node X of the tree decomposition, is if there is a set of paths with this pattern in $G[V_X]$. If there is such a set of paths, we say that this particular subproblem is *valid* on node X. We assume, for technical reasons, that the root and the leaves contain exactly one node (they are not empty as defined in Definition 4.2.2). We argue about the subproblem (B_X^0, B_X^1, B_X^2, M) on node X according to its type:

- **Node X is a leaf node:** Only the trivial subproblem $(\{v\}, \emptyset, \emptyset, \emptyset)$ is valid on X, where $B_X = \{v\}$.
- **Node X is a forget node:** Suppose that the child of X is node Y and $B_X = B_Y \setminus \{v\}$. As each node is introduced only once in a nice

tree decomposition, for a solution H of (B_X^0, B_X^1, B_X^2, M), vertex v has degree 2 and therefore (B_X^0, B_X^1, B_X^2, M) is valid on X if and only if $(B_X^0, B_X^1, B_X^2 \cup \{v\}, M)$ is valid on Y.

- **Node X is an introduce node:** Suppose that the child of X is node Y and $B_X = B_Y \cup \{v\}$.
  - *Case 1*: $v \in B_X^0$. It is easy to confirm that (B_X^0, B_X^1, B_X^2, M) is valid on X if and only if $(B_X^0 \setminus \{v\}, B_X^1, B_X^2, M)$ is valid on Y.
  - *Case 2*: $v \in B_X^1$. Then, every neighbour of v in V_X has to be in $B_Y \subseteq B_X$. Suppose that v is adjacent with just one vertex of B_Y. Subproblem (B_X^0, B_X^1, B_X^2, M) is valid on X if there is a subproblem $(B_Y^0, B_Y^1, B_Y^2, M')$ of node Y such that making a vertex of B_Y adjacent to v produces a solution to (B_X^0, B_X^1, B_X^2, M) on node X. This can be checked in at most w steps.
  - *Case 2*: $v \in B_X^1$ and v is adjacent with two vertices of B_Y. The analysis is similar to Case 2 and the validity of (B_X^0, B_X^1, B_X^2, M) on X can be checked in w^2 steps (we check for every pair of vertices in B_Y if we can produce a solution by connecting them both with v).
- **Node X is a join node:** Node X has two children, Y and Z, and $B_X = B_Y = B_Z$. A solution H on X is the union of a subgraph $H_1 \subseteq G[V_Y]$ and a subgraph $H_2 \subseteq G[V_Z]$. For every solution H_1 for $(B_Y^0, B_Y^1, B_Y^2, M_1)$ of node Y and every solution H_2 for $(B_Z^0, B_Z^1, B_Z^2, M_2)$ of node Z, we check whether their union $H_1 \cup H_2$ is a solution for (B_X^0, B_X^1, B_X^2, M) of node X.

Theorem 4.2.2. *Given a graph G and a tree decomposition with width w of G, the* Hamiltonian Cycle *problem on G can be solved in $w^{O(w)} \cdot n$ steps.*

The dynamic programming approach for solving Hamiltonian Cycle (and many other important connectivity problems) that we presented, can be improved with the use of the so called *Cut & Count* technique (see [23] and [22]). The improved running time, which is $2^{O(w)} \cdot n$, is single exponential on the treewidth of the input graph and linear on the size of the whole input.

After demonstrating the use of dynamic programming for solving problems on graphs of bounded treewidth, we proceed by applying this technique on the Cyclability problem, where things get more complicated

4.3 Dynamic programming for Cyclability

The construction of the following algorithm is the goal of this section.

Algorithm DP$(G, R, k, q, \mathcal{D})$
Input: A graph G, a vertex set $R \subseteq V(G)$, two non-negative integers k and q, where $k \leq q$, and a tree decomposition $\mathcal{D}$ of G of width q.
Output: An answer whether (G, R, k) is a yes-instance of Planar Annotated Cyclability problem, or not.
Running time: $2^{2^{O(q \cdot \log q)}} \cdot n$.

We observe that the question of Planar Annotated Cyclability can be expressed in monadic second-order logic (MSO$_2$). It is sufficient to notice that an instance (G, R, k) is a yes-instance of Planar Annotated Cyclability if and only if for any (not necessarily distinct) $v_1, \ldots, v_k \in R$, there are sets $X \subseteq V(G)$ and $S \subseteq E(G)$ such that $v_1, \ldots, v_k \in X$ and $C = (X, S)$ is a cycle. The property of $C = (X, S)$ being a cycle is equivalent to asking whether

i) for any $x \in X$, there are two distinct $e_1, e_2 \in S$ such that x is incident to e_1 and e_2,

ii) for any $x \in X$ and any three pairwise distinct $e_1, e_2, e_3 \in S$, e_1 is not incident to x or e_2 is not incident to x or e_3 is not incident to x, and

iii) for any $Z_1, Z_2 \subseteq X$ such that $Z_1 \cap Z_2 = \emptyset$, $Z_1 \neq \emptyset$, $Z_2 \neq \emptyset$ and $Z_1 \cup Z_2 = X$, there is $\{x, y\} \in S$ such that $x \in Z_1$ and $y \in Z_2$.

We have already seen (Lemma 4.1.1) that we can express Cyclability in MSO$_2$ and from Courcelle's theorem we infer that Planar Annotated Cyclability can be solved in $f(q, k) \cdot n$ steps if the treewidth of an input graph is at most q, for some computable function f.

As the general estimation of f provided by Courcelle's theorem is immense, we proceed by giving a dynamic programming algorithm in order to achieve a more reasonable running time.

First, we introduce some notation. For every two integers a and b, with $a < b$, we denote by $[\![a, b]\!]$ the set of integers $\{a, a+1, \dots, b\}$. Let S be a set and $i \in \mathbb{N}$. We define $S^{[i]} = \{A \subseteq S \mid |A| = i\}$.

Sub-cyclic pairs. Let G be a graph, C a cycle in G, and $\{A, X, B\}$ a partition of $V(G)$ such that no edge of G has one endpoint in A and the other in B. The restriction of C in $G[A \cup X]$ is called a *sub-cyclic* pair of G (with respect to A, X and C). We denote such a sub-cyclic pair by $(\mathcal{Q}, Z)$, where $\mathcal{Q}$ contains the connected components of the restriction of C in $G[A \cup X]$ (observe that $\mathcal{Q}$ can contain isolated vertices, a unique cycle, and disjoint paths) and $Z = V(C) \cap X$.

Pairings. Let W be a set. A *pairing* of W is an undirected graph H with vertex set $V(H) \subseteq W$ and where each vertex has degree at most 2 (a loop contributes 2 to the degree of its vertex) and if H contains a cycle then it is unique and all the vertices not in this cycle are of degree 0. Moreover, H may also contain the vertex-less loop. We denote by $\mathcal{P}(W)$ the set of all pairings of W. It is known that if $|W| = w$ then $|\mathcal{P}(W)| = 2^{O(w \cdot \log w)}$.

Edge lifts. Let G be a graph and $v \in V(G)$ such that $\deg_G(v) = 2$. Let also $N_G(v) = \{u, w\}$. We say that the operation of deleting edges $\{v, u\}$ and $\{v, w\}$ and adding edge $\{u, w\}$ (if it does not exist, i.e. we do not allow double edges) is the *edge lift* from vertex v. We denote by $\mathsf{lift}(G, v)$ the graph resulting from G after the edge lift from v (for an example see Figure 4.1).

For a vertex set $L \subseteq V(G)$ and a vertex $v \in V(G)$ we say that graph H=$\mathsf{lift}(G, v)$ is the result of an *L-edge-lift* if $v \in L$.

The dynamic programming algorithm that we present in this section works for the class of all graphs (not just the planar) so we consider the extension of PAC to the class of all graphs

Annotated Cyclability
Input: A graph G, a set $R \subseteq V(G)$, and a non-negative integer k.
Question: Does there exist, for every set S of k vertices in R, a cycle C of G such that $S \subseteq V(C)$?

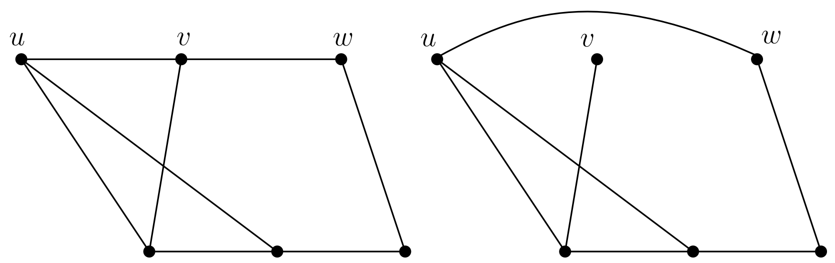


Figure 4.1: A graph G is depicted at the left and at the right there is the graph $\mathsf{lift}(G, v)$ that results from G after the edge lift from vertex v.

Let (G, R, k) be an instance of Annotated Cyclability. Let also $\mathcal{D} = (T, \mathcal{X}, r)$ be a nice tree decomposition of G of width w, where r is the root of T. For every $x \in V(T)$ let T_t be the subtree of T rooted at t (the vertices of T_t are t and its descendants in T). Then for every $t \in V(T)$, we define

$$G_t = G\Big[\bigcup_{t' \in V(T_t)} X_{t'}\Big] \text{ and } V_t = V(G_t).$$

For every $i \in \mathbb{Z}_{>0}$, we set $\mathcal{R}_t^i = (V(G_t) \cap R)^{[i]}$. We also denote $\mathcal{R}_t = \bigcup_{i=1}^{k} \mathcal{R}_t^i$.

If $(\mathcal{Q}, Z)$ is a sub-cyclic pair of G_t where X_t is thought of as the separator and $Z \subseteq X_t$, we simply say that $(\mathcal{Q}, Z)$ is a *sub-cyclic pair on* t. Notice that each sub-cyclic pair $(\mathcal{Q}, Z)$ on t corresponds to a pairing in $\mathcal{P}(X_t)$, which we denote by $P_{\mathcal{Q},Z}$ (just dissolve all vertices of $\mathcal{Q}$ that do not belong in X_t).

Let P be a pairing of X_t and S be a subset of $V(G_t)$. We say that vertex set S *realizes* P in G_t if there exists a sub-cyclic pair $(\mathcal{Q}, Z)$ on t such that $P_{\mathcal{Q},Z} = P$ and $S \subseteq V(\bigcup \mathcal{Q})$.

We also define the *signature* of S in G_t to be the set of all pairings of X_t that S realizes and we denote it by $\mathbf{sig}_t(S)$. Notice that $\mathbf{sig}_t(S) \subseteq \mathcal{P}(X_t)$, therefore $|\mathbf{sig}_t(S)| = 2^{O(w \log w)}$.

Tables. We describe the tables of the dynamic programming algorithm. For each $t \in V(T)$, we define $\mathcal{C}_t = [\![0,k]\!] \times X_t^{[i]} \times \mathcal{P}(X_t)$ and

$$\mathcal{F}_t \quad = \quad \{(i, K, \mathcal{P}) \in \mathcal{C}_t \mid \exists S \in \mathcal{R}_t^i \text{ such that } K = X_t \cap S \text{ and } \mathbf{sig}_t(S) = \mathcal{P}\}$$

We call $\mathcal{F}_t$ the *table* at node $t \in V(T)$. As $|\mathcal{P}(X_t)| = 2^{O(w \cdot \log w)}$, it follows that $|\mathcal{F}(t)| = 2^{2^{O(w \cdot \log w)}}$.

Observe that (G, R, k) is a yes-instance of Annotated Cyclability if and only if $\mathcal{F}_r = \{(0, \emptyset, P_r), (1, \emptyset, P_r), \ldots, (k, \emptyset, P_r)\}$, where P_r is the unique pairing of $\mathcal{P}(X_r)$, i.e., the pairing that is the vertex-less loop (i.e., contains no vertices and a single edge with no endpoints).

New pairings from old. Before we describe the dynamic programming algorithm we need some more definitions. Suppose that t is an insert node of $\mathcal{D}$ and $X_t = X_s \cup \{v\}$, where s is the only child of t in T and $v \in V(G)$. Let $E_v^t = \{\{v, u\} \in E \mid u \in X_t\}$. We denote by $\mathcal{P}_v^{\text{aux}}$ the set of all graphs (V, E) where $V \subseteq N_{G_t}(v) \cup \{v\}$ and $E \subseteq E_v^t$. For any $P \in \mathcal{P}(X_s)$, $\widetilde{P} \in \mathcal{P}_v^{\text{aux}}$, and $L \subseteq X_t$, we define

$$P \oplus_L \widetilde{P} = \{P' \in \mathcal{P}(X_t) \mid P \text{ results from } P \cup \widetilde{P} \text{ after a sequence of } L\text{-edge-lifts}\}.$$

See Figure 4.2 for a visualization of the these definitions. For every $P' \in \mathcal{P}(X_t)$ and $L \subseteq X_t$, we define

$$\zeta_L(P') = \{P \in \mathcal{P}(X_s) \mid \exists \widetilde{P} \in \mathcal{P}_v^{\text{aux}} \text{ such that } P' \in P \oplus_L \widetilde{P}\}.$$

We are now ready to describe the dynamic programming algorithm. We distinguish the following cases for the computation of **table**(t), $t \in V(G)$:

- Node t is a **leaf node**: As $X_t = \emptyset$, we have that $\mathcal{F}(t) = \{(0, \emptyset, G_\emptyset)\}$ where $G_\emptyset$ is the void graph.
- Node t is an **insert node**: Let $X_t = X_s \cup \{v\}$, where s is the unique child of t in T.

We construct **table**(t) by using the following procedure:

Procedure **make_join**

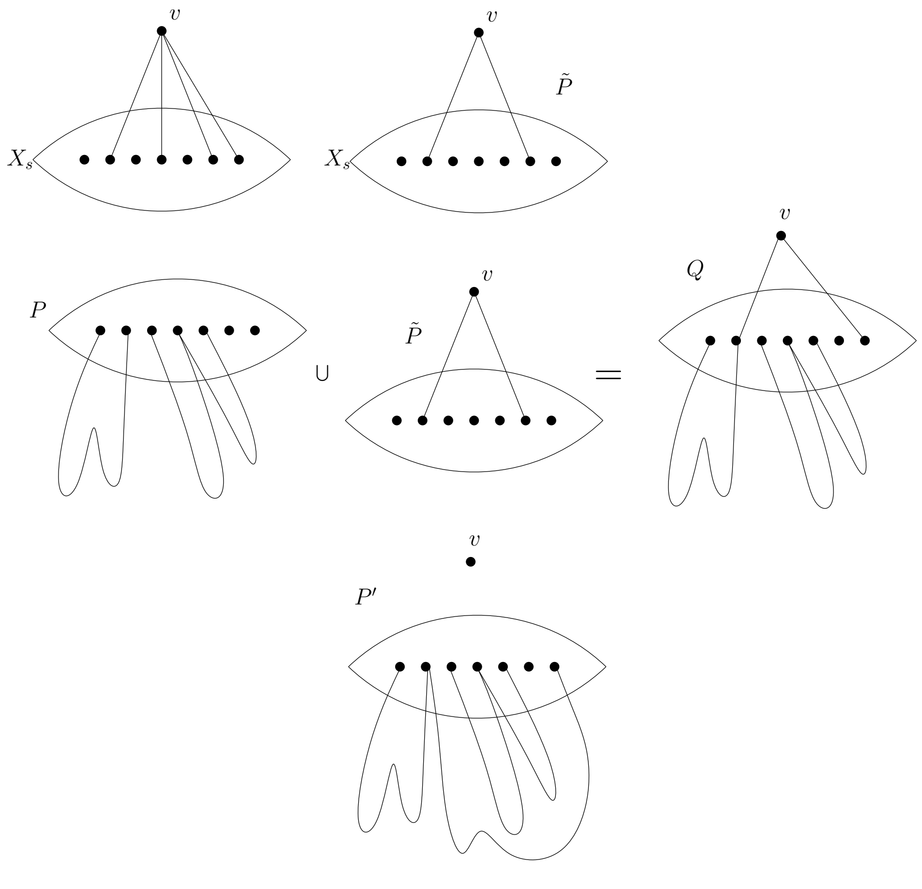


Figure 4.2: At the top we depict the neighborhood of node v in X_s (at the left) and an element, $\tilde{P}$ of P_v^{aux} at the right. In the middle we depict the result, Q, of the union $P \cup \tilde{P}$, where $P \in \mathcal{P}(X_s)$. At the bottom we have the result, $P' \in \mathcal{P}(X_s \cup \{v\}) = \mathcal{P}(X_t)$, of lifting v in Q.

```
Input: a subset $\mathcal{A}$ of $\mathcal{C}_s$
Output: a subset $\mathcal{B}$ of $\mathcal{C}_t$
let $\mathcal{B} = \emptyset$
for $(i, K, \mathcal{P}) \in \mathcal{A}$
    if $v \in R$ and $i < k$ then
      let $\mathcal{B} = \mathcal{B} \cup \{(i+1, K \cup \{v\}, \mathcal{P}')\}$
          where $\mathcal{P}' = \{P \in \mathcal{P}(X_t) \mid \zeta_{X_s \setminus K}(P) \cap \mathcal{P} \neq \emptyset\}$
    let $\mathcal{B} = \mathcal{B} \cup \{(i, K, \mathcal{P}'')\}$
          where $\mathcal{P}'' = \{P \in \mathcal{P}(X_t) \mid \zeta_{X_t \setminus K}(P) \cap \mathcal{P} \neq \emptyset\}$
```

Lemma 4.3.1. $\mathcal{F}_t =$ **make_join**$(\mathcal{F}_s)$.

Proof. We first prove that **make_join**$(\mathcal{F}_s) \subseteq \mathcal{F}_t$. Let $(i+1, K \cup \{v\}, \mathcal{P}) \in$ **make_join**$(\mathcal{F}_s)$ with $v \in R$ and $i < k$ (the other case is similar). We prove that $(i+1, K \cup \{v\}, \mathcal{P}) \in \mathcal{F}_t$.

By the operation of the procedure **make_join** we have that there exists a triple $(i, K, \mathcal{P}) \in \mathcal{F}_s$ such that $\mathcal{P}' = \{P \in \mathcal{P}(X_t) \mid \zeta_{X_s \setminus K}(P) \cap \mathcal{P} \neq \emptyset\}$. Let $S \subseteq \mathcal{R}_s^i$ be the annotated vertex set which justifies the existence of $(i, K, \mathcal{P})$ in $\mathcal{F}_s$, i.e. $X_s \cap S = K$ and **sig**$_s(S) = \mathcal{P}$. Now, let $S' = S \cup \{v\}$. Clearly, $S' \subseteq \mathcal{R}_t^{i+1}$ (where $i + 1 \leq k$) and $X_t \cap S' = K \cup \{v\}$.

It remains to show that **sig**$_t(S') = \mathcal{P}'$ or, equivalently, $\forall P \in \mathcal{P}(X_t)$ it holds that $P \in$ **sig**$_t(S') \Leftrightarrow \zeta_{X_s \setminus K}(P) \cap \mathcal{P} \neq \emptyset$. Let $P \in$ **sig**$_t(S')$. We distinguish three cases:

- **Case 1:** deg$_P(v) = 0$. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \emptyset)$ (notice that $(\{v\}, \emptyset) \in \mathcal{P}_v^{aux}$), which means that $P^* \in \zeta_{X_s \setminus K}(P)$. It is not hard to confirm that $P^* \in \mathcal{P}$ because S realizes P^* in G_s. It follows that $P \in \mathcal{P}'$.

- **Case 2:** deg$_P(v) = 1$. Let u be the only neighbor of v in P. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \{v, u\})$, which means that $P^* \in \zeta_{X_s \setminus K}(P)$. Again, $P^* \in \mathcal{P}$ because S realizes P^* in G_s, thus $P \in \mathcal{P}'$.

- **Case 3:** deg$_P(v) = 2$. Let $N_P(v) = \{u, w\}$. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \{\{v, w\}\{v, u\}\})$, which means that $P^* \in \zeta_{X_s \setminus K}(P)$. As before, S realizes P^* in G_s, thus $P \in \mathcal{P}'$.

We have proved that $\textbf{sig}_t(S') \subseteq \mathcal{P}'$. The converse, $\mathcal{P}' \subseteq \textbf{sig}_t(S')$, is clear from the definition of $\mathcal{P}'$.

To conclude the proof, we have to show that $\mathcal{F}_t \subseteq \textbf{make\_join}(\mathcal{F}_s)$. Let $(i, K, \mathcal{P}) \in \mathcal{F}_t$. From the definition of $\mathcal{F}_t$, there exists a vertex set $S \subseteq \mathcal{R}_t^i$ that realizes every pairing of $\mathcal{P}$ and $X_t \cap S = K$. Let $P \in \mathcal{P}$ and assume that $v \notin R$. We consider three cases and the arguments are similar to the previous ones:

- **Case 1:** $\text{deg}_P(v) = 0$. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \emptyset)$, which means that $P^* \in \zeta_{X_s \setminus K}(P)$.

- **Case 2:** $\text{deg}_P(v) = 1$. Let u be the only neighbor of v in P. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \{v, u\})$, which means that $P^* \in \zeta_{X_s \setminus K}(P)$.

- **Case 3:** $\text{deg}_P(v) = 2$. Let $N_P(v) = \{u, w\}$. Then, $P^* = P \setminus \{v\} \in X_s$ and $P = P^* \cup (\{v\}, \{\{v, w\}\{v, u\}\})$, which means that $P^* \in \zeta_{X_s \setminus K}(P)$.

Let $\mathcal{P}^* = \{P^* \in \mathcal{P}(X_s) \mid P \in \mathcal{P}\}$. Clearly, for $S^* = S \setminus \{v\} \subseteq \mathcal{R}_t^i$ and $K^* = X_s \cap S^*$, we have that $\textbf{sig}(S^*) = \mathcal{P}^*$ and thus $(i-1, K^*, \mathcal{P}^*) \in \mathcal{F}_s$.
The case where $v \in R$ is similar. We conclude that $\mathcal{F}_t \subseteq \textbf{make\_join}(\mathcal{F}_s)$, which completes the proof. □

• Node t is a **forget node**: Let $X_t = X_s \setminus \{v\}$, where s is the unique child of t in T. Then

$$\mathcal{F}_t = \{(i, K \setminus \{v\}, \mathcal{P}) \mid \exists (i, K, \mathcal{P}') \in \mathcal{F}_s : \forall P \in \mathcal{P}(X_t),\ P \in \mathcal{P} \Leftrightarrow \text{lift}(P, v) \in \mathcal{P}'\}$$

The proof that the set at the right part is equal to $\mathcal{F}_t$, is similar to the one of Lemma 4.3.1.

• Node t is a **join node**: Let s_1 and s_2 be the children of t in T. Thus, $X_t = X_{s_1} = X_{s_2}$ and clearly $\mathcal{P}(X_t) = \mathcal{P}(X_{s_1}) = \mathcal{P}(X_{s_2})$. Given a pairing $P \in \mathcal{P}(X_t)$, we define

$$\xi(P) = \{(P_1, P_2) \in \mathcal{P}(X_t) \times \mathcal{P}(X_t) \mid P_1 \cup P_2 = P\}$$

Then, $\mathcal{F}_t$ can be derived from $\mathcal{F}_{s_1}$ and $\mathcal{F}_{s_2}$ as follows:

$$\begin{aligned}\mathcal{F}_t \quad = \quad \{(i,K,\mathcal{P}) \mid \exists(i_1,K_1,\mathcal{P}_1) \in \mathcal{F}_{s_1}\ \exists(i_2,K_2,\mathcal{P}_2) \in \mathcal{F}_{s_2} :\\ i = i_1 + i_2 - |K_1 \cap K_2|,\\ K_1 \cup K_2 = K\\ \forall P \in \mathcal{P}\ \exists(P_1,P_2) \in \xi(P) : P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2\}.\end{aligned}$$

Lemma 4.3.2. *In the case where t is a join node with children s_1 and s_2, $\mathcal{F}_t$ is computed as described above, given $\mathcal{F}_{s_1}$ and $\mathcal{F}_{s_2}$.*

Proof. Let $\mathcal{U}_t = \{(i,K,\mathcal{P}) \mid \exists(i_1,K_1,\mathcal{P}_1) \in \mathcal{F}_{s_1}\quad \exists(i_2,K_2,\mathcal{P}_2) \in \mathcal{F}_{s_2} : i = i_1 + i_2 - |K_1 \cap K_2|,\quad K_1 \cup K_2 = K$ and $\forall P \in \mathcal{P}\quad \exists(P_1,P_2) \in \xi(P) : P_1 \in \mathcal{P}_1, P_2 \in \mathcal{P}_2\}$.

We will only prove the nontrivial direction: $\mathcal{F}_t \subseteq \mathcal{U}_t$. Let $(i,K,\mathcal{P}) \in \mathcal{F}_t$. From the definition of $\mathcal{F}_t$, there exists a vertex set $S \subseteq \mathcal{R}_t^i$ that realizes every pairing of $\mathcal{P}$ and $X_t \cap S = K$. Let P be any pairing of $\mathcal{P}$. Then, there exists a sub-cyclic pair $(\mathcal{Q},Z)$ on t that corresponds to pairing P. The restriction of $(\mathcal{Q},Z)$ in G_{s_1} (resp. G_{s_2}) is a sub-cyclic pair $(\mathcal{Q}_1,Z_1)$ on s_1 (resp. $(\mathcal{Q}_2,Z_2)$ on s_2) and clearly $Z_1 \subseteq X_{s_1}$ (resp. $Z_2 \subseteq X_{s_2}$). These sub-cyclic pairs meet some subsets S_1 and S_2 of S respectively and correspond to parings $P_1 \in \mathcal{P}_1$ and $P_2 \in \mathcal{P}_2$.

Let $|S_1| = i_1$ and $|S_2| = i_2$. It is now easy to confirm that $i = i_1 + i_2 - |K_1 \cap K_2|$, $K_1 \cup K_2 = Z_1 \cup Z_2 = K$ and that $(P_1,P_2) \in \xi(P)$.

As $P \in \mathcal{P}$ was chosen arbitrarily we conclude that $(i,K,\mathcal{P}) \in \mathcal{U}_t$ and we are done. $\square$

The algorithm that we described runs in $2^{2^{O(w \cdot \log w)}} \cdot n$ steps (where w is the width of the tree decomposition) and solves the Annotated Cyclability problem.

We insisted on a detailed presentation of the dynamic programming routine because (as we prove in Chapter 8) Cyclability (from the classical complexity point of view) can be classified to the second level of the polynomial hierarchy, and specifically to Π_2^{P}. Actually we believe that it is Π_2^{P}-complete, thus it does not belong to NP unless $\Pi_2^{\mathsf{P}} = \mathsf{NP}$. To our knowledge this is one of few, maybe the first, dynamic programming algorithm for such a problem.

CHAPTER 5 THE ALGORITHM

After constructing our combinatorial tools (Chapter 3) and designing an algorithm for efficiently solving Cyclability on graphs of bounded treewidth, we are in the position to construct an FPT-algorithm for solving Cyclability for planar graphs, as promised.

Thus, this section is devoted to the proof of Theorem 1.3.2. We consider the following, slightly more general, problem.

Planar Annotated Cyclability
Input: A plane graph G, a set $R \subseteq V(G)$, and a non-negative integer k.
Question: Does there exist, for every set S of k vertices in R, a cycle C of G such that $S \subseteq V(C)$?

In this section, for simplicity, we refer to Planar Annotated Cyclability as problem PAC. Theorem 1.3.2 follows directly from the following lemma.

Lemma 5.0.1. *There is an algorithm that solves* PAC *in* $2^{2^{O(k^2 \log k)}} \cdot n^2$ *steps.*

The rest of this section is devoted to the proof of Lemma 5.0.1.

Problem/colour-irrelevant vertices. Let (G, k, R) be an instance of PAC. We call a vertex $v \in V(G) \setminus R$ *problem-irrelevant* if (G, k, R) is a yes-instance if and only if $(G \setminus v, k, R)$ is a yes-instance. We call a vertex $v \in R$ *colour-irrelevant* when (G, k, R) is a yes-instance if and only if $(G, k, R \setminus \{v\})$ is a yes-instance.

Before we present the algorithm of Lemma 5.0.1, we need to introduce three algorithms that are used in it as subroutines.

Algorithm DP$(G, R, k, q, \mathcal{D})$
Input: A graph G, a vertex set $R \subseteq V(G)$, two non-negative integers k and q, where $k \leq q$, and a tree decomposition $\mathcal{D}$ of G of width q.
Output: An answer whether (G, R, k) is a yes-instance of PAC or not.
Running time: $2^{2^{O(q \cdot \log q)}} \cdot n$.

Algorithm **DP** is based on dynamic programming on tree decompositions of graphs and is the algorithm that we presented in Section 4.3 of Chapter 4.

Algorithm Compass(G, q)
Input: A planar graph G and a non-negative integer q.
Output: Either a tree decomposition of G of width at most $18q$ or a subdivided wall W of G of height q and a tree decomposition $\mathcal{D}$ of the compass K_W of W of width at most $18q$.
Running time: $2^{q^{O(1)}} \cdot n$.

We describe algorithm **Compass** in Section 5.1.

Algorithm concentric_cycles(G, R, k, q, W)
Input: A planar graph G, a set $R \subseteq V(G)$, a non-negative integer k, and a subdivided wall W of G of height at least $392k^2 + 40k$.
Output: Either a problem-irrelevant vertex v or a sequence $\mathcal{C} = \{C_1, C_2, \ldots, C_{98k+2}\}$ of concentric cycles of G, with the following properties:

(1) $\overline{C}_1 \cap R \neq \emptyset$.

(2) The set R is $32k$-dense in $\mathcal{C}$.

(3) There exists a sequence $\mathcal{W}$ of $2k + 1$ paths in K_W such that $(\mathcal{C}, \mathcal{W})$ is a $(98k + 2, 2k + 1)$-railed annulus.

Running time: $O(n)$.

We describe Algorithm **concentric_cycles** in Subsection 5.2. We now use the above three algorithms to describe the main algorithm of this paper which is the following.

Algorithm Planar_Annotated_Cyclability(G, R, k)
Input: A planar graph G, a set $R \subseteq V(G)$, and a non-negative integer k.
Output: An answer whether (G, R, k) is a yes-instance of PAC or not.
Running time: $2^{2^{O(k^2 \log k)}} \cdot n^2$.

Step 1. Let $r = 98k^2 + 2k$, $y = 16k$, and $q = 2y + 4r$. If **Compass**(G, q) returns a tree decomposition of G of width $w = 18q$, then return **DP**(G, R, k, w) and stop. Otherwise, the algorithm **Compass**(G, q) returns a subdivided wall W of G of height q and a tree decomposition $\mathcal{D}$ of the compass K_W of W of width at most w.

Step 2. If the algorithm **concentric_cycles**(G, R, k, q, W) returns a problem-irrelevant vertex v, then return **Planar_Annotated_Cyclability**$(G \setminus v, R \setminus v, k)$ and stop. Otherwise, it returns a sequence $\mathcal{C} = \{C_1, C_2, \ldots, C_r\}$ of concentric cycles of G with the properties (1)–(3).

Step 3. For every $i \in \{1, \ldots, r - 98k - 2\}$ let w_i be a vertex in $\hat{A}_{i+k,i+33\cdot k} \cap R$ (this vertex exists as, from property (2), R is $32k$-dense in $\mathcal{C}$), let $R_i = (R \cap V(\hat{C}_i)) \cup \{w_i\}$, and let $\mathcal{D}_i$ be a tree decomposition of $\hat{C}_i$ of width at most w – this tree decomposition can be constructed in linear time from $\mathcal{D}$ as each $\hat{C}_i$ is a subgraph of K_W.

Step 4. If, for some $i \in \{1, \ldots, r - 98k - 2\}$, the algorithm **DP**$(\hat{C}_i, R_i, k, q, \mathcal{D}_i)$ returns a negative answer, then return a negative answer and stop. Otherwise return **Planar_Annotated_Cyclability**$(G, R \backslash v, k)$ where $v \in V(\hat{C}_1) \cap R$ (the choice of v is possible due to property (1)).

For a visualisation of how our algorithm operates, see Figure 5.1.

Proof of Lemma 5.0.1. The only non-trivial step in the above algorithm is Step 4. Its correctness follows from Lemma 5.3.1, presented in Subsection 5.3.

We now proceed to the analysis of the running time of the algorithm. Observe first that the call of **Compass**(G, q) in Step 1 takes $2^{k^{O(1)}} \cdot n$ steps and, in the case that a tree decomposition is returned, the **DP** requires $2^{2^{O(k^2 \log k)}} \cdot n$ steps. For Step 2, the algorithm **concentric_cycles** takes $O(n)$ steps and if it returns a problem-irrelevant vertex, then the whole algorithm is applied again for a graph with one vertex less. Suppose now that Step 2 returns a sequence

$\mathcal{C}$ of concentric cycles of G with the properties (1)–(3). Then, the algorithm **DP** is called $O(k^2)$ times and this takes in total $2^{2^{O(k^2 \log k)}} \cdot n$ steps. After that, the algorithm either concludes to a negative answer or is called again with one vertex less in the set R. In both cases where the algorithm is called again we have that the quantity $|V(G)| + |R|$ is becoming smaller. This means that the recursive calls of the algorithm cannot be more than $2n$. Therefore, the total running time is bounded by $2^{2^{O(k^2 \log k)}} \cdot n^2$ as required.

5.1 The algorithm Compass

Before we start the description of algorithm **Compass** we present a result that follows from Proposition 2.4.1, the algorithms in [76] and [8], and the fact that finding a subdivision of a planar k-vertex graph H that has maximum degree 3 in a graph G can be done, using dynamic programming, in $2^{O(k\cdot\log k)}\cdot n$ steps (see also [2]).

Lemma 5.1.1. *There exists an algorithm A_1 that, given a graph G and an integer h, outputs either a tree decomposition of G of width at most $9h$ or a subdivided wall of G of height h. This algorithm runs in $2^{h^{O(1)}} \cdot n$ steps.*

Description of algorithm Compass. We use a routine, call it A_2, that receives as input a subdivided wall W of G with height equal to some even number h and outputs a subdivided wall W' of G such that W' has height $h/2$ and $|V(K_{W'})| \leq |V(G)|/4$. A_2 uses the fact that, in W, there are 4 vertex-disjoint subdivided subwalls of W of height $h/2$. Among them, A_2 outputs the one with the minimum number of vertices and this can be done in $O(n)$ steps. The algorithm **Compass** uses as subroutines the routine A_2 and the algorithm A_1 of Lemma 5.1.

```
Algorithm Compass($G, q$)
[Step 1.] if $A_1(G, 2q)$ outputs a tree decomposition $\mathcal{D}$ of $G$ with
              width at most $18q$ then return $\mathcal{D}$,
              otherwise it outputs a subdivided wall $W$ of $G$ of height $2q$
[Step 2.] Let $W' = A_2(W)$
              if $A_1(K_{W'}, 2q)$ outputs a tree decomposition $\mathcal{D}$ of
```

$K_{W'}$ of width at most $18q$ then return W' and $\mathcal{D}$, otherwise $W \leftarrow W'$ and go to Step 2.

Notice that, if A terminates after the first execution of Step 1, then it outputs a tree decomposition of G of width at most $18q$. Otherwise, the output is a subdivided wall W' of height q in G and a tree decomposition of $K_{W'}$ of width at most $18q$ (notice that as long as this is not the case, the algorithm keeps returning to step 2). The application of routine A_2 ensures that the number of vertices of every new K_W is at least four times smaller than the one of the previous one. Therefore, the i-th call of the algorithm A_1 requires $O(2^{h^{O(1)}} \cdot \frac{n}{2^{2(i-1)}})$ steps. As $\sum_{i=0}^{\infty} \frac{1}{2^{2i}} = O(1)$, algorithm **Compass** has the same running time as algorithm A_1.

5.2 The Algorithm concentric_cycles

We need to introduce two lemmata. The first one is strongly based on the combinatorial Lemma 3.2.4 that is the main result of Section 3.

Lemma 5.2.1. *Let (G, R, k) be an instance of* PAC *and let $\mathcal{C} = \{C_1, \ldots, C_r\}$ be a sequence of concentric cycles in G such that $V(\hat{C}_r) \cap R = \emptyset$. If $r \geq 16k$, then all vertices in $V(\hat{C}_1)$ are problem-irrelevant.*

Proof. We observe that for every vertex $v \in V(G)$, if $(G \setminus v, R, k) \in$ PAC then $(G, R, k) \in$ PAC because $G \setminus v$ is a subgraph of G and thus every cycle that exists in $G \setminus v$ also exists in G.

Assume now that $(G, R, k) \in$ PAC, let $v \in V(\hat{C}_1)$, and let $S \subseteq R$, $|S| \leq k$. We will prove that there exists a cycle in $G \setminus v$ containing all vertices of S.

As $(G, R, k) \in$ PAC, there is a cyclic linkage $\mathcal{L} = (C, S)$ in G. If $v \notin V(C)$, then C is a subgraph of $G \setminus v$ and we are done. Else, if $v \in V(C)$, let $\mathcal{L}' = (C', S)$ be a $\mathcal{C}$-weakly cheap cyclic linkage in the graph $H = G[V(C) \cup \left(\bigcup_{i=1}^{r} V(C_i)\right)]$, and assume that $v \in V(C')$ too. Then C' meets all cycles of $\mathcal{C}$ and its penetration in $\mathcal{C}$ is more than $16 \cdot |S|$, which contradicts Lemma 3.2.4.

Thus, $v \notin V(C')$ implying that there exists a cyclic linkage with S as its set of terminals that does not contain v. As S was arbitrarily chosen, vertex v is problem-irrelevant. □

Lemma 5.2.2. *Let y, r, q, z be positive integers such that $y+1 \leq z \leq r$, G be a graph embedded on $\mathbb{S}_0$ and let $R \subseteq V(G)$ be the set of annotated vertices of G. Given a subdivided wall W of height $h = 2 \cdot \max\{y, \lceil \frac{q}{8} \rceil\} + 4r$ in G then either G contains a sequence $\mathcal{C}' = \{C'_1, C'_2, \dots, C'_y\}$ of concentric cycles such that $V(\hat{C}'_y) \cap R = \emptyset$ or a sequence $\mathcal{C} = \{C_1, C_2, \dots, C_r\}$ of concentric cycles such that:*

1. *$\overline{C}_1 \cap R \neq \emptyset$.*
2. *R is z-dense in $\mathcal{C}$.*
3. *There exists a collection $\mathcal{W}$ of q paths in K_W, such that $(\mathcal{C}, \mathcal{W})$ is a (r,q)-railed annulus in G.*

Moreover, a sequence $\mathcal{C}'$ or $\mathcal{C}$ of concentric cycles as above can be constructed in $O(n)$ steps.

Proof. Let $p = \max\{y, \lceil \frac{q}{8} \rceil\}$. We are given a subdivided wall W of height $h = 2p + 4r$ and we define $\mathcal{C} = \{C_1, \dots, C_r\}$ such that $C_i = J_{\frac{h}{2}-p-2i+2}, i \in \{1, \dots, r\}$. Notice that there is a collection $\mathcal{W}$ of $8p$ vertex disjoint paths in W such that $(\mathcal{C}, \mathcal{W})$ is a (r,q)-railed annulus. If $\overline{C}_1 \cap R = \emptyset$, then

$$\mathcal{C}' = \{J_{\frac{h}{2}}, J_{\frac{h}{2}+1}, \dots, J_{\frac{h}{2}+y-1}\}$$

is a sequence of concentric cycles where $\overline{J}_{\frac{h}{2}+y-1} \subseteq \mathring{C}_1$ and we are done. Otherwise, we have that $\mathcal{C}$ satisfies property *1*.

Suppose now that Property *2* does not hold for $\mathcal{C}$. Then, there exists some $i \in \{1, \dots, r\}$ such that $A_{i,i+z-1} \cap R = \emptyset$. Notice that $A_{i,i+z-1}$ contains $2z-1 > 2y$ layers of W which are crossed by at least $2y$ of the paths in $\mathcal{W}$ (these paths certainly exist as $2y < 8p$). This implies the existence of a wall of height $2y$ in $A_{i,i+z-1}$ which, in turn contains a sequence $\mathcal{C}' = \{C'_1, \dots, C'_y\}$ of concentric cycles. As $\overline{C}'_y \subseteq A_{i,i+z-1}$ we have that $V(\hat{C}'_y) \cap R = \emptyset$ and we are done. It remains to verify property *3* for $\mathcal{C}$. This follows directly by including in $\mathcal{W}'$ any $q \leq 8p$ of the disjoint paths of $\mathcal{W}$. Then $(\mathcal{C}, \mathcal{W}')$ is the required (r,q)-railed annulus. It is easy to verify that all steps of this proof can be turned to an algorithm that runs in linear, on n, number of steps. □

Description of algorithm concentric_cycles. This algorithm first applies the algorithm of Lemma 5.2.2 for

$$y = 16k,\ r = 98k^2 + 2k,\ q = 2k + 1, \text{ and } z = 32k.$$

If the output is a sequence $\mathcal{C}' = \{C'_1, C'_2, \ldots, C'_y\}$ of concentric cycles such that $V(\hat{C}'_y) \cap R = \emptyset$, then it returns a vertex w of $\hat{C}'_1$. As $V(\hat{C}_r) \cap R = \emptyset$, Lemma 5.2.1 implies that w is problem-irrelevant. If the output is a sequence $\mathcal{C}$ then it remains to observe that conditions *1–3* match the specifications of algorithm **concentric_cycles**.

5.3 Correctness of the algorithm

As mentioned in the proof of Lemma 5.0.1, the main step – [*step 4*] – of algorithm **Planar_Annotated_Cyclability** is based on Lemma 5.3.1 below.

Lemma 5.3.1. *Let (G, R, k) be an instance of problem* PAC *and let $b = 98k + 2$ and $r = 98k^2 + 2k$. Let also $(\mathcal{C}, \mathcal{W})$ be a $(r, 2k+1)$-railed annulus in G, where $\mathcal{C} = \{C_1, \ldots C_r\}$ is a sequence of concentric cycles such that $\hat{C}_1$ contains some vertex $v \in R$ and that R is $32k$-dense in $\mathcal{C}$. For every $i \in \{1, \ldots, r - b\}$, let*

$$R_i = (R \cap V(\hat{C}_i)) \cup \{w_i\} \text{ where } w_i \in V(\hat{A}_{i+k+1,33k+i+1}) \cap R.$$

If $(\hat{C}_{i+b}, R_i, k)$ is a no*-instance of* PAC*, for some $i \in \{1, \ldots, r-b\}$, then (G, R, k) is a* no*-instance of* PAC*. Otherwise vertex v is* colour-irrelevant*.*

We first prove the following lemma, which reflects the use of the rails of a railed annulus and is crucial for the proof of Lemma 5.3.1.

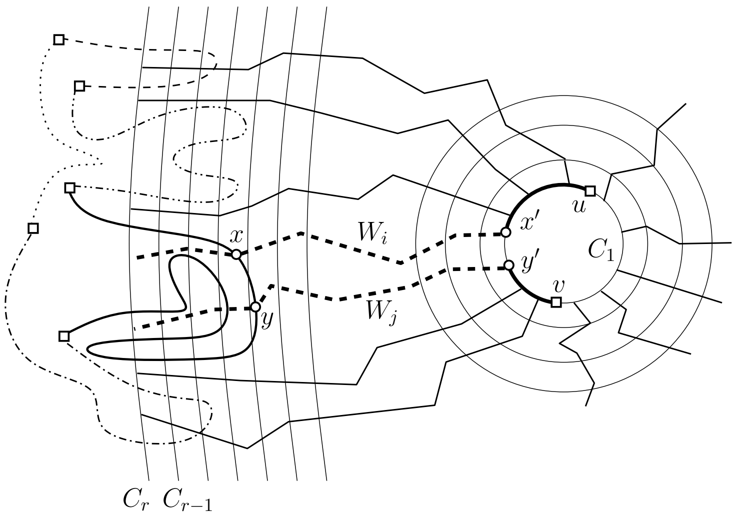


Figure 5.2: visualisation of proof of Lemma 5.3.2, case 1. The different lining on the parts of the cycle at the left indicates the different colours of these paths.

Lemma 5.3.2. *Let G be a graph embedded on the sphere $\mathbb{S}_0$, r, k be two positive integers such that $r \geq 16k$, and $(\mathcal{C}, \mathcal{W})$ be an $(r, 2k+1)$-railed annulus of G with $\mathcal{C} = \{C_1, \dots, C_r\}$ being its sequence of concentric cycles, $\mathcal{W} = \{W_1, \dots, W_{2k+1}\}$ its rails. Let also $S \subseteq V(G)$ such that $S \cap \hat{C}_r = \emptyset$ and $|S| = k$. Then for every two vertices $u, v \in V(C_1)$, if there exists a cyclic linkage $\mathcal{L} = (C, S)$, with penetration $k+1 \leq p_{\mathcal{C}}(\mathcal{L}) \leq r-1$, in G, then there exists a path $P_{u,v}$ with ends u and v that meets all vertices of S.*

Proof. Let $\{s_1, \dots, s_k\}$ be an ordering of the set S and let $f_{\mathcal{L}} : \mathcal{L}(\mathcal{P}) \to \{1, \dots, k\}$ be a function such that for every $i \in \{1, \dots k-1\}$, $f_{\mathcal{L}}(P) = i$ if the endpoints of P are s_i and s_{i+1} and $f_{\mathcal{L}}(P^*) = k$ for the unique path $P^* \in \mathcal{P}(\mathcal{L})$ whose endpoints are s_k and s_1.

Moreover, as W_i is a path with endpoints $w_i' \in V(C_1)$ and $w_i'' \in V(C_r)$, we define the ordering $\{w_i', \dots, w_i''\}$ of $V(W_i)$ and call it the natural ordering of W_i. Furthermore, for every $W_i \in \mathcal{W}$, let $m_{\mathcal{L}}(W_i) = f_{\mathcal{L}}(P)$ if P is the first path (with respect to the natural ordering of W_i) of $\mathcal{P}(\mathcal{L})$ that W_i meets and $m_{\mathcal{L}}(W_i) = 0$

if W_i does not meet C.

Let $C_j \in \mathcal{C}$. We pick an arbitrary vertex $v_1^j \in V(C_j)$ and order $V(C_j)$ starting from v_1^j and continuing in clockwise order. Let $\{v_1^j, \dots, v_{|V(C_j)|}^j\}$ be such an ordering of the vertices of C_j. We assign to each vertex of $v_i^j \in C_j$ a "colour" from the set $\{0, \dots k\}$ as follows: $c_{\mathcal{L}}(v_i^j) = 0$ if $v_i^j \notin V(C_j) \cap V(C)$ and $c_{\mathcal{L}}(v_i^j) = f_{\mathcal{L}}(P)$ if $v_i^j \in V(C_j) \cap V(P)$, where $P \in \mathcal{P}(\mathcal{L})$.

For the rest of the proof, if P_0 is a path, $P_0(v, w)$ is the subpath of P_0 with endpoints v and w. We examine two cases:

1. At least $k+1$ paths of $\mathcal{W}$ (i.e. rails of the railed annulus) meet C. Then, as $|\mathcal{P}(\mathcal{L})| = k$, there exist two rails $W_i, W_j \in \mathcal{W}$ and a path $P \in \mathcal{P}(\mathcal{L})$ such that
$$m_{\mathcal{L}}(W_i) = m_{\mathcal{L}}(W_j) = f_{\mathcal{L}}(P).$$
Let $V(C_1) \cap V(W_i)$ be the vertices of path $Q_{1,i}$ and $V(C_1) \cap V(W_j)$ the vertices of path $Q_{1,j}$. Then, we let $x \in V(C_1)$ be the endpoint of $Q_{1,i}$ that is not w_i' and $y \in V(C_1)$ be the endpoint of $Q_{1,j}$ that is not w_j' (notice that x and y can coincide with u and v). Let also x' be the vertex of $V(P) \cap V(W_i)$ with the least index in the natural ordering of W_i and y' be the vertex of $V(P) \cap V(W_j)$ with the least index in the natural ordering of W_j. We observe that there exist two vertex disjoint paths P_1 and P_2 with endpoints either v, x and u, y or v, y and u, x, respectively. We define path
$$P_{u,v} = (C \setminus P(x', y')) \cup W_i(x, x') \cup W_j(y, y') \cup P_1 \cup P_2.$$
Path $P_{u,v}$ has the desired properties. See also Figure 5.2.

2. There exist $k' = k+1$ paths, say $\mathcal{W}' = \{W_1, \dots, W_{k'}\}$, of $\mathcal{W}$ that do not meet C. As the penetration of C is at least $k+1$, for every $j \in \{r-k, \dots, r\}$, $V(C_j \cap C) \neq \emptyset$. For every $i \in \{1, \dots, k'\}$ and every $j \in \{r-k, \dots, r\}$ we assign to the vertex w_i^j of $V(W_i \cap C_j)$ with the least index in the natural ordering of W_i, a "colour" from the set $\{1, \dots, k\}$ as follows: $c_{\mathcal{L}}(w_i^j) = c_{\mathcal{L}}(v)$ if there exists a $v \in V(C)$ and a subpath $C_j(w_i^j, v)$ (starting from w_i^j and following C_j in counter-clockwise order) such that it does not contain any other vertices of $V(C)$ as internal vertices.

For every $W_i \in \mathcal{W}'$, we assign to W_i a set of colours, $\chi_i = \bigcup_{j=1}^{k+1} c_{\mathcal{L}}(w_i^j)$. Let $\mathcal{P}$ be the set of all maximal paths of C_r without internal vertices in C. Certainly, any $W_i \in \mathcal{W}'$ intersects exactly one path of $\mathcal{P}$. We define the equivalence relation $\sim$ on the set of rails $\mathcal{W}'$ as follows: $W_i \sim W_l$ if and only if W_i and W_l intersect the same path of $\mathcal{P}$. We distinguish two subcases:

- The number of equivalence classes of $\sim$ is k'. Then, there exist two rails $W_i, W_l \in \mathcal{W}'$ and $j_i, j_l \in \{r-k, \ldots, r\}$ such that $c_{\mathcal{L}}(w_i^{j_i}) = c_{\mathcal{L}}(w_l^{j_i}) = c_{\mathcal{L}}(P)$ for some path $P \in \mathcal{P}(\mathcal{L})$.
- The number of equivalence classes of $\sim$ is strictly less than k'. Then, there exist two rails $W_i, W_l \in \mathcal{W}'$ such that $c_{\mathcal{L}}(w_i^j) = c_{\mathcal{L}}(w_l^j)$ for every $j \in \{r-k, \ldots, r\}$. Therefore, there exist $j_i, j_l \in \{r-k, \ldots, r\}$ with $j_i \neq j_l$ such that

$$c_{\mathcal{L}}(w_i^{j_i}) = c_{\mathcal{L}}(w_l^{j_l}) = c_{\mathcal{L}}(P)$$

  for some path $P \in \mathcal{P}(\mathcal{L})$ (this holds because $|\{r-k, \ldots, r\}| = k+1$ – see also Figure 5.3).

For both subcases, as $c_{\mathcal{L}}(w_i^{j_i}) = c_{\mathcal{L}}(P)$, there exist a $v_j \in V(P)$ and a subpath $C_j(w_i^{j_i}, v_j)$ of C_j and, similarly, as $c_{\mathcal{L}}(w_l^{j_l}) = c_{\mathcal{L}}(P)$, there exist a $v_{j_l} \in V(P)$ and a subpath $C_j(w_l^{j_l}, v_{j_l})$ of C_j. These two subpaths do not contain any other vertices of C apart from v_{j_i} and v_{j_l}, respectively. Moreover, let x be the vertex of $V(W_i \cap C_1)$ of the least index in the natural ordering of W_i and y the vertex of $V(W_l \cap C_1)$ of the least index in the natural ordering of W_l. As in case 1, observe that there exist two vertex disjoint paths P_1 and P_2 with endpoints either v, x and u, y or v, y and u, x, respectively. We define the path

$$\begin{aligned} P_{u,v} \quad = \quad & (C \setminus P(v_{j_i}, v_{j_l})) \cup C_j(w_i^{j_i}, v_{j_i}) \cup C_j(w_l^{j_l}, v_{j_l}) \cup \\ & \cup W_i(w_i^{j_i}, x) \cup W_l(w_l^{j_l}, y) \cup P_1 \cup P_2. \end{aligned}$$

Path $P_{u,v}$ has the desired properties.

□

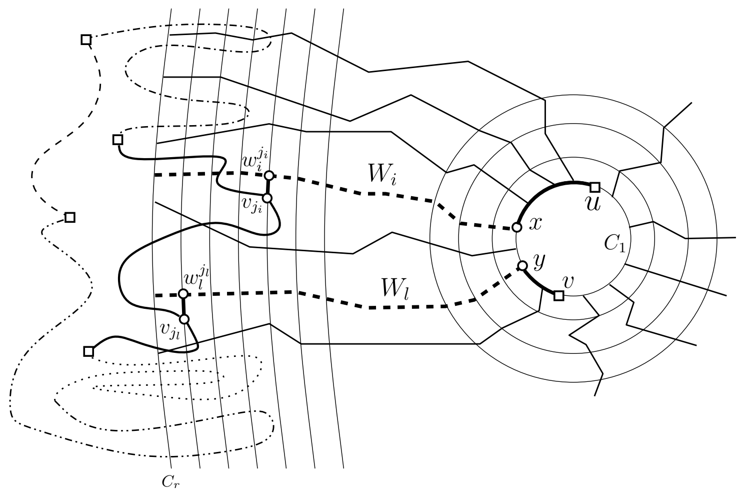


Figure 5.3: visualisation of proof of Lemma 5.3.2, case 2, subcase 2.

Proof of Lemma 5.3.1. We first prove that if $(\hat{C}_{i+b}, R_i, k)$ is a yes-instance of PAC for every $i \in \{1, \ldots, r-b\}$, then (G, R, k) is a yes-instance of PAC iff $(G, R \setminus v, k)$ is a yes-instance of PAC.

For the non-trivial direction, we assume that $(G, R \setminus v, k)$ is a yes-instance of PAC and we have to prove that (G, R, k) is also a yes-instance of PAC. Let $S \subseteq R$ with $|S| \leq k$. We have to prove that S is cyclable in G. We examine two cases:

1. $v \notin S$. As $(G, R \setminus v, k)$ is a yes-instance of PAC, clearly there exists a cyclic linkage $\mathcal{L} = (C, S)$ in G, i.e., S is cyclable in G.

2. $v \in S$. As $r \geq k(98k+1)$ and $S \leq k$, there exists i such that $A_{i,i+98k} \cap S = \emptyset$. We distinguish two sub-cases:

   - *Subcase 1.* $S \subseteq \overline{C}_{i+98k+1}$. Then, as $(\hat{C}_{i+98k+1}, R_{i+98k+1}, k)$ is a yes-instance of PAC, then S is cyclable in $\hat{C}_{i+98k+1}$ and therefore also in G.

   - *Subcase 2.* There is a partition $\{S_1, S_2\}$ of S into two non-empty sets, such that $S_1 \subset \mathring{C}_i$ and $S_1 \cap \overline{C}_{i+98k+1} = \emptyset$. As R is $32k$-dense in $\mathcal{C}$, there exist vertices

$$v_1 \in S \cap A_{i+k+1,i+33k+1} \text{ and } v_2 \in S \cap A_{50+k+1,i+82k+1}.$$

For $i \in \{1,2\}$, let $S_i' = S_i \cup \{v_i\}$ and observe that $|S_i| \leq k$. Let

$$\mathcal{C}_1 = \{C_{i+49k}, \ldots, C_i\} \text{ and } \mathcal{C}_2 = \{C_{i+49k}, \ldots, C_{98k}\}.$$

As $(\hat{C}_{i+98k+1}, R_{98k+1}, k)$ is a yes-instance of PAC, S_1' is cyclable in $\hat{C}_{i+98k+1}$. Also, $(G, R \setminus v, k)$ is a yes-instance, S_2' is cyclable in G. For each $i \in \{1,2\}$, there exists a cyclic linkage $\mathcal{L}_i = (C_i, S_i')$ that has penetration at least $k+1$ in $\mathcal{C}_i$. We may assume that $\mathcal{L}_i$ is $\mathcal{C}_i$-cheap. Then, By Lemma 5.2.1, the penetration of $\mathcal{L}_i$ in $\mathcal{C}_i$ is at most $49k$. Let $\mathcal{L}_i' = (C_i, S_i), i \in \{1,2\}$. For notational convenience we rename $\mathcal{C}_1$ and $\mathcal{C}_2$ where $\mathcal{C}_1 = \{C_1^1, \ldots, C_{49k+1}^1\}$ and $\mathcal{C}_2 = \{C_1^2, \ldots, C_{49k+1}^2\}$ (notice that $C_{49k+1}^1 = C_1^2$). Let x, y be two distinct vertices in C_{i+49k}. For $i \in \{1,2\}$, we apply Lemma 5.3.2, for $r = 49k+1, k, \mathcal{C}_i, \mathcal{W}$, and x and y and obtain two paths $P_i, i \in \{1,2\}$, such that $S_i \subseteq V(P_i)$ and whose endpoints are x and y. Clearly, $P_1 \cup P_2$ is a cycle whose vertex set contains S as a subset. Therefore S is cyclable in G, as required (see Figure 5.4).

□

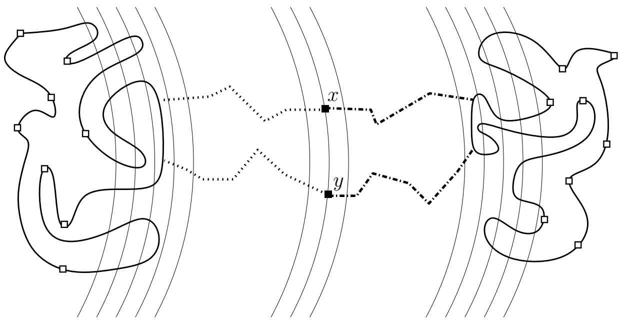


Figure 5.4: The squares of the right (resp. left) part represent the vertices of S_1 (resp. S_2). The connection between two cycles via rails and through x and y is derived from a double application of Lemma 5.3.2.

We have now concluded the presentation and analysis of the FPT-algorithm for solving the Cyclability problem on planar graphs. The next two chapters are devoted to our negative results:

In Chapter 6 we prove that Cyclability is hard (unlikely to be in FPT) when we allow the input to be any graph (even if it is a split graph), and in Chapter 7 we show that it is unlikely that Cyclability admits any polynomial kernel, even for the class of planar graphs.

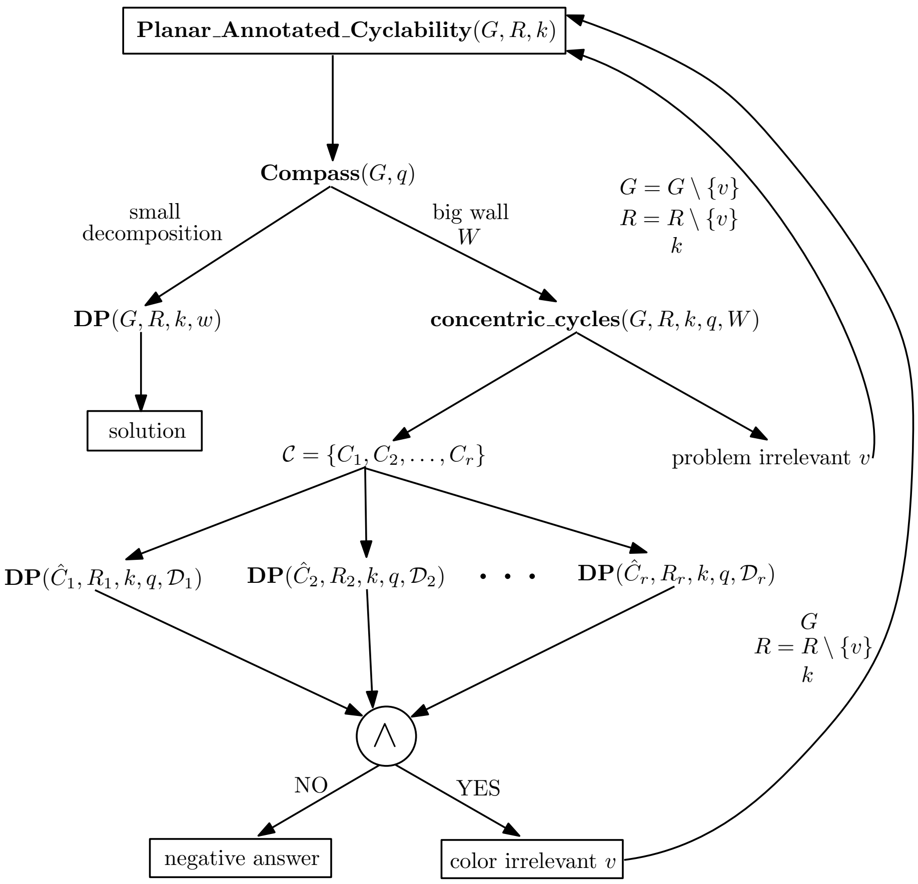

Planar_Annotated_Cyclability(G, R, k)
Compass(G, q)
small decomposition
big wall W
$G = G \setminus \{v\}$
$R = R \setminus \{v\}$
k
DP(G, R, k, w)
concentric_cycles(G, R, k, q, W)
solution
$\mathcal{C} = \{C_1, C_2, \ldots, C_r\}$
problem irrelevant v
DP$(\hat{C}_1, R_1, k, q, \mathcal{D}_1)$
DP$(\hat{C}_2, R_2, k, q, \mathcal{D}_2)$
DP$(\hat{C}_r, R_r, k, q, \mathcal{D}_r)$
G
$R = R \setminus \{v\}$
k
$\wedge$
NO
YES
negative answer
color irrelevant v

Figure 5.1: A visualisation of how our algorithm, **Planar_ Annotated_ Cyclability**, operates on input (G, R, k) for the Cyclability problem, where $G = (V, E)$ is a planar graph, S is a subset of V, and k is a non-negative integer.

CHAPTER 6 HARDNESS OF THE CYCLABILITY PROBLEM

In this Chapter, we prove our first negative result, implying that Cyclability is hard for general graphs. This, in way, justifies why our effort for designing an FPT-algorithm for the planar case is worth making.

We show that it is unlikely that Cyclability is FPT by proving Theorem 1.3.1 (mentioned in the introduction). For this, we first introduce some further notation.

A *matching* is a set of pairwise non-adjacent edges. A vertex v is *saturated* in a matching M if v is incident to an edge of M. By $x_1 \dots x_p$ we denote the path with the vertices $x_1, \dots, x_p$ and the edges $\{x_1, x_2\}, \dots, \{x_{p-1}, x_p\}$, and we use $x_1 \dots x_p x_1$ to denote the cycle with the vertices $x_1, \dots, x_p$ and the edges $\{x_1, x_2\}, \dots, \{x_{p-1}, x_p\}, \{x_p, x_1\}$. For a path $P = x_1 \dots x_p$ and a vertex y, yP (Py resp.) is the path $yx_1 \dots x_p$ ($x_1 \dots x_p y$ resp.). If $P_1 = x_1 \dots x_p$ and $P_2 = y_1 \dots y_q$ are paths such that $V(P_1) \cap V(P_2) = \{x_p\} = \{y_1\}$, then $P_1 + P_2$ is the *concatenation* of P_1 an P_2, i.e., the path $x_1 \dots x_{p-1} y_1 \dots y_q$.

We need some auxiliary results. The following lemma is due to Erdős [38].

Define the function $f(n,\delta)$ by

$$f(n,\delta) = \begin{cases} \binom{n-\delta}{2} + \delta^2 & \text{if } n \geq 6\delta - 2, \\ \binom{(n+1)/2}{2} + (\frac{n-1}{2})^2 & \text{if } n \leq 6\delta - 3 \text{ and } n \text{ is odd}, \\ \binom{(n+2)/2}{2} + (\frac{n-2}{2})^2 & \text{if } n \leq 6\delta - 4 \text{ and } n \text{ is even}. \end{cases}$$

Lemma 6.0.1 ([38]). *Let G be a graph with $n \geq 3$ vertices. If $\delta(G) \geq n/2$ or $|E(G)| > f(n, \delta(G))$, then G is Hamiltonian.*

Lemma 6.0.2. *Let $k \geq 75$ be an odd integer and let H be a graph such that*

i) $(k-2)(k-3)/2 < |E(H)| \leq k(k-1)/2 + 1$,

ii) $\delta(H) \geq (k-1)/2$,

iii) there is a set $S \subseteq E(H)$ such that $|S| > (k-2)(k-3)/2$ and $G[S]$ has at most $k+2$ vertices.

Then H is Hamiltonian.

Proof. Let H be an n-vertex graph that satisfies the above three conditions. Let $S \subseteq E(H)$ be a set such that $|S| > (k-2)(k-3)/2$ and $G[S]$ has at most $k+2$ vertices. Let also $U = V(H) \setminus V(G[S])$. Denote by R the set of edges of G incident to vertices of U. Since $|S| > (k-2)(k-3)/2$ and $|E(H)| \leq k(k-1)/2 + 1$, $|R| \leq 2k - 3$. Because $\delta(H) \geq (k-1)/2$, $|R| \geq |U|\delta(H)/2 \geq |U|(k-1)/4$. We have that $|U| \leq 7$, i.e., H has at most $k+9$ vertices. Then because $k \geq 75$, we obtain that $n \geq 6\delta(G) - 3$,

$$\binom{(n+1)/2}{2} + \left(\frac{n-1}{2}\right)^2 \leq \frac{(k-2)(k-3)}{2} < |E(H)|$$

and

$$\binom{(n+2)/2}{2} + \left(\frac{n-2}{2}\right)^2 \leq \frac{(k-2)(k-3)}{2} < |E(H)|.$$

We have that $|E(H)| > f(n, \delta(H))$, and by Lemma 6.0.1, H is Hamiltonian. □

We are now in the position to prove Theorem 1.3.1:

Proof of Theorem 1.3.1. We reduce the Clique problem. Recall that Clique asks for a graph G and a positive integer k, whether G has a clique of size k.

This problem is well known to be W[1]-complete [35] when parameterized by k. Notice that Clique remains W[1]-complete when restricted to the instances where k is odd. To see it, it is sufficient to observe that if the graph G' is obtained from a graph G by adding a vertex adjacent to all the vertices of G, then G has a clique of size k if and only if G' has a clique of size $k+1$. Hence, any instance of Clique can be reduced to the instance with an odd value of the parameter. Clearly, the problem is still W[1]-hard if the parameter $k \geq c$ for any constant c.

Let (G,k) be an instance of Clique where $k \geq 75$ is odd. We construct the graph G'_k as follows.

- For each vertex $x \in V(G)$, construct $s=(k-1)/2$ vertices v_x^i for $i \in \{1,\ldots,s\}$ and form a clique of size ns from all these vertices by joining them by edges pairwise.
- Construct a vertex w and edges $\{w, v_x^i\}$ for $x \in V(G)$, $i \in \{1,\ldots,s\}$.
- For each edge $\{x,y\} \in E(G)$, construct the vertex u_{xy} and the edges $\{u_{xy}, v_x^i\}$, $\{u_{xy}, v_y^i\}$ for $i \in \{1,\ldots,s\}$; we assume that $u_{xy}=u_{yx}$.

Let $k'=k(k-1)/2+1$. It is straightforward to see that G' is a split graph. We show that G has a clique of size k if and only if there are k' vertices in G'_k such that there is no cycle in G'_k that contains these k' vertices.

Suppose that G has a clique X of size k. Let

$$Y=\{u_{xy} \in V(G') | x,y \in X, x \neq y\}$$

and $Z = Y \cup \{w\}$. Because $|X|=k$, $|Z|=k(k-1)/2+1=k'$. Observe that Y is an independent set in G'_k and $|Y|=|N_{G'}(Y)|$. Hence, for any cycle C in G'_k such that $Y \subseteq V(C)$, $V(C) \subseteq Y \cup N_{G'_k}(Y)$. Because $w \notin Y \cup N_{G'_k}(Y)$, w does not belong to any cycle that contains the vertices of Y. We have that no cycle in G'_k contains Z of size k'.

Now we show that if G has no cliques of size k, then for any $Z \subseteq V(G'_k)$ of size k', there is a cycle C in G'_k such that $Z \subseteq V(C)$. We use the following claim.

Claim. *Suppose that G has no cliques of size k. Then for any non-empty $Z \subseteq \{u_{xy} | x,y \in V(G)\}$ of size at most $k(k-1)/2+1$, there is a cycle C in*

G'_k such that $Z \subseteq V(C) \subseteq Z \cup N_G(Z)$ and C has an edge $\{v_x^i, v_y^j\}$ for some $x, y \in V(G)$ and $i, j \in \{1, \dots, s\}$.

of Claim. For a set $Z \subseteq \{u_{xy} | x, y \in V(G)\}$, we denote by $S(Z)$ the set of edges $\{\{x, y\} \in E(G) | u_{xy} \in Z\}$, and $H(Z) = G[S(Z)]$.

If $Z = \{u_{xy}\}$, then the triangle $u_{xy}v_x^1v_x^2u_{xy}$ is a required cycle, and the claim holds. Let $r = |Z| \geq 2$ and assume inductively that the claim is fulfilled for smaller sets.

Suppose that $H(Z)$ has a vertex x with $\deg_{H(Z)}(x) \leq (k-3)/2$. Let $N_{H(Z)}(x) = \{y_1, \dots, y_t\}$. Notice that $t \leq (k-3)/2 = s-1$. Denote by Z' the set obtained from Z by the deletion of $u_{xy_1}, \dots, u_{xy_t}$, and let $H' = H(Z')$. If $Z' = \emptyset$, then the cycle

$$C = v_x^1 u_{xy_1} v_x^2 \dots v_x^t u_{xy_t} v_x^{t+1} v_x^1$$

satisfies the conditions and the claim holds. Suppose that $Z' \neq \emptyset$. Then, by induction, there is a cycle C' in G'_k such that $Z \subseteq V(C') \subseteq Z \cup N_G(Z)$ and C' has an edge $\{v_a^i, v_b^j\}$ for some $a, b \in V(G)$ and $i, j \in \{1, \dots, s\}$. We consider the path

$$P = v_x^1 u_{xy_1} v_x^2 \dots v_x^t u_{xy_t} v_x^{t+1}.$$

Then we delete $\{v_a^i, v_b^j\}$ and replace it by the path $v_a^i P v_b^j$. Denote the obtained cycle by C. It is straightforward to verify that $Z \subseteq V(C) \subseteq Z \cup N_G(Z)$ and $\{v_a^i, v_x^1\} \in E(C)$, i.e., the claim is fulfilled.

From now we assume that $\delta(H(Z)) \geq (k-1)/2$. We consider three cases.

Case 1. $r \leq (k-2)(k-3)/2$.

Consider the graph G'_{k-2}. We show that this graph has a matching M of size r such that every vertex of Z is saturated in M. By the Hall's theorem (see, e.g., [29]), it is sufficient to show that for any $Z' \subseteq Z$, $|Z'| \leq |N_{G'_{k-2}}(Z')|$. Let p be the smallest positive integer such that $|Z'| \leq p(p-1)/2$. By the definition of G'_{k-2}, $|N_{G'_{k-2}}(Z')| \geq p(k-3)/2$. Because $p \leq k-2$, we have that

$$|Z'| \leq p(p-1)/2 \leq p(k-3)/2 \leq |N_{G'_{k-2}}(Z')|.$$

Let M be a matching in G'_{k-2} of size r such that every vertex of Z is saturated in M. Clearly, M is a matching in G'_k that saturates Z as well. Let

$x_1, \ldots, x_q$ be the vertices of G such that for $i \in \{1, \ldots, q\}$, $\{v^1_{x_i}, \ldots, v^s_{x_i}\}$ contains saturated in M vertices. Because $v^1_{x_i}, \ldots, v^s_{x_i}$ have the same neighbourhoods, we assume without loss of generality that for $i \in \{1, \ldots, q\}$, $v^1_{x_i}, \ldots, v^{t_i}_{x_i}$ are saturated. Observe that since M is a matching in G'_{k-2}, $t_i \leq s-1$. For $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, t_i\}$, denote by u^j_i the vertex of Z such that $\{v^j_{x_i}, u^j_i\} \in M$. We define the path

$$P_i = v^1_{x_i} u^1_i v^2_{x_i} \ldots u^{t_i}_i v^{t_i+1}_{x_i}, \text{ for every } i \in \{1, \ldots, q\}.$$

As all the vertices $v^j_{x_i}$ are pairwise adjacent, by adding the edges

$$\{v^{t_1+1}_{x_1}, v^1_{x_2}\}, \ldots, \{v^{t_{q-1}+1}_{x_{q-1}}, v^1_{x_q}\}, \{v^{s_q+1}_{x_q}, v^1_{x_1}\},$$

we obtain from the the paths $P_1, \ldots, P_q$ a cycle. Denote it by C. We have that

$$Z \subseteq V(C) \subseteq Z \cup N_G(Z) \text{ and } \{v^{t_1+1}_{x_1}, v^1_{x_2}\} \in E(C),$$

and we conclude that the claim holds.

Case 2. $(k-2)(k-3)/2 < r$ and for any $S \subseteq E(H(Z))$ such that $|S| > (k-2)(k-3)/2$, $H(Z)[S]$ has at least $k+3$ vertices.

We use the same approach as in Case 1 and show that G'_{k-2} has a matching M of size r such that every vertex of Z is saturated in M. We have to show that for any $Z' \subseteq Z$, $|Z'| \leq |N_{G'_{k-2}}(Z')|$. If $|Z'| \leq (k-2)(k-3)/2$, we use exactly the same arguments as in Case 1. Suppose that $|Z'| > (k-2)(k-3)/2$. Then

$$|S(Z')| = |Z'| > (k-2)(k-3)/2.$$

Hence, $H(Z)[S(Z')]$ has at least $k+3$ vertices. It implies that

$$|N_{G'_{k-2}}(Z')| \geq (k+3)(k-3)/2.$$

Because $k \geq 75$ and $|Z'| \leq r \leq k(k-1)/2+1$, we get that

$$|N_{G'_{k-2}}(Z')| \geq (k+3)(k-3)/2 \geq k(k-1)/2+1 \geq |Z'|.$$

Given a matching M that saturates Z, we construct a cycle that contains Z in exactly the same way as in Case 1 and prove that the claim holds.

Case 3. $(k-2)(k-3)/2 < r$ and there is $S \subseteq E(H(Z))$ such that $|S| > (k-2)(k-3)/2$ and $H(Z)[S]$ has at most $k+2$ vertices.

By Lemma 6.0.2, $H(Z)$ is Hamiltonian. Let $p = |V(H(Z))|$ and denote by $R = x_1 \dots x_p x_1$ a Hamiltonian cycle in $H(Z)$. Let $U = \{u_{x_1x_2}, \dots, u_{x_{p-1}x_1}\}$ and let $Z' = Z \setminus U$.

We again consider G'_{k-2}. We show that this graph has a matching M of size $|Z'|$ such that every vertex of Z' is saturated in M. We have to prove that for any $Z'' \subseteq Z'$, $|Z''| \leq |N_{G'_{k-2}}(Z'')|$. If $|Z''| \leq (k-2)(k-3)/2$, we use exactly the same arguments as in Case 1. Suppose that $|Z''| > (k-2)(k-3)/2$. Let q be the smallest positive integer such that $|Z''| \leq q(q-1)/2$. Clearly, $q > k-2$. We consider the following three cases depending on the value of q.

Case a. $q = k-1$. Then $H(Z'')$ has at least $k-1$ vertices and at least $(k-2)(k-3)/2+1$ edges. Because $|Z| \leq k(k-1)/2+1$, $H(Z)$ has at most $2k-3$ edges that are not edges of $H(Z'')$. Because $\delta(H(Z)) \geq (k-1)/2$ and $k \geq 75$, $H(Z)$ has at most 4 vertices that are not adjacent to the edges of $H(Z'')$. Then at most 8 edges of the Hamiltonian cycle R in $H(Z)$ do not join vertices of $H(Z'')$ with each other. We obtain that at least $k-9$ edges of R join vertices of $H(Z'')$ with each other.

Suppose that $H(Z'')$ has $k-1$ vertices. Then

$$|Z''| \leq (k-1)(k-2)/2 - (k-9) \leq (k^2-5k+20)/2.$$

Because $H(Z'')$ has $k-1$ vertices, $|N_{G'_{k-2}}(Z'')| = (k-1)(k-3)/2$. Since $k \geq 75$, $|Z''| \leq |N_{G'_{k-2}}(Z'')|$.

Suppose that $H(Z'')$ has k vertices. If $H(Z)$ has a vertex x that is not adjacent to the edges of $H(Z'')$, then at least $(k-1)/2$ vertices of Z that correspond to the edges incident to x are not in Z''. Then

$$|Z''| \leq |Z| - (k-1)/2 - (k-9) \leq (k^2-4k+21)/2.$$

Because $|N_{G'_{k-2}}(Z'')| = k(k-3)/2$ and $k \geq 75$, $|Z''| \leq |N_{G'_{k-2}}(Z'')|$. If $H(Z)$ has no vertex that is not adjacent to the edges of $H(Z'')$, then the edges of R join vertices of $H(Z'')$ with each other. We have that

$$|Z''| \leq k(k-1)/2 - k = k(k-3)/2 \text{ and } |Z''| \leq |N_{G'_{k-2}}(Z'')|.$$

Finally, if $H(Z'')$ has at least $k+1$ vertices, then

$$|N_{G'_{k-2}}(Z'')| \geq (k+1)(k-3)/2 \geq (k-1)(k-2)/2 \geq |Z''|.$$

Case b. $q = k$. Then $H(Z'')$ has at least k vertices and at least $(k-1)(k-2)/2+1$ edges. Because $|Z| \leq k(k-1)/2+1$, $H(Z)$ has at most $k-1$ edges that are not edges of $H(Z'')$. Because $\delta(H(Z)) \geq (k-1)/2$ and $k \geq 75$, $H(Z)$ has at most 2 vertices that are not adjacent to the edges of $H(Z'')$. Then at most 4 edges of the Hamiltonian cycle R in $H(Z)$ do not join vertices of $H(Z'')$ with each other. We obtain that at least $k-4$ edges of R join vertices of $H(Z'')$ with each other.

Suppose that $H(Z'')$ has k vertices. If $H(Z)$ has a vertex x that is not adjacent to the edges of $H(Z'')$, then at least $(k-1)/2$ vertices of Z that correspond to the edges incident to x are not in Z''. Then

$$|Z''| \leq |Z| - (k-1)/2 - (k-4) \leq (k^2 - 4k + 11)/2.$$

Because $|N_{G'_{k-2}}(Z'')| = k(k-3)/2$ and $k \geq 75$, $|Z''| \leq |N_{G'_{k-2}}(Z'')|$. If $H(Z)$ has no vertex that is not adjacent to the edges of $H(Z'')$, then the edges of R join vertices of $H(Z'')$ with each other. We have that $|Z''| \leq k(k-1)/2 - k = k(k-3)/2$ and $|Z''| \leq |N_{G'_{k-2}}(Z'')|$.

Suppose that $H(Z'')$ has at least $k+1$ vertices. Then R has at least $k+1$ edges and $|Z'| \leq |Z| - (k+1) \leq k(k-3)/2$. As $|N_{G'_{k-2}}(Z'')| \geq (k+1)(k-3)/2$, we get that $|Z''| \leq |N_{G'_{k-2}}(Z'')|$.

Case c). $q \geq k+1$. Then $H(Z'')$ has at least $k+1$ vertices. We have that R has at least $k+1$ edges and $|Z'| \leq |Z| - (k+1) \leq k(k-3)/2$. Because $|N_{G'_{k-2}}(Z'')| \geq (k+1)(k-3)/2$, we get that $|Z''| \leq |N_{G'_{k-2}}(Z'')|$.

We conclude that for any $Z'' \subseteq Z'$, $|Z''| \leq |N_{G'_{k-2}}(Z'')|$. Hence, G'_{k-2} has a matching M of size r such that every vertex of Z' is saturated in M.

Clearly, M is a matching in G'_k as well. Recall that $R = x_1 \dots x_p x_1$ is a Hamiltonian cycle in $H(Z)$ and $U = \{u_{x_1x_2}, \dots, u_{x_{p-1}x_1}\}$. For $i \in \{1, \dots, p\}$, let t_i be the number of vertices in $\{v^1_{x_i}, \dots, v^s_{x_i}\}$ that are saturated in M. Because M is a matching in G'_{k-1}, $t_i \leq s-1$.

We prove that there is $j \in \{1, \dots, p\}$ such that $t_j < s-1$. Let q be the smallest positive integer such that $|Z| \leq q(q-1)/2$. The graph $H(Z)$ has at least q vertices. Suppose first that it has exactly q vertices. Then $p = q$ and $Z' = Z \setminus U$ has at most $p(p-1)/2 - p = p(p-3)/2$ vertices. Also $|N_{G'_{k-2}}(Z)| = p(k-3)/2$. If $p < k$, at least one vertex in $N_{G'_{k-2}}(Z)$ is not saturated and the statement holds. Let $p = k$. Then because G has no cliques of size k, $|Z| < k(k-1)/2$ and $|Z'| < k(k-3)/2$. We have that $|Z'| < |N_{G'_{k-2}}(Z)|$ and at least one vertex in $N_{G'_{k-2}}(Z)$ is not saturated. If $p \geq k+1$, then $|Z| = k(k-1)/2+1$. We have that $|Z'| \leq k(k-3)/2$ and $|N_{G'_{k-2}}(Z)| \geq (k+1)(k-3)/2$. Hence, the there is a non-saturated vertex in $N_{G'_{k-2}}(Z)$. Suppose now that $H(Z)$ has at least $q+1$ vertices. Then $p \geq q+1$ and $|Z'| \leq q(q-1)/2-(q+1) = q(q-3)/2-1$. As $|N_{G'_{k-2}}(Z)| \geq (q+1)(k-3)/2$, $|Z'| < |N_{G'_{k-2}}(Z)|$ if $q \leq k$. If $q \geq k+1$, then

$$|Z'| \leq |Z| - (k+2) \leq (k(k-1)/2+1) - (k+2) \leq k(k-3)/2 - 1.$$

Because $|N_{G'_{k-2}}(Z)| \geq (k+2)(k-3)/2$, we again have a non-saturated vertex in $N_{G'_{k-2}}(Z)$. We considered all cases and conclude that at least one vertex of $N_{G'_{k-2}}(Z)$ is not saturated in M. Hence, there is $j \in \{1, \ldots, p\}$ such that $t_j < s-1$. Without loss of generality we assume that $j = p$.

Because $v^1_{x_i}, \ldots, v^s_{x_i}$ have the same neighbourhoods, we assume without loss of generality that for $i \in \{1, \ldots, p\}$, $v^1_{x_i}, \ldots, v^{t_i}_{x_i}$ are saturated. For $i \in \{1, \ldots, q\}$ and $j \in \{1, \ldots, t_i\}$, denote by u^j_i the vertex of Z' such that $\{v^j_{x_i}, u^j_i\} \in M$. Notice that it can happen that $t_i = 0$ and we have no such saturated vertices. We define the path $P_i = v^1_{x_i} u^1_i v^2_{x_i} \ldots u^{s_i}_i v^{t_i+1}_{x_i}$ if $t_i \geq 1$ and let $P_i = v^1_{x_i}$ if $t_i = 0$ for $i \in \{1, \ldots, p\}$. Let

$$P = P_1 + v^{t_1+1}_{x_1} u_{x_1 x_2} v^1_{x_2} + \ldots + v^{t_{p-1}+1}_{x_{p-1}} u_{x_{p-1} x_p} v^1_{x_p} + P_p$$

and then form the cycle C from P by joining the end-vertices of P by

$$v^{t_p+1}_{x_p} v^{t_p+2}_{x_p} u_{x_p x_1} v^1_{x_1}$$

using the fact that $t_p \leq s-2$. We have that $Z \subseteq V(C) \subseteq Z \cup N_G(Z)$ and $v^{t_p+1}_{x_p} v^{t_p+2}_{x_p} \in E(C)$. It concludes Case 3 and the proof of the claim. $\square$

Let $Z \subseteq V(G'_k)$ be a set of size k'. Let

$$Z' = Z \cap \{u_{xy} | \{x, y\} \in E(G)\}.$$

If $Z' = \emptyset$, then Z is a clique and there is a cycle C in G'_k such that $Z \subseteq V(C)$. Suppose that $Z' \neq \emptyset$. By Claim, there is a cycle C' in G'_k such that $Z' \subseteq V(C') \subseteq Z' \cup N_G(Z')$ and C' has an edge $\{v^i_x, v^j_y\}$ for some $x, y \in V(G')$ and $i, j \in \{1, \ldots, s\}$. Let $\{u_1, \ldots, u_p\} = Z \setminus V(C')$. Notice that these vertices are pairwise adjacent and adjacent to v^i_x, v^j_y. We construct the cycle C from C' by replacing $\{v^i_x, v^j_y\}$ by the path $v^i_x u_1 \ldots u_p v^j_y$. It remains to observe that $Z \subseteq V(C) \subseteq Z \cup N_G(Z)$. $\square$

It now remains to present our negative result, stating that Cyclability, restricted to planar graphs, admits no polynomial kernels unless $\mathsf{NP} \subseteq \mathsf{co\text{-}NP/poly}$.

CHAPTER 7 KERNELIZATION LOWER BOUND FOR CYCLABILITY

As we have showed in Chapter 5, Cyclabilty becomes tractable, from the Parameterized Complexity point of view, when we restrict the inputs to be planar graphs. This is not the case for general graphs, as proved in Chapter 6.

We know (we also proved it in Section 2.2 of Chapter 2), that any problem in FPT admits a kernelization algorithm. However, a more interesting question is whether an FPT-problem admits a small sized kernel, meaning a kernel of polynomial or even linear size.

In this Chapter we prove our second negative result: The Cyclability problem does not admit a polynomial kernel unless $\mathsf{NP} \subseteq \mathsf{co}\text{-}\mathsf{NP}/\mathsf{poly}$. The assumption $\mathsf{NP} \not\subseteq \mathsf{co}\text{-}\mathsf{NP}/\mathsf{poly}$ is widely believed and is often used in theoretical computer science when trying to prove the unlikeliness of a statement. That is why our results implies that it is very unlikely for Cyclability to admit any polynomial kernel.

The above result indicates that the Cyclability problem does not follow the kernelization behavior of many other problems (see for example [9]), for which surface embeddability enables the construction of polynomial kernels.

Before proceeding to the proof of our last result, we need to introduce some further notation regarding the kernelization lower bound theory, which has been rapidly growing in the recent years (for more on kernelization lower bounds see [22] and [36]).

Definition 7.0.1. *Let $L \subseteq \Sigma^*$ be a language, let $\mathcal{R}$ be a polynomial equivalence relation on Σ^*, and let $Q \subseteq \Sigma^* \times \mathbb{N}$ be a parameterized problem. An* AND-cross-composition of L into Q *(with respect to $\mathcal{R}$) is an algorithm that, given t instances $x_1, x_2, \dots, x_t \in \Sigma^*$ of L belonging to the same equivalence class of $\mathcal{R}$, takes time polynomial in $\sum_{i=1}^{t} |x_i|$ and outputs an instance $(y, k) \in \Sigma^* \times \mathbb{N}$ such that:*

1. *the parameter value k is polynomially bounded in* $\max\{|x_1|, \dots, |x_t|\} + \log t$*,*
2. *the instance (y, k) is a* yes*-instance for Q if and only each instance x_i is a* yes*-instance for L for $i \in \{1, \dots, t\}$.*

It is said that L AND-cross-composes into Q *if a cross-composition algorithm exists for a suitable relation $\mathcal{R}$*

In particular, Bodlaender, Jansen and Kratsch [10] proved the following theorem, which is an analogue if the one that we presented in Section 2.2.5 of Chapter 2, where the assumption of an OR-cross-composition is replaced by that of an AND-cross-composition. Despite the relevance of the two statements, the original proof of the latter (which relies on the main result of Dracker [37]) is pretty involved and, thus, not presented here. A simpler proof of the result in [37] was given by Dell in [27].

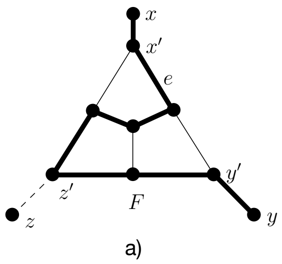
Theorem 7.0.1 ([10])**.** *If an* NP*-hard language L AND-cross-composes into the parameterized problem Q, then Q does not admit a polynomial kernelization unless* NP $\subseteq$ co-NP/poly*.*

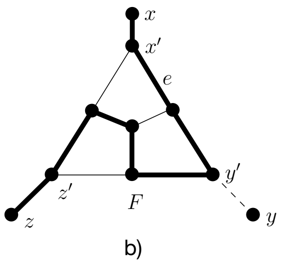
In this section we present our last result, i.e., a proof that it is unlikely that Cyclability, parameterized by k, admits a polynomial kernel when restricted to planar graphs. The proof uses the cross-composition technique introduced by Bodlaender, Jansen, and Kratsch in [10].

We consider the auxiliary Hamiltonicity with a Given Edge problem, which for a graph G and a given edge $e \in E(G)$, asks whether G has a Hamiltonian cycle that contains e. We use the following lemma.

Lemma 7.0.1. Hamiltonicity with a Given Edge *is* NP*-complete for cubic planar graphs.*

Proof. It was proved by Garey, Johnson and Tarjan in [46] that Hamiltonicity is NP-complete for planar cubic graphs. Let G be a planar cubic graph, and let v be an arbitrary vertex of G. Denote by x, y, z the neighbors of v in G. We replace v by a gadget F shown in Fig. 7.1. More precisely, we delete v, construct a copy of F and add theedges $\{x, x'\}$, $\{y, y'\}$ and $\{z, z'\}$. Denote by G' the obtained graph. Clearly, G' is a cubic planar graph. We claim that G is Hamiltonian if and only if G' has a Hamiltonian cycle that contains the edge e shown in Fig. 7.1.





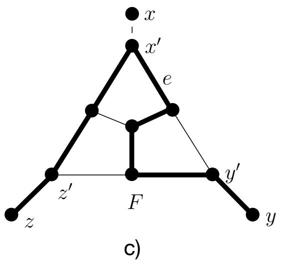


Figure 7.1: The gadget F; the edges of Hamiltonian cycles are shown by the bold lines.

Suppose that G has a Hamiltonian cycle C. Then C contains two edges incident to v. We construct the Hamiltonian cycle in G' by replacing these two edges by paths shown in Fig. 7.1. If C contains $\{x, v\}$ and $\{v, y\}$, then they are replaced by the path shown in Fig. 7.1 a), if C contains $\{x, v\}$ and $\{v, z\}$, then they are replaced by the path shown in Fig. 7.1 b) and if C contains $\{y, v\}$ and $\{v, z\}$, then we use the path shown in Fig. 7.1 c). It is easy to see that we obtain a Hamiltonian cycle that contains e. If G' has a Hamiltonian cycle, then it is straightforward to see that G is Hamiltonian as well. □

We are now ready to prove Theorem 1.3.5.

Proof of Theorem 1.3.5. We construct an AND-cross-composition of Hamiltonicity with a Given Edge. By Lemma 7.0.1, the problem is NP-complete.

We assume that two instances (G, e) and (G', e') of Hamiltonicity with a Given Edge are equivalent if $|V(G)| = |V(G')|$. Let (G_i, e_i) for $i \in \{1, \dots, t\}$ be equivalent instances of Hamiltonicity with a Given Edge, $|V(G_i)| = n$. We construct the graph G as follows (see Fig. 7.2).

i) Construct disjoint copies of $G_1, \dots, G_t$.

ii) For each $i \in \{1, \dots, t\}$, subdivide e_i twice and denote the obtained vertices by u_i, v_i.

iii) For $i \in \{1, \dots, t\}$, construct an edge $\{v_i, u_{i+1}\}$ assuming that $u_{n+1} = u_1$.

It is straightforward to see that G is a cubic planar graph.

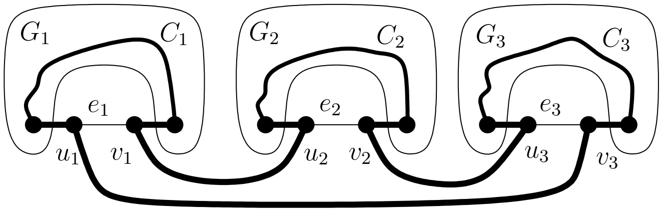


Figure 7.2: The construction of G for $t = 3$; the edges of a Hamiltonian cycle in G are shown by the bold lines.

We claim that G is $n+2$-cyclable if and only if (G_i, e_i) is a yes-instance of Hamiltonicity with a Given Edge for every $i \in \{1, \dots, t\}$. If every G_i has a Hamiltonian cycle C_i that contains e_i, then G is Hamiltonian as well; the Hamiltonian cycle in G is constructed from $C_1, \dots, C_t$ as it is shown in Fig. 7.2. Since G is Hamiltonian, G is $n+2$-cyclable. Suppose now that G is $n+2$-cyclable. Let $i \in \{1, \dots, t\}$. Consider $X = V(G_i) \cup \{u_i, v_i\}$. Because $|X| = n+2$, G has a cycle C that goes trough all the vertices of X. It remains to observe that by the removal of the vertices of $V(G) \setminus V(G_i)$ and by the addition of the edge e_i, we obtain from C a Hamiltonian cycle in G_i that contains e_i. $\square$

We have proved that it is unlikely that we can apply any efficient preprocessing routine for the Cyclability problem, even when restricted to the class of planar graphs.

The next, and last, chapter of this thesis contains an overview of our results and some suggestion for further research.

CHAPTER 8 CONCLUSION

As a conclusion of this thesis, we briefly review our results and suggest some directions for further research.

8.1 Our results

Let us quickly review our results:

- In Chapter 5, we construct an algorithm for solving Cyclability on planar graphs. Actually, we solve a slightly more general version with annotated vertices, which we call PAC. For the analysis of our algorithm we need the results presented in Chapters 3 and 4, which we think are of independent interest:
  - In Chapter 3, we prove a series of combinatorial results about cyclic linkages which enable us to use the irrelevant vertex technique.
  - In Chapter 4, we design a, somehow unusual, dynamic programming routine for efficiently solving the Cyclability problem on graphs of bounded treewidth.

- In Chapter 6 we prove that it is unlikely that Cyclability is in FPT by giving a parameterized reduction from the Clique problem to the problem that is complementary to Cyclability.
- In Chapter 7, we prove that Cyclability, even when restricted to planar graphs, admits no polynomial kernelization algorithm unless NP $\subseteq$ co-NP/poly.

8.2 Complexity of Cyclability and better running time

Another way to define the class NP (besides using non-deterministic Turing Machines), is in terms of existential quantification of polynomial relationships. More specifically: Let $L \subseteq \Sigma^*$ be a language. $L \in$ NP if and only if there is a polynomially decidable and polynomially balanced relation R, such that

$$L = \big\{x : \exists\, y\, [(x, y) \in R]\big\},$$

where R is called *polynomially decidable* if there is a deterministic Turing machine deciding the language $\big\{x; y : (x, y) \in R\big\}$ in polynomial time and is called *polynomially balanced* if $(x, y) \in R$ implies $|y| \leq |x|^k$ for some $k \geq 1$.

This, equivalent, definition supports the intuition that the class NP consists of problems where, given an input (corresponding to x) and a short (of size polynomial to $|x|$) *witness* or solution (corresponding to y) it is easy to decide (in polynomial time) if the given solution verifies that the input is valid for the problem.

For example, for the Hamiltonicity problem, given an input graph $G = (V, E)$ with $V = \{v_1, \dots, v_n\}$, a witness $w = (v_{i_1}, \dots, v_{i_n})$, is a permutation of the vertices in V (w is clearly of polynomial size) and the relation R corresponds to checking whether the elements of the set $\big\{\{v_{i_j}, v_{i_{j+1}}\} \mid j = 1, \dots, n-1\big\} \cup \{v_{i_n}, v_{i_1}\}$ exist in E. Clearly, this check can be done in polynomial time and this way we can prove that Hamiltonicity$\in$ NP.

The classical complexity of Cyclability**.** Notice that we have no proof (or evidence) that Cyclability is in NP. However, using a similar definition to the one we just gave for the class NP, we can directly place Cyclability to the second level of the polynomial hierarchy, and more precisely to the class Π_2^{P} (for detailed definitions see [75]).

Let $L \subseteq \Sigma^*$ be a language. $L \in$ NP if and only if there is a polynomially decidable and polynomially balanced relation R, such that

$$L = \{\, x : (\forall y)(\exists z)\big[\big(x, (y, z)\big) \in R\big)\big] \,\}.$$

We can place Cyclability in Π_2^{P} as follows: Given a graph $G = (V, E)$ and an integer k, the term $\forall\, y$ corresponds to every possible k-sized subset of V and the witness z corresponds to a permutation of some vertices of G. The polynomial relation, similarly to the the case of Hamiltonicity, checks whether there exist edges in E between all vertices that are successive in the permutation z (and between the last and the first vertex) and, additionally, whether z contains all the vertices of y. If both conditions are met, then x is a YES-instance for Cyclability. This classifies Cyclability in Π_2^{P}.

Although we do not have a proof, the previous arguments prompt us to conjecture the following:

Conjecture 8.2.1. Cyclability *is* Π_2^{P}*-complete.*

Moreover, while we have proved that Cyclability is co-W[1]-hard, we have no evidence regarding which level of the parameterized complexity hierarchy it belongs to (lower than the XP class). We find it an intriguing question whether there is some $i \geq 1$ for which Cyclability is W$[i]$-complete (or co-W$[i]$-complete).

Clearly, another challenging question is whether the, double exponential, parametric dependance of our FPT-algorithm can be substantially improved. We believe that this is not possible and we suspect that this issue might be related to Conjecture 8.2.1.

8.3 Generalizations

Another direction of research is to investigate whether Cyclability is in FPT on more general graph classes.

Actually, all results that were used for our algorithm can be extended on graphs embeddable on surfaces of bounded genus – see [47, 28, 80, 82, 60]

– and yield an FPT-algorithm on such graphs (with worst time bounds). We believe that this is still the case for graph classes excluding some fixed graph as a minor. However, in our opinion, such an extension, even though possible, would be too technically involved. Therefore, we state the following

Conjecture 8.3.1. Cyclability *is in* FPT *when restricted to the class of graphs embeddable on surfaces of bounded genus and to classes excluding some fixed graph as a minor.*

A research direction that is also very interesting is to define and study, both from a combinatorial and from an algorithmic point of view, problems similar to Cyclability, where the pattern of the linkages we look for is not a cycle but some other graph H.

It would be interesting to study the following problem

H-Linkability
Input: A graph G (*host*), a graph H (*pattern*) and an integer k.
Question: Is it true that, for every k-element subset S of $V(G)$, there is a topological minor of G that contains all the vertices of S and is isomorphic to H?

This, very general, problem can be associated with a cyclability-like parameter on graphs: We say that a graph $G = (V, E)$ is (H, k)*-linkable* if for every k-element subset S of V, there exists a topological minor of G that contains all the vertices of S and is isomorphic to H. The H*-linkability* of G is defined to be the greatest integer k for which G is (H, k)*-linkable*.

From the point of view of structural properties, the property of a graph having H-linkability equal to k can provide various kinds of information about the structure of G, depending on H and k. For example:

- If k is big (for example $k = \Omega(n)$), graph G being (H, k)*-linkable* could give some information (depending on the pattern H) of the global structure of G.

- On the other hand, if k is very small, graph G being (H, k)*-linkable* could tell something about local properties of G.

- Given two patterns H_1 and H_2 and two integers k_1 and k_2, we could study a question of the following form: Given that graph G is (H_1, k_1)-*linkable*, is it true that it is (H_2, k_2)-*linkable*? It feels like answering this kind of questions could give some insight about the structure of the graph under study and one could try a variety of mixtures of patterns and integers.

Of course, this problem is computationally hard when examined from the classical complexity theory point of view, as for $k = n$ and H being any cycle of length at most n, it is equivalent to the Hamiltonicity problem.

It would be reasonable to study H-Linkability parameterized by k, as we have done with Cyclability. Unfortunately, the structure of a cyclic linkage is essential for some parts of our work and it seems that for studying the, much more general, H-Linkability problem new ideas and techniques need to be introduced.

BIBLIOGRAPHY

[1] F. Abu–Khzam, M. Fellows, M. Langston, W. Suters, *Crown structures for vertex cover kernelization*, Theory Comput. Syst., 41(3), 411–430 (2007)

[2] I. Adler, F. Dorn, F. Fomin, I. Sau, D. Thilikos, *Fast minor testing in planar graphs*, ESA 2010, 18th Annual European Symposium (1). Lecture Notes in Computer Science, vol. 6346, 97–109 (2010)

[3] I. Adler, S. Kolliopoulos, P. Krause., D. Lokshtanov, S. Saurabh, D. Thilikos, *Irrelevant vertices for the disjoint paths problem*, J. Comb. Theory, Series B 122: 815–843 (2017)

[4] R. Aldred, S. Bau, D. Holton, B. McKay, *Cycles through 23 vertices in 3-connected cubic planar graphs*, Graphs and Combinatorics 15(4), 373–376 (1999)

[5] S. Arora, B. Barak, *Computational Complexity : A Modern Approach*, Cambridge University Press (2009)

[6] S. Arnborg, D. Corneil, A. Proskurowski, *Complexity of finding embeddings in a k-tree*, SIAM Journal on Matrix Analysis and Applications, 8 (2), 277–284 (1987)

[7] S. Arnborg, J. Lagergren, D. Seese, *Easy problems for tree-decomposable graphs*, Journal of Algorithms, 12, 308–340 (1991)

[8] H. Bodlaender, *A linear-time algorithm for finding tree-decompositions of small treewidth*, SIAM J. Comput., 25(6), 1305–1317 (1996)

[9] H. Bodlaender, F. Fomin, D. Lokshtanov, E. Penninkx, S. Saurabh, D. Thilikos, *(meta)Kernelization*, Proceedings of the 2009 50th Annual IEEE Symposium on Foundations of Computer Science. pp. 629–638. FOCS '09, IEEE Computer Society, Washington, DC, USA (2009)

[10] H. Bodlaender, B. Jansen, S. Kratsch, *Kernelization lower bounds by cross–composition*, SIAM J. Discrete Math., 28(1), 277–305 (2014)

[11] B. Bollobás, *Modern Graph Theory* Springer-Verlag New York Inc. (2002)

[12] A. Bondy, U. Murty, *Graph Theory : An Advanced Course* Springer London Ltd (2011)

[13] R. Borie, R. Parker, C. Tovey, *Automatic generation of linear-time algorithms from predicate calculus descriptions of problems on recursively constructed graph families*, Algorithmica, 7, 555–581 (1992)

[14] J. Chen, I.A. Kanj, W. Jia, *Vertex cover: Further observations and further improvements*, J. Algorithms, 41(2), 280–301 (2001)

[15] J. Chen, I. Kanj, G. Xia, *Improved upper bounds for Vertex Cover*. Theor. Comput. Sci., 411 (40–42), 27–88 (2010)

[16] M. Chlebík and J. Chlebíková, *Crown reductions for the minimum weighted vertex cover problem*, Discrete Applied Mathematics, 156(3), 292–312 (2008)

[17] V. Chvátal, *New directions in hamiltonian graph theory*, New Directions in the Theory of Graphs, Ed.: F. Harary, Academic Press, New York, 65 – 95 (1973)

[18] S. Cook, *The complexity of theorem proving procedures*, Proceedings of the Third Annual ACM Symposium on Theory of Computing, 151–158 (1971)

[19] T. Cormen, C. Leiserson, R. Rivest, C. Stein, *Introduction to Algorithms*, MIT Press & McGraw-Hill, 2nd edition (2001)

[20] B. Courcelle, *The monadic second–order logic of graphs I: Recognizable sets of finite graphs*, Information and Computation, 85(1), 12–75 (1990)

[21] B. Courcelle, J. Engelfriet, *Graph Structure and Monadic Second-Order Logic – A Language-Theoretic Approach*, Encyclopedia of mathematics and its applications, vol. 138. Cambridge University Press (2012)

[22] M. Cygan, F.V. Fomin, L. Kowalik, D. Lokshtanov, D. Marx, M. Pilipczuk, M. Pilipczuk, S. Saurabh, *Parameterized Algorithms*, Springer (2015).

[23] M. Cygan, J. Nederlof, M. Pilipczuk, M. Pilipczuk, J.M.M van Rooij, J.O Woj- taszczyk, *Solving connectivity problems parameterized by treewidth in single exponential time*, Proceedings of the 52nd Annual Symposium on Foundations of Computer Science (FOCS), 150–159 (2011)

[24] S. Dasgupta, C. Papadimitriou, U. Vazirani, *Algorithms*, McGraw-Hill Education, Europe (2011)

[25] A. Dawar, M. Grohe, Stephan Kreutzer, *Locally excluding a minor*, Logic in Computer Science (LICS'07), pages 270–279, IEEE Computer Society, 2007

[26] A.Dawar, S. Kreutzer, *Domination problems in nowhere-dense classes*, IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2009), pages 157–168, 2009.

[27] H. Dell, *AND-compression of NP-complete Problems: Streamlined Proof and Minor Observations*, International Symposium on Parameterized and Exact Computation (IPEC), 184-195 (2014)

[28] E. Demaine, M. Hajiaghayi, D. Thilikos, *The bidimensional theory of bounded–genus graphs*, SIAM J. Discrete Math., 20(2), 357–371 (2006)

[29] R. Diestel, *Graph theory*, Graduate Texts in Mathematics, vol. 173. Springer, Heidelberg, 4th edition (2010)

[30] G. Dirac, *n abstrakten Graphen vorhandene vollständige 4-Graphen und ihre Unterteilungen*, Math. Nachr. 22, 61–85 (1960)

[31] R. Downey, M. Fellows, *Parameterized complexity*, Springer–Verlag, New York (1999)

[32] R. Downey, M. Fellows, *Fixed–parameter tractability and completeness III: Some structural aspects of the W–hierarchy*, Complexity theory, 191–225. Cambridge Univ. Press, Cambridge (1993)

[33] R. Downey, M. Fellows, *Fixed–parameter tractability and completeness*, 21st Manitoba Conference on Numerical Mathematics and Computing (Winnipeg, MB, 1991), vol. 87, pp. 161–178 (1992)

[34] R. Downey, M. Fellows, *Fixed–parameter tractability and completeness I: Basic results*, SIAM J. Comput., 24(4), 873–921 (1995)

[35] R. Downey, M. Fellows, *Fixed–parameter tractability and completeness II: On completeness for $W[1]$*, Theoretical Computer Science, 141(1–2), 109–131 (1995)

[36] R. Downey, M. Fellows, *Fundamentals of Parameterized Complexity*, Texts in Computer Science, Springer (2013)

[37] A. Drucker, *New limits to classical and quantum instance compression*, Proceedings of the 53rd Annual Symposium on Foundations of Computer Science (FOCS), 609 618, IEEE (2012)

[38] P. Erdős, *Remarks on a paper of Pósa*, Magyar Tud. Akad. Mat. Kutató Int. Közl. 7, 227–229 (1962)

[39] M. Fellows, M. Langston, *Nonconstructive tools for proving polynomial-time decidability*, Journal of the ACM, 35 (3): 727–739 (1988)

[40] E. Flandrin, H. Li, A. Marczyk, M. Woźniak, *A generalization of dirac's theorem on cycles through k vertices in k–connected graphs*, Discrete mathematics, 307(7), 878–884 (2007)

[41] E. Flandrin, E. Györi, H. Li, J. Shu, *Cyclability in k-connected $K_{1,4}$-free graphs*, Discrete Mathematics 310 (20), 2735–2741 (2010)

[42] J. Flum, M. Gröhe, *Parameterized Complexity Theory*, Springer (2006)

[43] F. Fomin, P. Golovach, D. Thilikos, *Contraction obstructions for treewidth*, J. Comb. Theory, Ser. B 101(5), 302–314 (2011)

[44] M. Frick, M. Gröhe, *The complexity of first-order and monadic second-order logic revisited*, Annals of Pure and Applied Logic, 130 (1–3), 3–31 (2004)

[45] M. Garey, D. Johnson, *Computers and Intractability : A Guide to the Theory of* NP*–completeness*, W.H.Freeman and Co Ltd (1979)

[46] M. Garey, D. Johnson, R. Tarjan, *The planar Hamiltonian circuit problem is NP–complete*, SIAM J. Comput., 5(4), 704–714 (1976)

[47] J. Geelen, R. Richter, G. Salazar, *Embedding grids in surfaces*, European J. Combin. 25(6), 785–792 (2004)

[48] P. Giblin *Graphs, Surfaces and Homology* Cambridge University Press (2010)

[49] P. Golovach, M. Kamiński, D. Paulusma, D. Thilikos, *Induced packing of odd cycles in a planar graph*, 20th International Symposium on Algorithms and Computation (ISAAC 2009), Volume 5878 of LNCS, pages 514–523, Springer, Berlin, 2009.

[50] M. Gröhe, K. Kawarabayashi, D. Marx, P. Wollam, *Finding topological subgraphs is fixed-parameter tractable*, Proceedings of 43rd ACM Symposium on Theory of Computing (STOC '11), 79–88 (2011)

[51] J. Gross , T. Tucker, *Topological Graph Theory*, Dover Publications Inc. (1987)

[52] M. Grötschel, *Hypohamiltonian facets of the symmetric travelling salesman polytope*, Zeitschrift für Angewandte Mathematik und Mechanik 58, 469–471 (1977)

[53] Q.P. Gu, H. Tamaki, *Improved bounds on the planar branchwidth with respect to the largest grid minor size*, Algorithms and Computation – 21st International Symposium, (ISAAC 2010), 85–96 (2010)

[54] D. Holton, B. McKay, M. Plummer, C. Thomassen, *A nine-point theorem for 3-connected graphs*, Combinatorica, 2, 53–62 (1982)

[55] R. Karp, *On the computational complexity of combinatorial problems*, Networks, Volume 5, pages 45–68 (1975)

[56] K. Kawarabayashi, Y. Kobayashi, *The induced disjoint paths problem*, 13th Conference on Integer Programming and Combinatorial Optimization (IPCO 2008), volume 5035 of LNCS, pages 47–61, Springer, Berlin, 2008.

[57] K. Kawarabayashi, Y. Kobayashi, B. Reed, *The disjoint paths problem in quaratic time*, J. Combin. Theory, Series B 102(2): 424–435 (2012)

[58] K. Kawarabayashi, Y. Kobayashi, *Algorithms for finding an induced cycle in planar graphs and bounded genus graphs*, 20th ACM-SIAM Symposium on Discrete Algorithms (SODA 2009), pages 1146–1155, ACM-SIAM 2009.

[59] Ken-ichi Kawarabayashi, B. Reed, *Odd cycle packing*, 42nd ACM Symposium on Theory of Computing (STOC 2010) pages 695–704, ACM 2010.

[60] K. Kawarabayashi, P. Wollan, *A shorter proof of the graph minor algorithm: The unique linkage theorem*, Proceedings of the Forty–second ACM Symposium on Theory of Computing. pp. 687–694. STOC '10, ACM, New York, NY, USA (2010)

[61] J. Kleinberg, E. Tardos, *Algorithm Design*, Pearson Education (US) (2005)

[62] A. Langer, F. Reidl, P. Rossmanith, S. Sikdar, *Evaluation of an MSO-solver*, Proceedings of the Fourteenth Workshop on Algorithm Engineering and Experiments, ALENEX 2012, The Westin Miyako, Kyoto, Japan, January 16, 2012, ed. by D. Bader, P. Mutzel (SIAM/Omnipress, Philadelphia, 2012), 55–63

[63] L. Levin, *Universal search problems* (in Russian), Problems of Information Transmission (in Russian), 9 (3), 115–116 (1973)

[64] J. Kneis, A. Langer, *A Practical Approach to Courcelle*'s *Theorem*, Electronic Notes on Theoretical Computer Science, vol. 251, Elsevier Amsterdam, 65–81 (2009)

[65] J. Kneis, A. Langer, P. Rossmanith, *Courcelle*'s *Theorem—a game theoretic approach*, Discrete Optim., 8(4), 568–594 (2011)

[66] M. Kramer, J. van Leeuwen, *The complexity of wire–routing and finding minimum area layouts for arbitrary VLSI circuits*, Advances in Comp. Research, 2: 129–146 (1984)

[67] R. Lewis, C. Papadimitriou, *Elements of the Theory of Computation*, Pearson Education (US), 2nd edition (1997)

[68] K. Menger, *Zur allgemeinen Kurventheorie*, Fundamenta Mathematicae, 10, 96–115 (1927)

[69] M. Middendorf, F. Pfeiffer, *On the complexity of the disjoint paths problem*, Combinatorica, 13(1): 97–107 (1993)

[70] M. Mitzenmacher, E. Upfal, *Probability and Computing : Randomized Algorithms and Probabilistic Analysis*, Cambridge University Press (2005)

[71] R. Motwani, P. Raghavan, *Randomized Algorithms*, Cambridge University Press, 1995

[72] M. Müller, *Randomized Approximations of Parameterized Counting Problems*, Parameterized and Exact Computation, Volume 4169 of the series LNSC, pages 50–59, Springer, Berlin (2006)

[73] R. Niedermeier, *Invitation to fixed-parameter algorithms*, Habilitation thesis (Sep 2002)

[74] T. Nishizeki, N. Chiba, *Planar Graphs: Theory and Algorithms*, Dover Publications Inc. (2008)

[75] Christos H. Papadimitriou, *Computational Complexity*, Addison–Wesley (1994)

[76] L. Perkovic, B.A. Reed, *An improved algorithm for finding tree decompositions of small width*, Int. J. Found. Comput. Sci. 11(3), 365–371 (2000)

[77] M. Plummer, E. Győri, *A nine vertex theorem for 3-connected claw-free graphs*, Studia Scientiarum Mathematicarum Hungarica, 38(1), 233–244 (2001)

[78] N. Robertson, P. Seymour, *Graph minors III: Planar tree-width*, J. Combin. Theory, Series B, 36 (1): 49–64 (1984)

[79] N. Robertson, P. Seymour, *Graph Minors X: Obstructions to Tree-decomposition*, J. Combin. Theory, Series B 52(2), 153–190 (1991)

[80] N. Robertson, P. Seymour, *Graph Minors XIII: The disjoint paths problem*, J. Combin. Theory, Series B 63(1), 65–110 (1995)

[81] N. Robertson, P. Seymour, *Graph minors XVI: Excluding a non-planar graph*, J. Combin. Theory, Series B 89, 43–76 (2003)

[82] N. Robertson, P. Seymour, *Graph minors XXI: Graphs with unique link-ages.*, J. Combin. Theory, Series B 99(3), 583–616 (2009)

[83] N. Robertson, P. Seymour, *Graph minors XXII: Irrelevant vertices in linkage problems*, Journal of Combinatorial Theory, Series B 102(2), 530 – 563 (2012)

[84] N. Robertson, P. Seymour, *Graph minors XXIII: Nash-Williams' immersion conjecture*, Journal of Combinatorial Theory, Series B 100(2), 181 – 205 (2010)

[85] R. Rubinfeld, A. Shapira, *Sublinear Time Algorithms*, SIAM Journal on Computing, Volume 25, Issue 4, pages 1562–1588 (2011)

[86] M. Sipser, *Introduction to the Theory of Computation*, Cengage Learning, 3rd edition (2012)

[87] R. Solovay, V. Strassen, *A Fast Monte-Carlo Test for Primality*, SIAM Journal on Computing, Volume 6, Issue 1, pages 84–84 (1976)

[88] C. Thomassen , B. Mohar, *Graphs on Surfaces*, John Hopkins University Press (2001)

[89] A.M. Turing, *On Computable Numbers, with an Application to the Entscheidungs problem*, Proceedings of the London Mathematical Societ,. 42, 230–265 (1937)

[90] V.V. Vazirani, *Approximation Algorithms*, Springer–Verlag Berlin and Heidelberg GmbH and Co. KG (2001)

[91] J. Vygen, *NP-completeness of some edge–disjoint paths problems.*, Discrete Appl. Math., 61(1): 83–90 (1995)

[92] M. Watkins, D. Mesner, *Cycles and connectivity in graphs*, Canad. J. Math, 19, 1319–1328 (1967)

[93] D. Williamson, D. Shmoys, *The Design of Approximation Algorithms*, Cambridge University Press (2011)